Topological and Geometric Healing on Solid Models

Theoretical aspects, implementation and testing of Healing Routines on solid models for numerical simulations

Felix Frischmann

Master's thesis
for the Master of Science program Computational Mechanics

Author:	Felix Frischmann
Matriculation no.:	03273642
Supervisor:	Dipl.-Ing. Christian Sorger M.Sc.
	Prof. Dr. rer. nat. Ernst Rank
Date of issue:	01. August 2011
Date of submission:	18. November 2011

Involved Organisations



Chair for Computation in Engineering
Faculty of Civil Engineering and Geodesy
Technische Universität München
Arcisstraße 21
D-80333 München

Declaration

With this statement I declare, that I have independently completed this Master's thesis. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

München, June 14, 2013

Felix Frischmann

Felix Frischmann
Oberföhringer Str. 236a
D-81925 München
e-Mail: frischmann@bv.tum.de

Abstract

This thesis presents geometrical and topological healing algorithms and routines for geometry models. Issues of practical implementation in modern computational environments are part of this thesis as well as theoretical backgrounds on healing and modelling. Healing algorithms and procedures are necessary whenever geometrical input data can not directly be used for further analysis processes or any kind of downstream computational methods.
Numerical simulations are in most cases performed on discrete models approximating the real physical domain. A widely used method is the Finite-Elements Method (FEM). To apply the FE-technique it is necessary to produce a set of finite elements, which is done in most cases by meshing techniques. There exists a wide range of efficient and sofisticated algorithms and software components for mesh generation. Nevertheless mesh generation can be a very expensive and time consuming task. Problems related to mesh generation often arise due to geometrical or topological inconsistencies in the model description. Processing time and work load can be strongly affected by these model errors. Those inconsistencies or geometrical errors can have different sources like inaccuracies introduced by the modeler, inaccuracies introduced due to tolerances and floating point problems of computer architectures, errors introduced while transforming into different file formats and many more. Depending on the robustness of the meshing algorithms these errors and inaccuracies can be a severe problem and in the worst case request data manipulation by hand. But not only meshing techniques depend on correct and consistent geometrical information. Over the last years several meshing-free analysis techniques for computational methods were presented, for example isogeometric FEM analysis. The basic idea of these methods is to use the geometrical description also as basis for further numeric techniques (e.g. FEM). For such kind of methods it is even more important, that the geometrical description is free of errors and inconsistencies since no intermediate step like meshing is able to erase them. This is the reason why only few meshing-free techniques are implemented until now in the wide range of industrial applications.
The reader will first be introduced in aspects of geometric modelling and computational description of geometrical information. In the second part of this work the reader will be provided with a general overview over aspects, techniques and applications of geometrical and topological healing. The third and fourth part of this thesis presents the code framework TUM.GeoFrame (formerly Visual DoMesh) programmed, tested and maintained by the chair of Computation in Engineering (CiE) of Technische Universität München. The core of this framework consists of a topology and geometry kernel and a grahical user interface (GUI) and provides a suitable database for different numerical processes like mesh generation or numerical solvers. Within the scope of this work the framework was extended by healing algorithms and routines. In the last part the reader will be presented a sample of examples, demonstrating the performance, efficiency and stability of the healing routines as well as possible drawbacks and persistent problems, which were transformed into ideas for ongoing development in the outlook.

Contents

1 Introduction **1**
- 1.1 Introduction and Motivation . . . 1
- 1.2 Goals of this work . . . 2

2 Geometric Modelling and Numerical Simulation **3**
- 2.1 The Engineering Workflow . . . 3
- 2.2 Geometric Modelling . . . 6
  - 2.2.1 Sets, relations and graphs . . . 6
  - 2.2.2 Spatial Structures . . . 6
  - 2.2.3 Constructive methods / Direct Representation Schemes . . . 8
  - 2.2.4 Standard Cell Enumeration Schemes . . . 9
  - 2.2.5 Space Trees . . . 10
  - 2.2.6 Boundary Representation Model - BREP . . . 12
  - 2.2.7 Description of geometric attributes and shapes of curves and faces . . . 15
- 2.3 File formats and exchange of geometric information . . . 28
- 2.4 Meshing and discretization . . . 34
- 2.5 Healing routines in the engineering workflow . . . 37

3 Theoretical aspects of healing routines and algorithms **39**
- 3.1 Modelling errors and healing approaches . . . 39
  - 3.1.1 Errors and inconsistencies of geometric models . . . 39
  - 3.1.2 Classifications of healing approaches . . . 45
- 3.2 Theoretical background on selected healing routines . . . 48
  - 3.2.1 Theoretical review on geometric healing . . . 48
  - 3.2.2 Theoretical review on topologic healing . . . 55

4 TUM.GeoFrame Framework **74**
- 4.1 The Existing TUM.GeoFrame Framework . . . 74
  - 4.1.1 The Topology Kernel - DataBase . . . 76
  - 4.1.2 The Geometry Kernel . . . 77
- 4.2 Implementation of Healing Routines in TUM.GeoFrame . . . 78
  - 4.2.1 Healer in the TUM.GeoFrame Processing Workflow . . . 80

5 Implemented Healing Routines and Algorithms **83**
- 5.1 General Structure and Healing Settings . . . 83
  - 5.1.1 Concept and Implementation of Healing Tolerances . . . 85
- 5.2 Healing Routines and Processing Workflow . . . 87
- 5.3 Region Healing Routines and Algorithms . . . 90
  - 5.3.1 Region Healing Workflow . . . 91
  - 5.3.2 Wire Healing Routines . . . 92

5.3.3 Surface Healing Routines . . . 99
5.4 Extrusion Volume Healing Routines and Algorithms . . . 100
5.5 BREP Volume Healing Routines and Algorithms . . . 101
5.5.1 BREP Volume Healing Workflow . . . 101
5.5.2 BREP Volume Healing Routines . . . 102

6 Examples and Assessment of Healing Routines 106
6.1 Assessment of Algorithms and Routines . . . 106
6.2 Region Healing Examples . . . 106
6.2.1 Ex_01 Region Healing: Simple Surface Patch . . . 107
6.2.2 Ex_02 Region Healing: Simple Surface Patch . . . 112
6.2.3 Ex_03 Region Healing: Complex Trimmed Surface Shape - Violin . . . 113
6.2.4 Ex_04 Region Healing: Complex Surface Shape - wind turbine . . . 116
6.2.5 Ex_05 Region Healing: Complex Surface Shape - ship model . . . 117
6.2.6 Ex_06 Region Healing: Wire sorting . . . 120
6.3 Extrusion Volume Healing Examples . . . 123
6.3.1 Ex_07 Extrusion Volume Healing . . . 123
6.4 BREP Volume Healing Examples . . . 127
6.4.1 Ex_08 BREP Volume Healing: Simple BREP example . . . 128
6.4.2 Ex_09 BREP Volume Healing: Simple BREP example . . . 131
6.4.3 Ex_10 BREP Volume Healing: BREP model with inner holes . . . 133
6.4.4 Ex_11 BREP Volume Healing: non-linear BREP model . . . 134

7 Summary, Conclusions and Outlook 135
7.1 Summary and Conclusions . . . 135
7.2 Outlook and further development . . . 136

Main Part

Chapter 1

Introduction

1.1 Introduction and Motivation

Numerical methods and simulations play an important role in nowadays tasks of engineering. The simulation of physical processes is a crucial part of modern design and production processes throughout the different disciplines of engineering. To simulate physical effects of structures, fluids or materials it is often required to solve partial differential equations over a physical domain, describing the body or component of interest.
The complexity of physical effects taking place as well as the complexity of the geometric objects motivates the usage of numerical methods since in most cases analytic solutions are not applicable or can not be applied in a stable and efficient manner. Numerical solutions can be found by discretising the physical domain and with that reformulate the appearing differential equations on the discrete set of unknowns. A widely used method is the Finite-Element Method (FEM), which approximates the physical domain by a set of finite elements. An advantage of FEM is the relatively simple reformulation of differential equations on the discrete set of points over the domain. Every node location holds a set of unknows related to the fisible kinematic constraints. There exists a wide range of software and code frameworks for FEM techniques throughout the different disciplines of engineering. The long history of development and maintainace often guarantees a high level of stability and efficiency.
With the improvement of practical implementations an increase of complexity concerning problems and models went along. The reasons for persistent challenges for practical implementations are a wide range of aplications with different aspects and intrinsic characteristics, increased demands concerning accurate description of underlying physical effects and advanced possibilites to describe complex shapes. In particular the progresses in the fields of geometric modelling are significant. A state of the art way to describe geometrical objects or physical domains is the so called free-from formulation based on NURBS and other mathematical models, which will be described within this thesis. A basic knowledge of these methods and concepts is necessary and useful since many practical problems of numerical methods like FEM occur due to the geometric representations. The progresses within the last years concerning geometrical description and modelling went along with problems in the downstream processes like meshing and solution techniques opening a gap in the engineering workflow. Closing this gap is an important issue for the community of computational engineering.
The gap in the workflow from modelling towards downstream methods affects significantly mesh generation processes for example. But even meshing-free techniques can suffer from inaccurate or defective description of geometry and topology. Errors in the geometric model can influence efficiency and stability of all downstream processes. In the worst case they might request intermediate steps to validate the geometric description. The importance of these intermediate healing routines becomes apparent since up to 70 percent of the processing time for problems

of modern engineering are related to modelling, meshing and possible prior steps making the geometric description meshable, the so called healing step (Michael Hardwick and Robert Clay, Sandia National Laboratories).
The focus of this thesis lies exactly in the necessary or useful intermediate healing routines. The author will reveal gaps in the engineering workflow and provide the reader with information about the possible sources of these errors and inconsistencies in the models. Efficient and stable algorithms are required to perform healing routines. The author will give an overview over different approaches and concepts to deal with defected geometry and provide downstream algorithms and applications with suitable models. Many approaches and techniques are already successfully implemented in industrial software applications. Nevertheless the complexity of models and geometry is still a big challenge for healing routines. The author will show persistent problems concerning healing and validating of geometrical input data. Especially non-meshing methods like the isogeomertric analysis which are related directly to the geometric description to reformulate and discretize the problem still suffer significantly from model errors. The big problem arising in this kind of methods is that no intermediate meshing algorithm is present to erase the errors in the model. Other problems result from the mathematical formulation of NURBS itself which will be described more in detail in the following chapters. Especially for this kind of numerical techniques healing procedures have to be very robust and stable.
This work will not only describe and analyse the problems and existing software applications to deal with them. The implementation of healing routines and algorithms in an existing framework developed at the chair of Computaion in Engineering (CiE) at Technische Universität München will be presented to the reader. The performance, functionality and efficiency as well as persistent problems are described within this work.

1.2 Goals of this work

The goal of this work is to describe and present an implementation concept for healing algorithms and routines in form of an extension to an existing code framework called TUM.GeoFrame (formerly Visual DoMesh). The existing code is extended by functionalities for healing purposes to close the gap between input of geometric models provided by third party modelling software like CAD or Rhinoceros and downstream processes like mesh generation or numerical solvers within the TUM.GeoFrame framework.
The implementation of self-developed healing algorithms allows a high flexibility and optimization of the healing operations with reference to the data structure of the existing framework. The healing algorithms and operations are bundled into routines forming a sub-framework in TUM.GeoFrame. Within the scope of this thesis it was not possible to provide a complete and absolutely stable software applicable for all types of errors and defects. The purpose was rather to set up a structure for a healer sub-project in the framework and equip it with fundamental healing routines tested against a range of selected problems and examples. The selected examples demonstrate the performance of the various routines. Many of these examples were selected due to detected toplogical and geometric errors during mesh generation.
The demand for a very stable and efficient code for healing purposes requires a huge effort concerning debugging and maintainace of the code. The almost infinite number of possible errors, inconsistencies or geometric shapes makes intensive testing and constant improvement of codes and algorithms necessary. Therefore the code presented here will need further development after applying it to a wider range of examples and geometries. The author will mention further routines not yet implemented in the algorithms necessary to heal known problems and geometric errors.

Chapter 2

Geometric Modelling and Numerical Simulation

2.1 The Engineering Workflow

Many problems in engineering, in particular in civil engineering deal with interactions of bodies and structural elements and describe physical effects like deformations, stress states under loading, self weight or contact problems between different bodies. Picture 2.1 depicts the entire workflow of engineering product design, production, maintenance as well as numerical services in engineering like FEM analysis. The product cycles might include many subsystems and subtasks depending on size and complexity of the problem. Therefore efficient techniques for problem formulation, solution and interpretation are required to shorten production and service cycles. Automated processes and interactions are preferred. Documentation and quality control of the single steps is crucial in industry but also in research and development. This thesis will name problems arising in this workflow due to inconsistencies and errors which might lead to time-critical delays or cumbersome user interaction.

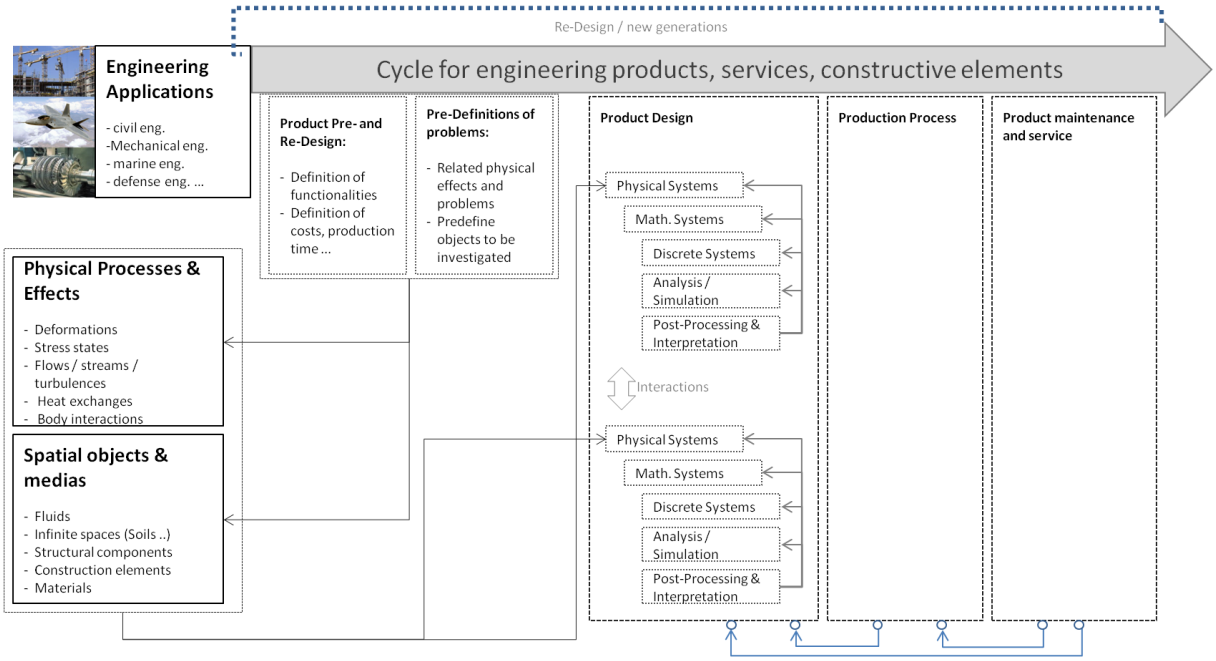


Figure 2.1: Entire production and design cycle in the engineering workflow

The problems of interest within the scope of this thesis are usually formulated over a physical domain forming structural elements or machinery parts, possibly including surrounding medias like water, air or soil. In general the physical domain has only to cover the elements and objects or range of infinite spaces which have an influence on the physical effects to be investigated. This step to reduce the physical system in the real world into a mathematical model is called idealization. Picture 2.2 shows two examples for idealization in tunneling and building construction.

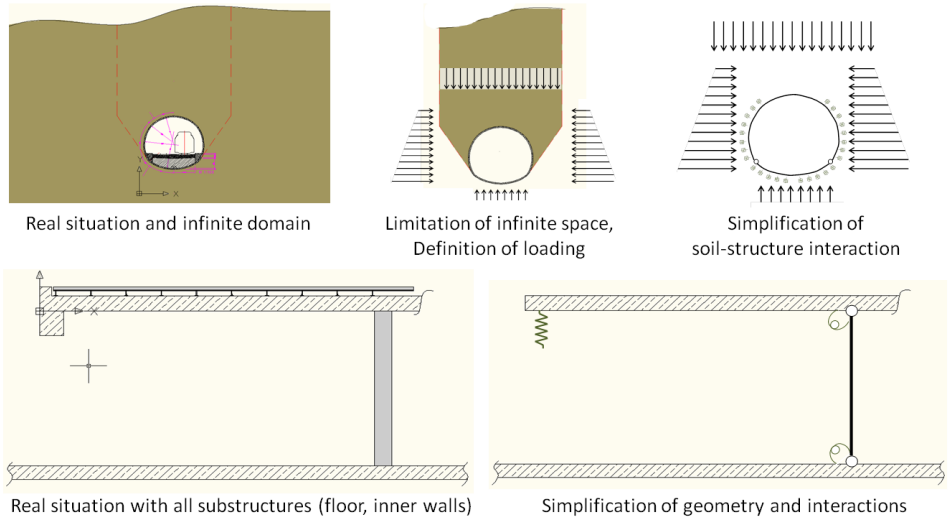


Figure 2.2: Idealization example in tunneling and building construction

The idealization step to obtain a suitable mathematical model for downstream processes includes several subtasks. The real problem has to be redefined on a mathematically described geometric model excluding or simplifying details with no significant influence on the physics. The simplification is important to make modelling and processing more efficient an derive solvable mathematical formulations for physical effects. Boundary conditions and body or subsystem interactions have to be formulated in a mathematical sense. All these steps are illustrated in the picture 2.2 above. To solve general types of problems defined on such mathematical models efficiently techniques like the Finite Element Method (FEM, see [25]) or Finite Cell Methods (FCM, see [35]) amongst others are widely used in modern engineering. Pure analytical solutions are usually only found for restricted simple problems with simple boundary conditions, physical domains and load cases. Numerical solution techniques are usually based on a discrete problem description to reformulate the partial differential equations which often arise due to the mathematical description of the physical effects. This step from the idealized mathematical model to a discrete model is called discretization and covers discretisation in space and time. Mesh generation is one possibility of discretisation providing a finite set of elements approximating the physical domain. The partial differential equations evolve into a finite set of equations. Picture 2.3 sketches the classical workflow from the physical system to the discrete solution.

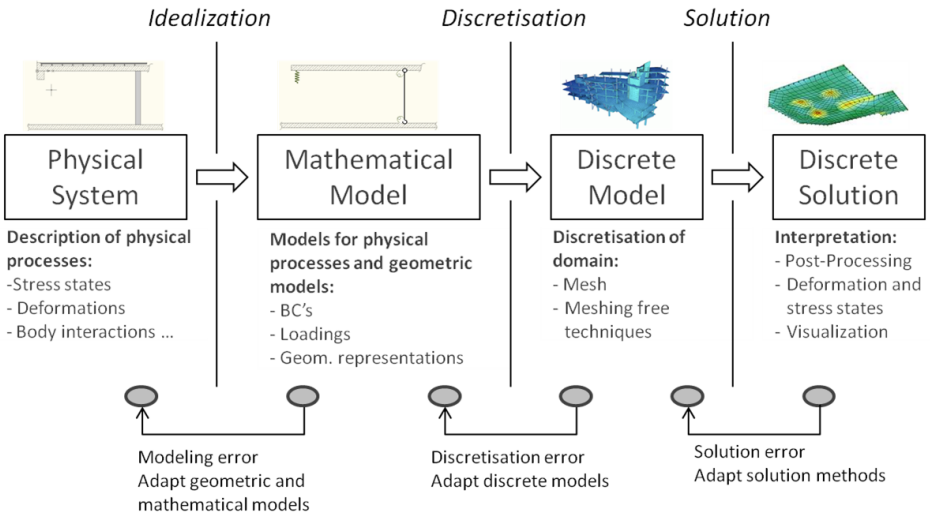


Figure 2.3: Pre-Processing, Processing and Post-Processing workflow

The entire workflow including information on the substeps is depicted in picture 2.4. The picture lays stress on the fact that the mathematical model does not only include mathematical descriptions of physical effects in means of formulary but also a suitable digital model for the geometry. This geometric model mathematically describes the real spatial objects like construction elements or machinery parts. The geometric model is in fact a basis for nearly all downstream processes. The workflow holds also backward relations since downstream processes influence the properties of the mathematical and geometric model.

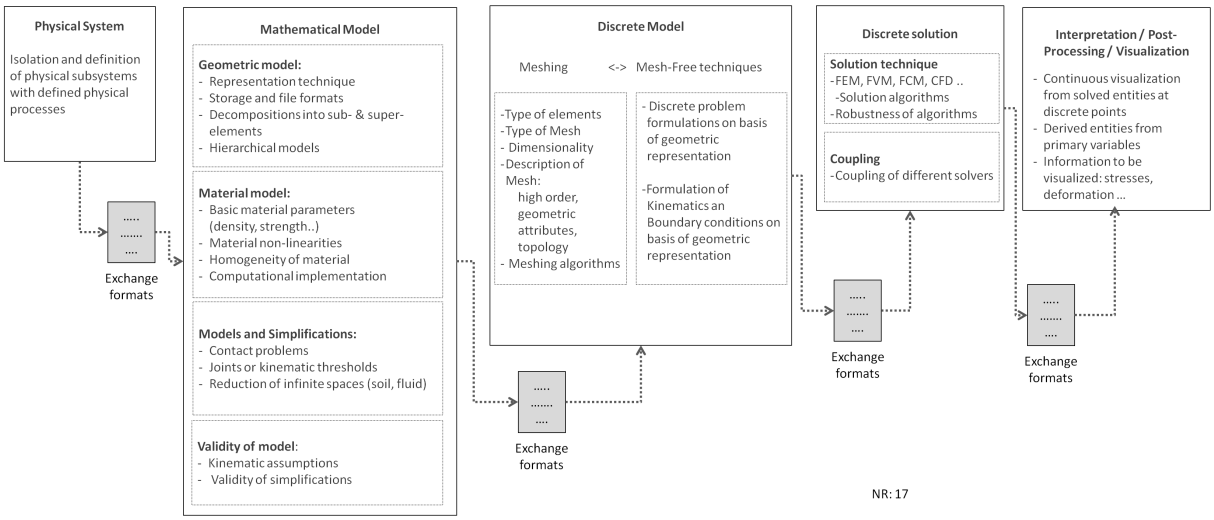


Figure 2.4: Sub-tasks in the processing and analysis workflow

Geometric modelling can be, depending on the complexity of geometry, a very time-consuming and cumbersome task. Errors or inconsistencies in the model might affect all downstream processes in terms of accuracy or stability. Manual data manipulation can be inevitable if no efficient and robust algorithms are present to detect defects and errors. Problems in the engineering workflow due to defects in geometric descriptions is the main aspect of this work. To understand the sources for problems it is important to have an understanding of the concepts for geometric modelling.

2.2 Geometric Modelling

There are many different approaches to describe the geometry of physical domains or spatial objetcs. Which concepts to choose depends on the intrinsic characteristics of the problem and objects to be modeled. Aspects like dimensionality and complexity of the objects influence the geometric modelling significantly. In the following sections some basic concepts for geometric modelling alongside with their advantages and disadvantages will be presented. There exists quite a huge body of literature on geometric modelling. A good overview is provided by [22], [43] and [29] for example.

2.2.1 Sets, relations and graphs

Before the reader will be introduced in several schemes for geometric representation this section will cover sets, graphs and relations since these concepts are used in the following sections. For a more detailed review on graph theory the reader is referred to [12]. Graphs are often used to store and describe geometric objects efficiently in a digital environment. A graph is an abstract representation including a set of elements and the links between pairs of these elements. The elements to be connected are usually called vertices, whereas the links connecting pairs of these vertices are referred to as edges. A graph can be represented in diagrammatic form by a set of dots for the vertices and lines indicating the edges. Picture 2.5 sketches a labeled graph on 6 vertices and 7 edges.

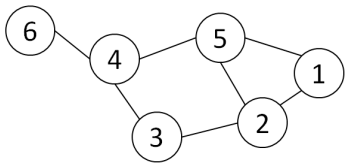


Figure 2.5: Simple graph enxample

Let us consider a set of elements $A = \{a, b, c, d, \dots\}$. The order of the elements within the set is irrelevant. Two elements within the set can form a pair, for example (a, b). If the order is relevant we speak of an ordered pair. For two given sets A and B, the set of all ordered pairs (a, b) which can be formed from the elements $a \in A$ and $b \in B$ is given by the Cartesian product of the sets A and B and designated by:

$$A \times B = \{(a, b) | \; a \in A \wedge b \in B\} \tag{2.1}$$

A binary relation ρ is a collection of ordered pairs of elements in the corresponding sets A and B, or in other words the binary relation is a subset of the Cartesian product of two sets A and B: $\rho \subseteq A \times B$. In case the sets A and B are equal: $A = B$ we speak of a homogeneous relation denoted by $\rho \subseteq A \times A$.
A relation differs slightly from the concept of a graph. Two relations ρ and ϱ can share the same graph but may be different due to their underlying sets and therefore form distinct relations.
A very important graph for geometric representations is the so-called directed graph, which is defined on the domain $G = (V; R)$. V describe the set of vertices and $R \subseteq V \times V$ is the set of edges of the graph. The edges are given in form of ordered pairs (x, y) with start vertex $x \in V$ and end vertex $y \in V$ for a specific edge $r \in R$. The ordering of the pairs (x, y) differentiates directed graphs from ordinary undirected graphs. For geometric representation schemes and their algorithms like for example search algorithms or validity checks, the concept of directed graphs is essential. At a later point of this thesis healing routines based on graphs will be described.

2.2.2 Spatial Structures

To describe the effects taking part in a physical system it is in general necessary to define the material parameters like density or strength of objects in the domain. The material properties

can influence the modelling aspects significantly, especially in the case of composites and composed materials since areas of different material have to be distinguished in the model as well. Beside the material also boundary conditions for the partial differential equations have to be defined. These boundary conditions are also linked to the geometry of the spatial objects since they are defined at geometric locations on the structure like points, lines or faces. The boundary conditions themselves have a mathematical model approximating the real-life physical effects for example supports or clamping. Load conditions like for example single forces or pressure fields are a second type of boundary conditions also linked to locations on the structure.
By taking into account the boundary conditions and material properties, we realize that the geometric model is influenced not by the pure geometric appearance but has rather to be adjusted to appearance, material and solution technique. This fact is also shown in picture 2.1 by the reverse arrows indicating a backward link from the solution process to the modelling process. This concept can be extended to the discretization and meshing which will itself also influence the modelling.
An entire model is often composed of individual bodies or complexes. For each a suitable concept of modelling and geometric description has to be chosen, also taking into account the possible interactions between the bodies. For most of the problems appearing in the fields of civil engineering, mechanical engineering and also marine engineering we can identify bodies which are bounded by faces, which are themselves bounded by sequences of edges. The faces have their specific geometric shapes and likewise the edges. The edges itself are bounded by points in space, so-called nodes.
This circumstance can be used to derive one possible suitable geometric representation scheme for spatial objects. Picture 2.6 shows a schematic UML code describing a spatial structure, including load and deformation boundary conditions and indicates the influence of these objects on the geometric description. The concept presented in picture 2.6 is closely related to the so-called boundary representation scheme (BREP). It is one possible geometrical description, but there exists a wide range of concepts which are described in detail in the following sections.

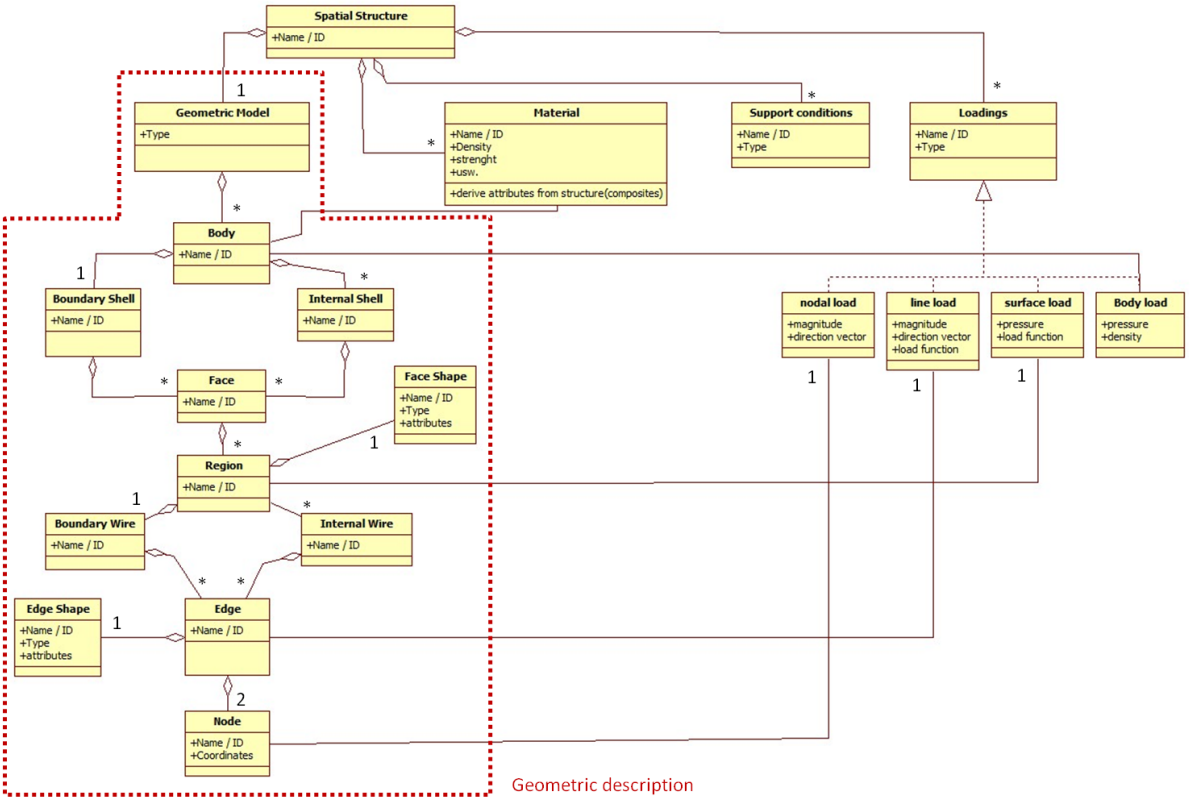


Figure 2.6: UML structure for description of spatial objects

2.2.3 Constructive methods / Direct Representation Schemes

The constructive or direct representation approach defines a body by a sequence of operations to construct the object out of a set of primitive objects. The applied geometric operations to define the final geometric object are limited and rather simple. A popular representative for this concept is the so-called CSG scheme (for Constructive Solid Geometry). The primitive objects in this model are cubes and cylinders in three dimensions and respectively rectangles and circles in two dimensions. By the use of Boolean operations like union, intersection and differences these primitives can be transformed to more general geometric objects. The primitive operations can be extended by operations like scaling, rotation or extrusion to make modelling more intuitive.

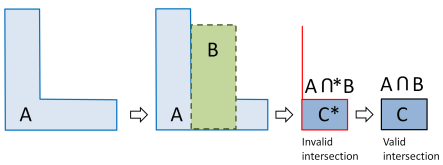


Figure 2.7: Invalid intersection operations in the CSG scheme

The primitive Boolean operations have to be modified slightly as described more in detail in [22] and [36] to guarantee the validity of the resulting models as shown in figure 2.7 which sketches a L-shaped object A and its intersection with a rectangle B. Both objects form valid 2D solids. Nevertheless the resulting object has a dangling edge. Therefore the intersection operations have to be modified such that regularized intersections result in topologically valid objects. The term topology will be explained more in detail when presenting the BREP scheme in section 2.2.6. A condition for validity is that the resulting objects resulting from a sequence of geometric operations should form a closed solid with no dangling edges. This condition is stated here without a proof or further explanation and left to the readers intuition. In a floating point environment

with possible round-off problems this validity condition might not be easy and trivial to be fulfilled. Modelling with this representation scheme is rather intuitive, therefore a range of programs like for example CAD-software make use of this concept. Picture 2.8 shows an example created in the CAD-software AutoCAD as a pure result of constructive operations.

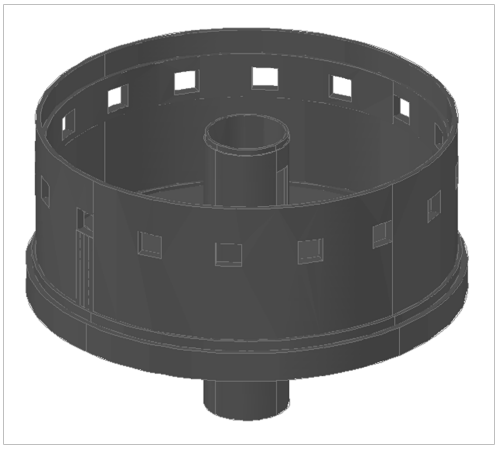


Figure 2.8: Complex CSG example constructed in AutoCAD

The sequence of applied operations and basic primitives can be stored in form of a tree as shown in figure 2.9. It is important to mention that this representation is not unique. For example the final object in figure 2.9 can be constructed in different ways. This fact is a disadvantage of the CSG scheme. It makes it for example difficult to judge if two objects are equal – the so-called same object detection problem. Concerning healing operations it is not trivial to detect if two double objects appear in the model. Even more complicated is the fact, that it is not trivial and easy to determine if a sequence of operations results in a null object. For example when a collision of objects is to be detected it is sufficient to check if the intersection of those objects is null. The intersection is rather easy to represent, but the nullity check is much more complicated with this scheme. Undesired intersections of objects are therefore more difficult to detect by healing routines, especially in a floating point environment. Complex shapes might result in large constructive trees, which is another disadvantage of this scheme. Furthermore local modifications of the geometry cannot so easily be performed in the SCG model.

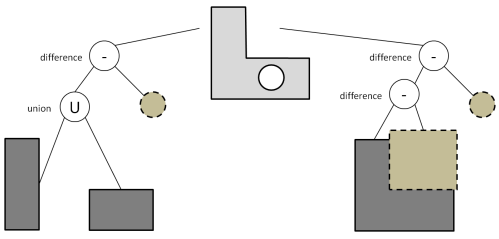


Figure 2.9: Non-unique CSG-tree for a geometric model

2.2.4 Standard Cell Enumeration Schemes

This representation concept is based on a subdivision of the entire domain by a grid in two dimensions respectively cubes in three dimensions. Therefore a body can simply be approximated by the grids which are inside and those outside the body. By refining the grid resolution all bodies can be described with a desired accuracy. The inside-outside information can be

translated into simple bit information for each grid cell. Storage of the geometry information is therefore efficient with this scheme. The position of the cell (also called pixels in 2D or voxels in 3D) is defined by the midpoint. Additionally grid size and the bit information for each cell has to be stored. A varying grid resolution adjusted to the local shape complexity increases the accuracy and at the same time optimizes storage efficiency. Picture 2.10 sketches a standard cell enumeration description of a 2D element with a varying grid resolution.

The advantages of this concept lie in the easy and efficient description and storage. Nevertheless there are several drawbacks. A very fine resolution is necessary to represent complex shapes with an acceptable accuracy. The scheme is not very flexible to react to local shape characteristics and complexities. Furthermore it might be non-trivial to define the intersections of a body with the cells to obtain the inside-outside information for complex shapes. Therefore the voxelization can be very expensive and overshoot the advantages of efficient storage and simple description. Nevertheless this concept is widely used in several applications and algorithms. Some hybrid approaches include this concept to store coarse geometrical information like the approximate position of elements in space. Like that efficiency of algorithms like intersections for example can be increased. Computer graphics is another important field of application.

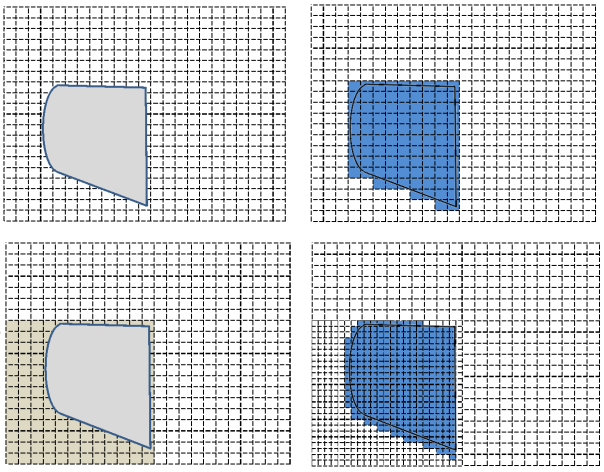


Figure 2.10: Simple standard enumeration scheme example

2.2.5 Space Trees

The basic idea behind the standard cell enumeration scheme is a space decomposition. This idea can be applied in a more general manner for geometrical representation. Therefore the domain is decomposed into subdivisions called cells or pixels respectively cubes or voxels in three dimensions as described before. To overcome the disadvantage of a high level of decomposition (equivalent to a small grid size) a hierarchical decomposition is performed. The idea is to adjust the level of decomposition to the shape complexity. This approach of hierarchical spatial decomposition evolved indeed out of simpler spatial enumeration schemes as described in [37]. The difference between varying grid sizes which can be applied in standard enumeration schemes as well and this concept is that the construction rules for refinement are clearly defined. By this many algorithms can be optimized. The subdivision of the space is done by a fixed number of sub-elements. The amount of sub-elements depends on the applied scheme and dimension of the problem. Widely used decomposition schemes are the following:

Name	List	Binary Tree	Quadtree	Octree
Subdivision	$1 = 2^0$	$2 = 2^1$	$4 = 2^2$	$8 = 2^3$

Octrees are simply the extension of Quadtrees into the third dimension. However both concepts were developed independently by various researchers mainly in the field of computer graphics as described in [37]. Picture 2.11 sketches the construction of a Quadtree applied on the same object as used for standard enumeration scheme example. Those cells where a higher accuracy is needed are subdivided into four children-elements. The advantage is apparent since subdivision can be performed only where necessary according to a clearly defined construction rule

Picture 2.11 also shows the representation of this scheme in form of a tree forming a directed graph. This tree structure can be efficiently transformed in a bit storage scheme and exploited in many computational algorithms of computational science like spatial search routines or neighbouring searches.

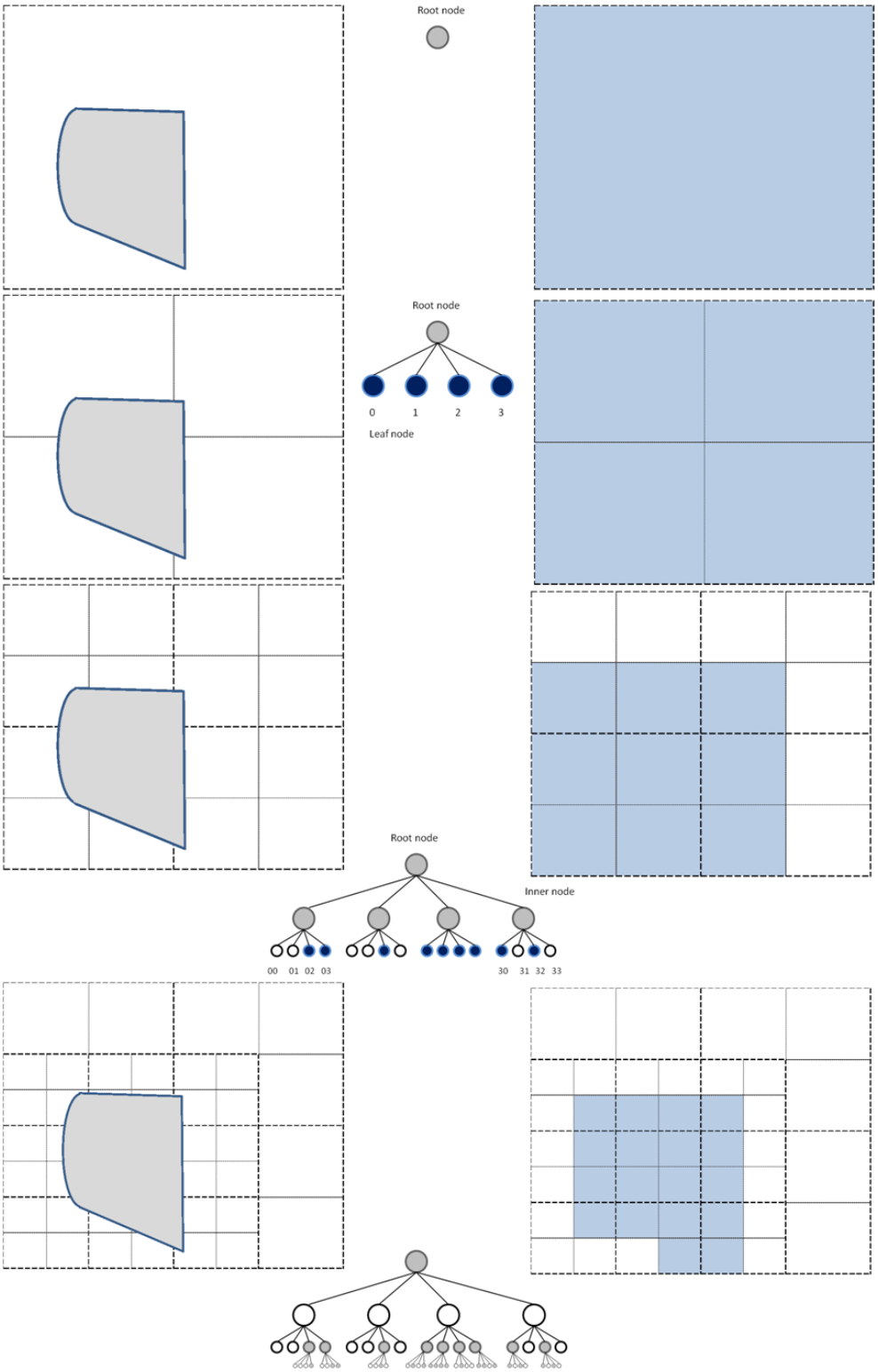


Figure 2.11: Simple Quadtree example

The objects in picture 2.12 are examples for Octrees applied in graphical object rendering. Space trees like Octrees play an important role in computer graphics.

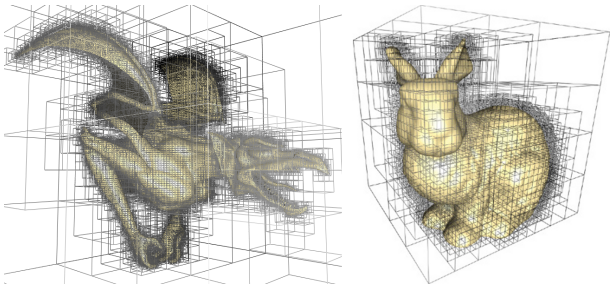


Figure 2.12: Quadtree example in computer graphics [28]

2.2.6 Boundary Representation Model - BREP

The boundary representation (BREP) scheme is a very flexible, powerful and accurate concept to describe geometric information efficiently and is widely used in many modern modelling tools. A compact overview of CAD and CAE applications and the BREP modelling scheme can be found in [48] and [46]. Its underlying concept makes use of the fact that the boundaries of a solid (valid closed solid) separate the space into regions inside the solid and regions complementary exterior. Therefore the solid can be represented by its boundary faces and every point can be unambiguously tested against the solid by testing it against the boundary of the solid. This concept is an indirect method to model a spatial object. To apply this scheme certain conditions have to be fulfilled. The objects have to be defined by a set of closed volumes. That implies that the surfaces to describe one body form one specific closed volume complex. Only by that a certain point can clearly be identified to be inside or outside this body. We will come back to this important condition of validity for BREP models, since violation can be a source of errors and inconsistencies in the geometric description possibly causing problems in downstream applications. The ability to define or compute orientations of the sub-areas is another condition necessary for this representation technique. Orientation information is used to test a point in space against the surfaces of the BREP objects.

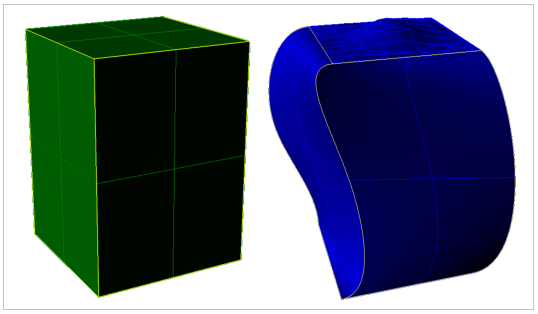


Figure 2.13: BREP model with same topology but different geometric shapes

After introducing the concept of BREP schemes more in detail the reader will be provided with further information on the validity conditions since a profound understanding is important for several algorithms and routines which will be discussed in this thesis. The concept of BREP models makes use of graphs and relations, the so called vef-relations. These relations plus a limited number of primitives are sufficient to indirectly describe the spatial objects by means of their boundaries. The primitives which are used are nodes, edges and faces. They hold all the geometric shape information of the objects, the so-called geometry of the object. The information how the body is constructed out of these primitives is given by its topology stored in vef-sets and the vef-relations:

V = *vertices* $\{v_1, v_2, \ldots, v_{n_v}\}$
E = *edges* $\{e_1, e_2, \ldots, e_{n_e}\}$ with $e_i = (v_a, v_b)$ ordered pair of vertices
F = *faces* $\{f_1, f_2, \ldots, f_{n_f}\}$ with $f_i = (e_{i1}, \ldots, e_{im})$ ordered pair of edges

The BREP scheme has the ability to lower the dimensionality for geometric modelling of an object. To model a 3D solid we must only model 2D faces. The faces itself are defined by their 1D edges. Since the edges are bounded by the vertices (or nodes) we eventually end up with 0D point primitives, given by the coordinates of points with respect to the used coordinate system. In case all primitives have a linear shape, the dimensionality is reduced significantly. No further geometric information is needed except the node coordinates. All higher level entities like edges and faces can be constructed due to the topological connectivity (vef-relations).
Modern modelling tools expand this simple approach by allowing more complex (e.g. curved) shapes for primitives. The geometric shape information is included in the model by assigning geometrical shape attributes to the primitives. The nodes or vertices are not affected, but edges and faces can be provided with additional shape information. Topologically an edge is simply bounded by nodes. Between the nodes the geometric shape of the edge (e.g. curvature) is given by geometric attributes. A surface or region is bounded by a sequence of edges. Likewise for the edges the shape of the face between the boundaries (external and internal boundaries) is given by the geometric attributes. This expansion to more complex primitives increases the flexibility of the BREP scheme significantly to an almost generally applicable representation scheme. One of its big advantages is the clear distinction between topology and geometry. Modifications on the shape can be performed easily without changing the connectivity information stored in the topology. Two objects can share the same topology but nevertheless form two clearly different geometric objects due to their shape attributes assigned to the primitives. Picture 2.13 shows an example for two objects with the same topology but different shapes.
Topology describes the connectivity of the entities, stored as vef-realtions which are given in detail in table 2.1. Picture 2.14 sketches the relations graphically.

$vv \subseteq V \times V$	Points are adjacent - have a common edge
$ve \subseteq V \times E$	Point bounds edge
$vf \subseteq V \times F$	Point is on boundary of a face
$ev \subseteq E \times V$	Edge has point as corner
$ee \subseteq E \times E$	Edges are adjacent - have common vertex
$ee' \subseteq ee$	Edges are adjacent, plus they border the same face
$ef \subseteq E \times F$	Edge bounds face
$fv \subseteq F \times V$	Face has point as border
$fe \subseteq F \times E$	Face has edge as border
$ff \subseteq F \times F$	Faces are adjacent - have a common edge

Table 2.1: vef-relations in the BREP modelling scheme

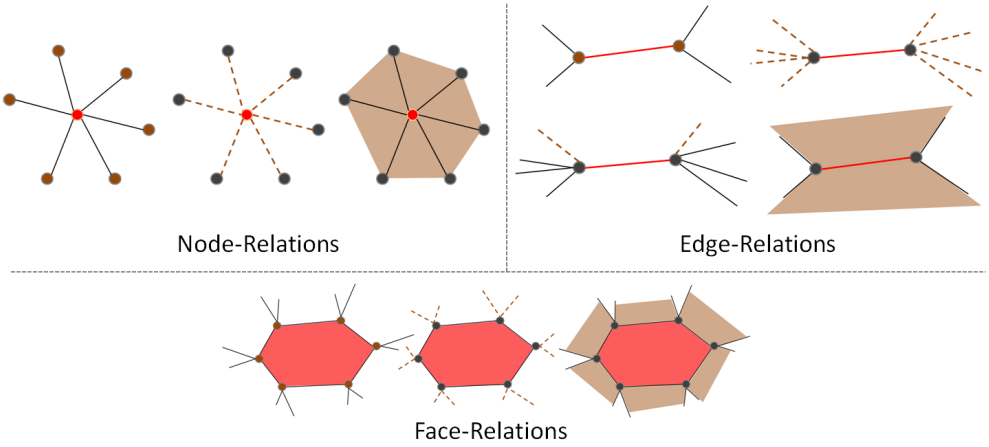


Figure 2.14: Vef-relations in the BREP representation scheme

In comparison to space strees storage is not so efficient within the BREP scheme. Topological and geometrical information has to be stored. The geometry can be stored in means of geometric attributes. Furthermore the vef-relations have to be stored to derive the topology. There exist two main models to store the topology, a minimal model and the so-called winged-edge data structure. For the minimal model it is enough to store the ev- and ef-relations and from that derive all other topological information. This model is efficient in storage but inefficient for many algorithms. A commonly used storage scheme is the so-called winged-edge representation proposed by Baumgart ([11]). The winged-edge structure stores additionally the following information:

- ee' relation: adjacent edges bordering a common face
- v-list: one edge per vertex bordered by this vertex
- f-list: one edge per face bordering that face

These additional relations speed up many algorithms significantly. Efficiency is related to storage as well as performance of geometric algorithms. This fact will also be exploited when discussing the practical implementation of healing routines. It can be advantageous to expand the stored relations, such that for often applied algorithms the necessary connectivity information is stored in the database.

Further conditions of validity

At the beginning of this section several conditions of validity were already mentioned. At this point the reader will be provided with more information on important validity requirements since a violation of them is often the source of errors or inconsistencies in the model. The BREP scheme originates from the topological concept of manifolds which induces the validity conditions described in the following. A detailed analysis of topological concepts and validity conditions of the BREP model can be found in [39]. In short: "B-rep directly employs the Jordan's curve theorem on boundary determinism stating that a closed, connected, orientable and nonself-intersecting surface determines the interior of a solid." ([39]).
Objects in BREP models have to be described by boundaries which form manifold cell complexes. Therefore all distinct primitives have to obey disjointedness. Picture 2.15 shows an invalid BREP model due to the fact that two distinct nodes (number 4 and 5) and two edges (number 11 and 12) are not disjoint. These entities are referred to as double entities and violate validity conditions.

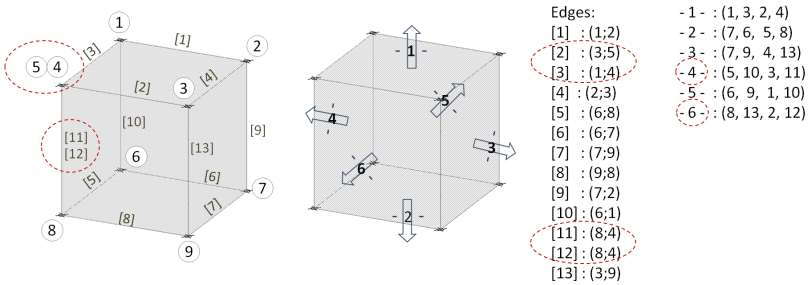


Figure 2.15: Invalid BREP model: double entities

The conditions to ensure the absence of non-manifold elements are easy to state, but can be much more difficult to obey in practical applications. Problems arise due to round-off errors and tolerance mismatches, especially when geometries are translated to different file formats or migrated to different systems. The problem to exchange geometric information is covered at a later point in detail. At this point it is already mentioned, that tolerances to define disjointedness of points or other primitives can influence geometric modelling significantly. The absence of non-manifold points is ensured by the following conditions:

- All pairs of vertices are disjoint.
- Pairs of edges are either disjoint or intersect at one vertex.
- Pairs of faces are disjoint or intersect at a common edge.

Routines to check if these conditions are fulfilled are non-trivial, especially for complex geometric shape attributes. Picture 2.16 visualizes the meaning of the stated conditions above. Picture 2.16 shows also the problem of possible intersections (or even self-intersections) related to complex shape attributes.

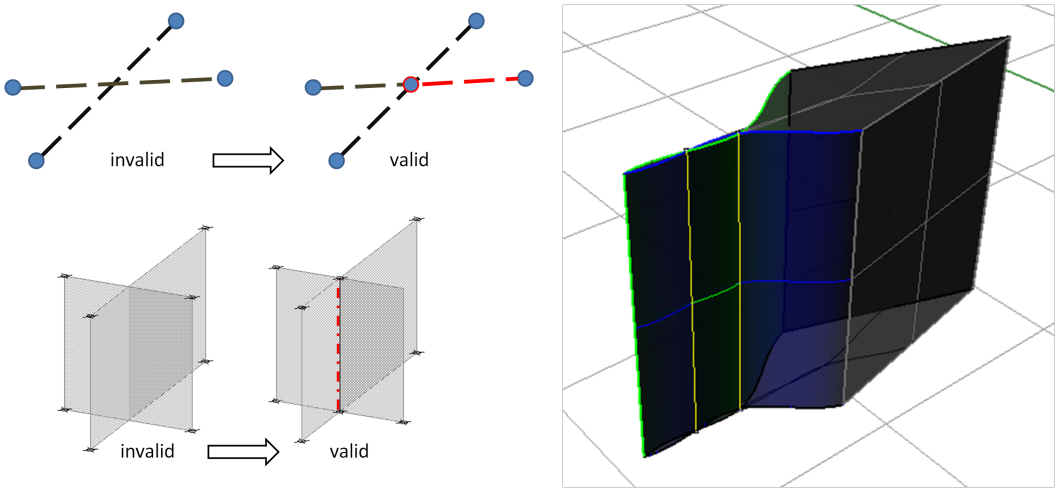


Figure 2.16: Invalid BREP model: non-manifold points and surface intersections

2.2.7 Description of geometric attributes and shapes of curves and faces

After the overview on different geometric representation schemes in the previous sections, this section will review some basics on representation of geometric shapes itself. The difference between a representation scheme of a spatial object in total and the geometric shapes becomes

evident for BREP models. Here geometry and topology is distinguished. Topology holds information about the connectivity of the primitive entities. Geometry refers to geometric shape information like curvature of lines or surfaces, assigned to the primitives. Content of this section is not anymore the concepts of different representation schemes but rather the shape attributes assigned to single entities in those representation schemes. For the BREP model this distinction is easy to understand and easy to apply. In other schemes this might not be the case. In the standard cell enumeration scheme or space tree concept for example, geometric shape attributes for lines or surfaces are needed to derive the information about the cells in the decomposed space which have intersections with the spatial structure to be described. In a direct scheme like the CSG-models this clear distinction is much harder to preserve. The CSG concept is based on primitives and geometric operations out of which a spatial object can be constructed. To perform the geometric operations like intersections or unions geometric shapes are necessary.
In the previous sections it was already mentioned that almost all of the concepts listed above have their specific conditions for validity. It was described that these conditions are referred to topological and geometrical properties. One example for links between validity conditions and geometric shape information can be found in the CSG-model, where it was stated that the geometric operations must end up in enclosed volumes with no dangling edges or faces. For complex shapes these postulations can be quite difficult to be analyzed. Therefore robust algorithms are needed which will make intensive use of the geometric shape information of surfaces or lines and exploit their specific characteristics. A similar example was given for BREP models where the non-manifold condition might be violated by intersections due to the geometry of lines or faces.
These observations motivate the following short introduction into geometric shape representations for lines and surfaces. For detailed information on these topics the reader is referred to [34].

Implicit and parametric forms of representation

The most common applied methods for representing curves and surfaces are implicit and parametric schemes. The implicit representation is given by formulas of the type: $f(x, y, z) = 0$ which describe the implicit relationship of x-,y- and z-coordinates of points on the line or surface. In contrast to the implicit form, the parametric description provides an equation for each coordinate with reference to an independent parameter u. The representation formulas are given as vector-valued functions. To illustrate the difference the equations for a circle in 2D-space with center at the origin is given in both forms.

Implicit:

$$f(x, y) = x^2 + y^2 - 1 = 0 \tag{2.2}$$

Parametric: for the first quadrant, with $0 \leq u \leq \frac{\pi}{2}$

$$\begin{aligned} x(u) &= cos(u) \ , \\ y(u) &= sin(u) \end{aligned} \tag{2.3}$$

The range of the parameter u is in general arbitrary but usually normalized to $u \in [0, 1]$. The parametric description is not unique. For example for the circle given above also different

formulas can be derived by setting $t = tan(\frac{u}{2})$. Then the formulas evolve into:

$$\begin{aligned} x(t) &= \frac{1-t^2}{1+t^2} \\ y(t) &= \frac{2t}{1+t^2} \qquad , \ 0 \leq t \leq 1 \end{aligned} \tag{2.4}$$

The parametric formulas can also be understood as formulas for paths traced out by a particle as a function of time. The parameter u has the function of a pseudo time, whereas the range of the parameter describes the interval over which the particles are traced. The first and second derivatives $C'(u) = (x'(u), y'(u))$ respectively $C''(u) = (x''(u), y''(u))$ of the parametric formulas provide information like velocity and acceleration. For the circle example given above one can show that the magnitude of the velocity vector $|C'(u)|$ is constant. This means that the directions of the particles are a function of time but the velocity is not. This refers to a uniform parameterization.
The parametric equations for a surface are given analogously with respect to two independent parameters u and v. On a surface at every point a unit normal vector N can be defined using the partial derivatives of $S(u, v)$, denoted by $S_u(u, v)$ and $S_v(u, v)$. The formula reads:

$$N = \frac{(S_u \times S_v)}{|(S_u \times S_v)|} \tag{2.5}$$

Independent from the parameterization a normal vector exists at every point of the surface. Nevertheless in the formulas the denominator can vanish and therefore it might be possible that for a certain parameterization at specific points the normal vector cannot be computed.
Both the implicit and parametric forms have their advantages and shortcommings, so no general statement can be given about which method to use. The following list gives an overview over differences, weaknesses and strengths of the two methods.

- The parametric functions can be extended to 3 dimensions very easily and intuitively.
- Bounded curve segments or surface patches are not trivial to be represented in a implicit way. Boundedness can rather easily be included in parametric formulas by bounding the interval of the independent parameters.
  On the other hand unbounded geometry is difficult to implement using parametric descriptions.
- Parametric curves possess a direction of traversal from start to end point in contrast to implicit descriptions. Therefore routines demanding traversal information like generating ordered sequences are easier to apply using parametric formulations.
- Shape design, shape modelling, shape modification and graphical representation of parametric shapes are in general more intuitive to users and computational algorithms.
- The complexity of geometric operations refers strongly to the form of representation. For example computing points on a curve or surface is more difficult for implicit functions. Whereas to check if a given point is on a curve or surface described in parametric manner is more difficult.
- Parametric descriptions might suffer from anomalies related to the parameterization with no geometric background. The inability to compute a normal vector for certain parameters is one example for this type of drawback.

Parametric descriptions for geometric modelling

Complex shapes for curves and surfaces can efficiently and very intuitively be modeled and described using free-form descriptions like NURBS or Splines. In many engineering applications and also other fields like computer graphics these methods are widely used and almost state of the art due to their flexibility to all kind of different shapes. For this reason the following sections will have a closer look on those techniques which all belong to the type of parametric geometric representations.
Many examples given within this thesis to evaluate the performance of the code are based on free-forms, and on NURBS in particular. Therefore a profound understanding of the underlying concepts is necessary to understand their strengths but also problems arising due to this representation with reference to healing routines.
There is theoretically an infinite number of functions able to form a basis for parametric descriptions. Due to modelling and implementation aspects the suitable functions are practically limited by fulfilling required or desired conditions and properties:

- Ability to precisely represent all shapes required by the systems
- Easy, efficient and accurate processing in a computer:
  - efficient computation of points and derivatives on the curve (surface)
  - numerical processing of curves is insensitive to round-off and floating point errors
  - efficient storage of functions in memory
- Functions are intuitively to use and mathematically well understood:
  - mathematically “stable” behaviour
  - local influence of control points or coefficents on curve, resulting in a more intuitive modelling an shape design
  - minimization of needed control points, decoupling of curves degree and number of necessary control points

The following sections will give an overview over the most commnly used basis functions.

Polynomials - Power Basis Curves Polynomial basis functions can be used for description and representation of curves. Nevertheless there are several important shapes which cannot be represented precisely and only approximated by polynomials. There are two important methods for polynomial basis functions - power basis and Bézier. They are mathematically equivalent but Bézier curves are better suited for computational geometric modelling.
Power basis curves of degree n are given by:

$$C(u) \;=\; (x(u), y(u), z(u)) \;=\; \sum_{i=0}^{n} \boldsymbol{a_i} * u^i \tag{2.6}$$

Where $\boldsymbol{a_i}$ is a vector $a_i(x_i, y_i, z_i)$. Hence:

$$x(u) \;=\; \sum_{i=0}^{n} x_i * u^i \quad , \; y(u) \;=\; \sum_{i=0}^{n} y_i * u^i \quad , \; z(u) \;=\; \sum_{i=0}^{n} z_i * u^i \tag{2.7}$$

Which can be reformulated in matrix form:

$$C(u) \;=\; [\boldsymbol{a_0}, \boldsymbol{a_1}, \ldots, \boldsymbol{a_n}] \times \begin{bmatrix} 1 \\ u \\ . \\ . \\ u^n \end{bmatrix} \;=\; [\boldsymbol{a_i}]^T [u^i] \tag{2.8}$$

The $n+1$ functions $\{u^i\}$ are called basis functions; $\{\boldsymbol{a_i}\}$ are referred to as coefficients and can be derived from:

$$\boldsymbol{a_i} \;=\; \frac{C^{(i)}(u)|_{u=0}}{i!} \tag{2.9}$$

With the i-th derivative of $C(u)$ at $u = 0$. For a given u_0 the point $C(u_0)$ on the curve is efficiently computed by the Horner's algorithm, described in [34].
There are several shortcomings and drawbacks of power basis functions:

- Unnatural for interactive shape design since coefficients $\{a_i\}$ hold only little geometrical insight of the underlying shape;
- End conditions for both ends of the curve are not easy to apply;
- Numerical problems might appear, Hoerner's algorithm is prone to round-off errors;

Polynomials - Bézier Curves Bézier curves address the shortcomings of polynomial power basis curves. Both are mathematically equivalent and can be transformed into each other. A n-th-degree Bézier curve is given by:

$$C(u) \;=\; \sum_{i=0}^{n} B_{i,n}(u)\boldsymbol{P_i} \qquad , \; 0 \leq u \leq 1 \tag{2.10}$$

Where $\{\boldsymbol{P_i}\}$ are the coefficients, also called control points of the shape. The basis functions $\{B_{i,n}(u)\}$ are given by the *nth*-degree Bernstein polynomials:

$$B_{i,n}(u) \;=\; \frac{n!}{i!(n-i)!} u^i (1-u)^{n-i} \tag{2.11}$$

There are several important geometric properties of Bézier curves:

- *Endpoint interpolation property*: $C(0) = P_0$ and $C(1) = P_n$; the curve meets the control points at the bounday of the parametric interval.
- The polygon formed by the control points $\boldsymbol{P_i}$ is called the control polygon. It appproximates the curve. The curve is contained in the convex hull formed by the control points - *convex hull property.*
- The curve at $u = 0$ and $u = 1$ is tangent to the connection of the control points $\boldsymbol{P_1} - \boldsymbol{P_0}$ respectively $\boldsymbol{P_n} - \boldsymbol{P_{n-1}}$.
- Initially at $u = 0$ the curve is turning in the same direction as given by the polygon $\boldsymbol{P_0 P_1 P_2}$.

- *Variation Diminishing Property*: no straight line intersects the curve more often than it intersects the curve's control polygon (in 3D replace line by plane).
- Curve is invariant under the usual transformations (rotations, scalings, translations). Therefore the transformations can be applied to the control points instead of the curve itself.

The following properties hold for the Bernstein polynomials:

- Nonnegativity: $B_{i,n}(u) \geq 0$ for all i, n and $0 \leq u \leq 1$;
- Partition of unity: $\sum_{i=0}^{n} B_{i,n}(u) = 1$ for all $0 \leq u \leq 1$;
- $B_{0,n}(0) = B_{n,n}(1) = 1$;
- $B_{i,n}(u)$ attains exactly one maximum at $u = i/n$ over the interval [0,1];
- For any n, the set of polynomials $\{B_{i,n}(u)\}$ is symmetric with respect to $u = 1/2$;
- Recursive definition: $B_{i,n}(u) = (1-u) * B_{i,n-1}(u) + u * B_{i-1,n-1}(u)$

The recursive definition property can be transformed into an efficient computational algorithm for the non-zero Bernstein polynomials at a fixed u to obtain a point on the curve for a given u-value (see [34]). The algorithm is called deCasteljau algorithm. This algorithm is numerically more stable than the Horner's algorithm for power basis functions. Another advantage of the Bézier representations is the more intuitive use of the control points to modify the curve's shape, in comparison to the coefficients in a power basis representation. Nevertheless there are also some drawbacks of this curve description:

- Some simple shapes like circles cannnot be represented exactly (only approximated).
- In general offset curves cannot be expressed by a Bézier representation.
- Modifications of the curve are non-local and have a global effect on the shape.

Polynomials - Rational Bézier Curves Rational Bézier curves are a generalization of Bézier curves addressing the drawback of non-rational Béziers not being able to express simple shapes like circles, ellipses, hyperbolas, etc. exactly. They are given by the following formula:

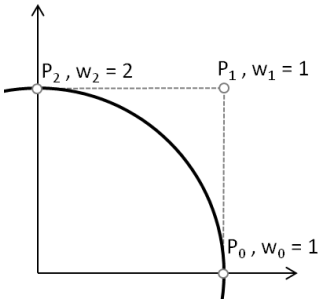


Figure 2.17: Rational Bézier description of a circle quadrant

$$C(u) = \frac{\sum_{i=0}^{n} B_{i,n}(u) w_i \boldsymbol{P_i}}{\sum_{i=0}^{n} B_{i,n}(u) w_i} \qquad 0 \leq u \leq 1 \tag{2.12}$$

with the scalar values w_i, called weights. By assuming $w_i \geq 0$ for all i we assure the denominator to be greater than zero for all $u \in [0, 1]$ and therefore the formula can be reformulated:

$$C(u) = \sum_{i=0}^{n} R_{i,n}(u) \boldsymbol{P_i} \qquad 0 \leq u \leq 1 \tag{2.13}$$

with the rational basis functions:

$$R_{i,n}(u) = \frac{B_{i,n}(u) w_i}{\sum_{j=0}^{n} B_{j,n}(u) w_j} \tag{2.14}$$

Since non-rational Bézier curves are a special case of rational Bézier curves in case all weights $w_i = 1$, the geometric properties of both types are very similar. In comparison to purely non-rational Bézier curves it is possible for a rational description to represent circles and other shapes exactly. For example a exact formula for the first quadrant of the unit circle with radius $R = 1$ and control points $P_i = \{(1,0);(1,1);(0,1)\}$ is obtained for weights $w_i = \{w_0 = 1; w_1 = 1; w_2 = 2\}$. Picture 2.17 sketches the resulting shape.
At this point the reader will be introduced in a very convenient property of rational functions with one common denominator concerning data storage and computational processing which also holds for rational B-Splines introduced at a later point of this section. The basic idea is to use homogeneous coordinates to represent a curve given in n-dimensional space in $(n+1)$-dimensional space. A point $\boldsymbol{P} = \boldsymbol{(x, y, z)}$ in 3D is transformed to 4D space if $w \neq 0$ by $\boldsymbol{P^w} = (wx, wy, wz, w) = (X, Y, Z, W)$. Reversely the original point $\boldsymbol{P}$ is obtained from $\boldsymbol{P^w}$ by the perspective mapping H with its center at the origin, given by:

$$\boldsymbol{P} = H\boldsymbol{P^w} = H\{(X,Y,Z,W)\} = \begin{cases} (\frac{X}{W}, \frac{Y}{W}, \frac{Z}{W}) & \text{if } W \neq 0 \\ direction(X,Y,Z) & \text{if } W = 0 \end{cases} \tag{2.15}$$

We can make use of this concept by constructing the weighted control points $\boldsymbol{P}_i^w$ for given weights w_i and contol points $\boldsymbol{P}_i$ and formulate a *non-rational* Bézier curve in 4D-space:

$$C^w(u) = \sum_{i=0}^{n} B_{i,n}(u)\boldsymbol{P_i^w} \tag{2.16}$$

Many algorithms are easier to perform for non-rational functions. After processing relocating the curve in 3D-space is done via:

$$x(u) = \frac{X(u)}{W(u)} = \frac{\sum_{i=0}^{n} B_{i,n}(u) w_i x_i}{\sum_{i=0}^{n} B_{i,n}(u) w_i} \tag{2.17}$$

..respectively for the other coordinates. Picture 2.18 sketches a curve in 3D-Space and its non-rational projection in 4D-space.

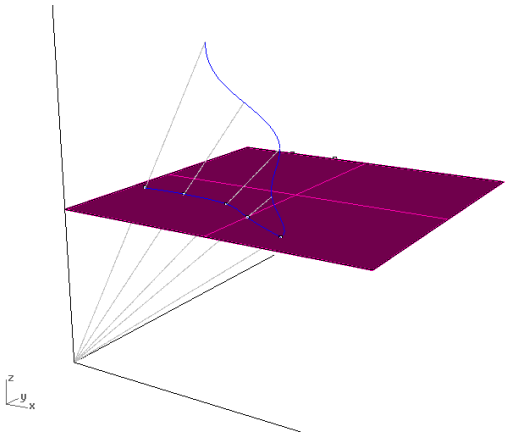


Figure 2.18: Non-rational curve projection in 4D space

Piecewise Polynomials - B-Spline Curves Motivated by the desired properties of a curve description for modelling purposes piecewise representations are introduced to overcome the shortcomings of curves consisting of just one polynomial or rational segment, in particular:

- A high degree is required to satisfy large number of constraints (e.g. $(n-1)$ degree needed for Bézier to pass trough n data points). High-order degree curves tend to be numerically unstable.
- A high degree is needed to fit complex shapes.
- Shape design via modification of one single control point has a global influence. A local influence would be preferable for intuitive shape design.

A remedy for these drawbacks are piecewise defined functions consisting of several segments meeting at the *breakpoints* of the curve. Picture 2.19 sketches a curve $C(u)$ defined over the domain $u \in [0,1]$ with four segments $\{C_1(u), C_2(u), C_3(u), C_4(u)\}$. The parameters at the breakpoints $u_0 = 0 < u_1 < u_2 < u_3 < u_4 = 1$ map into the end of the piecewise defined segments. At these points the segments meet with certain continuity.

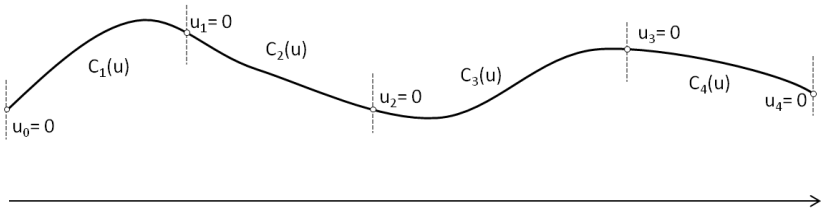


Figure 2.19: Piecewise parametric curve description

It is desired to formulate the parametric formula in the same style as for mono-segment curves (Béziers) by:

$$C(u) \;=\; \sum_{i=0}^{n} f_i(u)\boldsymbol{P}_i \tag{2.18}$$

with control points $\boldsymbol{P}_i$ and *piecewise polynomial basis functions* $f_i(u)$ with the desired degree and continuity for a fixed breakpoint sequence given by a vector $U = \{u_i,\ 0 \le i \le m\}$. The continuity is defined by the basis functions, therefore the positions of control points can be modified without changing the curves continuity. Furthermore the curves should have "nice" geometric properties like convex hull, transformation invariance or variation diminishing properties.
The motivation for piecewise functions was the local support property, yielding that the basis functions are non-zero only over a defined subinterval $[u_a, u_b]$. A representation fulfilling these properties are B-Splines, given by:

$$C(u) \;=\; \sum_{i=0}^{n} N_{i,p}(u)\boldsymbol{P}_i \tag{2.19}$$

with the basis functions $N_{i,p}(u)$ of degree p computed recursively by the Cox-deBoor algorithm (other definitions are possible but less suitable for computational algorithms) for a given knot vector $U = \{u_0, \ldots, u_m\}$. The knot vector is an ordered non-decreasing sequence of real numbers such that $u_i \le u_{i+1}\ ,\ i = 0, \ldots, m-1$. The basis functions are given by:

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

$$N_{i,p}(u) = \frac{u-u_i}{u_{i+p}-u_i} N_{i,p-1}(u) + \frac{u_{i+p+1}-u}{u_{i+p+1}-u_{i+1}} N_{i+1,p-1}(u)$$

These basis functions have the following important properties:

- $N_{i,0}(u)$ is a step function, non-zero only in the interval $u \in [u_i, u_{i+1})$
- For $p > 0$, $N_{i,p}(u)$ is a linear combination of two $(p-1)$-degree basis functions.
- The quotient $0/0$ might appear and is defined to be zero.
- To compute the basis function a knot vector U and the degree p have to be defined.
- The interval $[u_i, u_{i+1})$ is called knot-span and might be of zero lenght
- The functions $N_{i,p}(u)$ are defined on the entire real line, but zero outside the interval $[u_i, u_{i+p+1})$ (Local support). The recursive definition of the basis functions allows it to obtain the knot span over which a given basis function $N_{i,p}(u)$ is non-zero as sketched in picture 2.20 (left side).
- The recursive definition yields reversely that for a given knot span $[u_j, u_{j+1})$ only the basis functions $\{N_{j-p,p}, \ \ldots, N_{j,p}\}$ are nonzero, depicted in picture 2.20 (right side).
- Non-negativity: $N_{i,p}(u) \geq 0$ for all i,p and u.
- Partition of unity: Over $[u_i, u_{i+1})$ it holds $\sum_{j=i-p}^{i} N_{j,p}(u) = 1$
- All derivatives of $N_{i,p}(u)$ exist in the interior of the knot spans. At a knot the function $N_{i,p}(u)$ is $(p-k)$-times continuously differentiable, where k is the multiplicity of the knot. So the continuity can be increased by increasing the degree and decreased by increasing the knot multiplicity.
- For $p \geq 0$ all functions $N_{i,p}(u)$ attain exactly one maximum value for $u \in \mathbb{R}$.

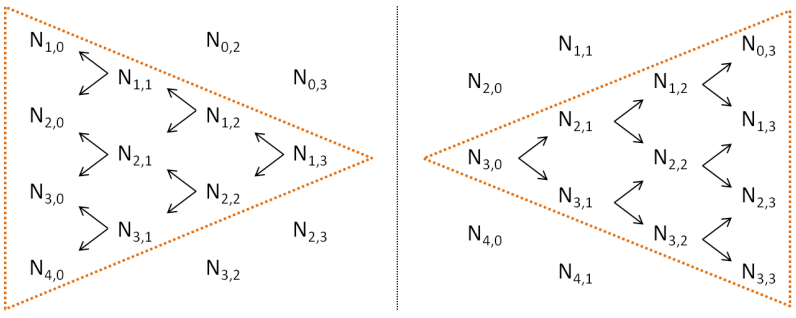


Figure 2.20: B-Spline basis function triangle

B-Spline curves have the following important geometric properties:

- The number of control points is equal to the number of basis functions, since every basis function is related to exactly one control point.

- The control points are independent of the knots. B-Splines are more flexible concerning shape fitting since shape and continuity are defined by the three parameters - the knot vector, the degree and the control points - in comparison to only two parameters for Béziers (control points and degree).

- The Cox-deBoor algorithm is very similar to the deCasteljau's algorithm for Béziers and is a corner cutting construction algorithm with geometrical insight.

- For a strictly increasing set of breakpoints $\{u_j\}$, $0 \leq j \leq k$, the set of all piecewise polynomial functions of dregree p on $\{u_j\}$ which are C^{r_j} continuous at $u = u_j$ forms a vector space, ν $(-1 \leq r_j \leq p)$. It holds $dim(\nu) = k(p+1)$ if $r_j = -1$ for all j, meaning there are no continuity constraints. Every constraint lowers the dimension by one and it holds:

$$dim(\nu) = k(p+1) - \sum_{j=0}^{k}(r_j + 1) \tag{2.21}$$

- When the degree p of the curve is fixed, the knot vector fully determines the basis functions $N_{i,p}(u)$.

- There is a special type of knot vectors, referred to as nonperiodic (or clamped / open) knot vector, given by:

$$U = \{\underbrace{a, \ldots, a}_{p+1}, u_{p+1}, \ldots, u_{m-p-1}, \underbrace{b, \ldots, b}_{p+1}\} \tag{2.22}$$

- For non-perodic knot vectors there are further geometric properties:

  · Bézier curves are a special type of B-Splines with a special knot vector given below:

$$U = \{\underbrace{0, \ldots, 0}_{p+1}, \underbrace{1, \ldots, 1}_{p+1}\} \tag{2.23}$$

  · Nonperiodic knot vectors lead to end point interpolation

  · furthermore we have $(m+1)$ number of knots and $(n+1)$ basis functions with $n = m - p - 1$

Piecewise Polynomials - NonUniform Rational B-Spline Curves Analogously to Béziers, rational basis functions can also be constructed for piecewise polynomials. We then speak of NonUniform Rational B-Spline Curves (NURBS). The non-uniformity refers to non-uniformly (non-equidistant) distributed knots. The basis functions evolve into rational functions due to assigned weights w_i. The formula for the parametric curve over the parameter domain $[a, b]$ is given by:

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i \mathbf{P}_i}{\sum_{i=0}^{n} N_{i,p}(u) w_i} \qquad a \leq u \leq b \tag{2.24}$$

with the control points $\{\boldsymbol{P}_i\}$, the weights w_i and the p-th-degree basis functions $N_{i,p}(u)$ defined over the knot vector U (in general non-uniform and usually nonperiodic). Rewritten with the rational basis functions:

$$R_{i,p}(u) \;=\; \frac{\sum_{i=0}^{n} N_{i,p}(u) w_i}{\sum_{j=0}^{n} N_{j,p}(u) w_j} \tag{2.25}$$

the formula reads:

$$C(u) \;=\; R_{i,p}(u)\mathbf{P}_i \tag{2.26}$$

The rational basis functions have the following properties:

- Non-negativity $R_{i,p}(u) \geq 0$ for all i,p and $u \in [0,1]$
- Partition of unity: $\sum_{i=0}^{n} R_{i,p}(u) = 1$ for all $u \in [0,1]$
- $R_{0,p}(0) = R_{n,p}(1) = 1$
- For all $p > 0$ the basis functions attain exactly one maximum over the domain
- Local support: $R_{i,p}(u) = 0$ for $u \notin [u_i, u_{i+p+1})$. Furthermore for a given $u \in [u_i, u_{i+1})$ only the functions $\{R_{i-p,p}(u), \ldots, R_{i,p}(u)\}$ might be nonzero.
- All derivatives exist in the interior of the breakpoints. At a knot $R_{i,p}(u)$ is $(p-k)$-times continuously differentiable, with knot multiplicity k.
- For weigths $w_i = 1$, the functions transform into B-Splines.

From these properties of the basis functions the following important geometric properties for NURBS-curves can be derived:

- End-point interpolation for nonperiodic knot vectors: $C(0) = P_0$ and $C(1) = P_n$
- Affine invariance: an affine transformation is applied to the curve by applying it to the control points. NURBS are also invariant under perspective projections.
- Strong convex hull property: The curve segment $C(u)$ for $u \in [u_i, u_{i+1})$ lies within the convex hull spanned by the control points $\{P_{i-p}, \ldots, P_i\}$.
- $C(u)$ is infinitely differentiable in the interior of knot spans and $(p-k)$-times differentiable at a knot of multiplicity k.
- Variation diminishing property: no plane has more intersections with the curve than with the control polygon.
- Local approximation: Moving control point P_i or changing weight w_i affects the curve only in the interval $u \in [u_i, u_{i+p+1})$. NURBS therefore make use of weight modification and control point movement for interactive shape design.

The concept of homogeneous coordinates, obtained by mapping the coordinates in 4D space described for the Rational Bézier curves holds also for NURBS.

NURBS surfaces As already mentioned parametric descriptions can easily be extended into the third dimension, yielding a surface descriptions. This is one of the advantages of the parametric representation. The formula for surfaces reads:

$$S(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}\boldsymbol{P}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}} \qquad 0 \leq u,v \leq 1 \tag{2.27}$$

The control points form a bidirectional (rectangular) control net $\boldsymbol{P}_{i,j}$, weighted by the weights $w_{i,j}$. The basis functions are of degree p and q respectively and defined on the knot vectors U and V. The properties for the product of rational basis functions $R_{i,j}(u,v)$ are more or less equal to the ones if looking at a single basis function. The same holds for the geometrical properties, given by:

- Corner point interpolation of points $\boldsymbol{P}_{0,0}, \boldsymbol{P}_{n,0}, \boldsymbol{P}_{0,m}, \boldsymbol{P}_{n,m}$
- Affine invariance: apply transformations to control point net
- Strong convex hull property: For $(u,v) \in [u_{i_0}, u_{i_0+1}) \times [v_{j_0}, v_{j_0+1})$ the surface $S(u,v)$ is in the convex hull of the control points $\boldsymbol{P}_{i,j}$, $i_0 - p \leq i \leq i_0$ and $j_0 - q \leq j \leq j_0$.
- Local support
- Differentiability: $S(u,v)$ is $(p-k)$-times differentiable w.r.t. u at a u-knot of multiplicity k, respectively for v.

There is in contrast to curves no variation diminishing property of NURBS surfaces!

Trimmed NURBS surfaces NURBS surfaces have an underlying rectangular net of control points defining the shape of the surface. Shapes and surfaces where the interpolated endpoints are not adjusted like a rectangular cannot be modeled by NURBS surfaces. This fact limits the flexibility and practical application of this representation dramatically. A remedy for that problem are trimmed NURBS surfaces (also applicable for trimmed curves). Trimmed NURBS surfaces are defined by a regular NURBS surface and a set of closed trimming loops in the parameter space of the surface to define boundaries or holes for the surface. Parts of the surface are cut from the domain which actually is supposed to be the real surface by being situated outside the trimmed boundary or inside a hole in the surface. The trimming loops consist of a sequence of curves (NURBS or other). The orientation of this sequence of curves defines if the loop formed by the curves describes a boundary or a hole (internal boundary).
Picture 2.21 shows a surface of a violin body, constructed by trimmed NURBS.

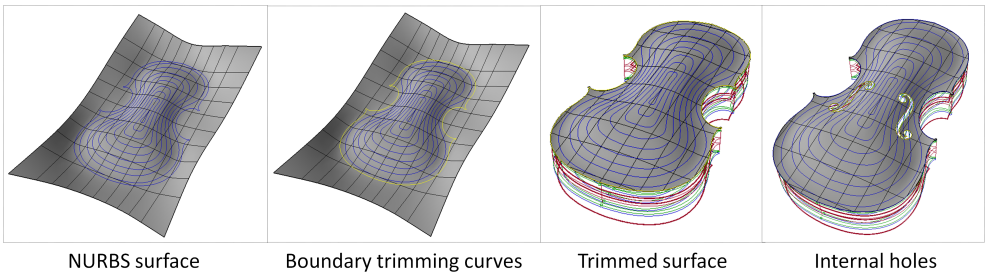


Figure 2.21: Trimmed NURBS surface model describing a violin top plate

Trimmed NURBS surfaces are extremely flexible and applicable for almost all kind of shapes. The concept of trimmed NURBS is the main reason for the wide range of applications and usage. Unfortunately trimmed NURBS have a massive drawback. Parametric trimming curves are mathematically incapable to represent the intersection between NURBS surfaces exactly unless they are of a very high degree. Therefore two NURBS surface patches with a common boundary defined by a trimming curve might not be mathematically watertight, which means the intersection curves on both surfaces might have overlaps or small gaps. Picture 2.22 shows an example for this kind of problem.

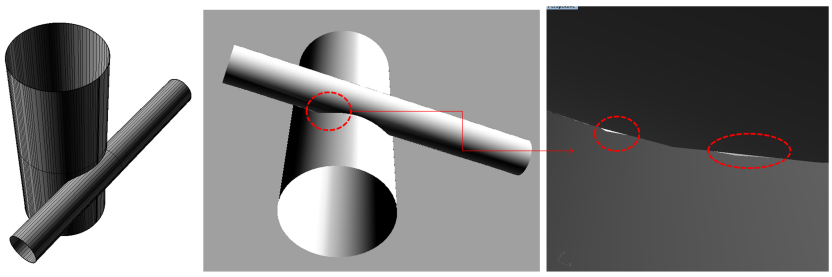


Figure 2.22: Trimming and non-watertight intersections

Non-watertight interfaces between NURBS patches can cause problems in many downstream processing algorithms like ray casting for example. In fact non-watertight gaps and overlaps between trimmed NURBS surface patches was identified as the "single greatest cause of poor reliability of CAD systems" (see [20]).

T-Splines surfaces T-Spline surfaces address the inability of trimmed NURBS surfaces to form watertight intersections. Especially meshing-free techniques suffer from non-watertight trimmed NURBS as described in [16] and motivated originally the development of T-Splines. The reader is referred to [42] for a detailed description of T-Splines. At this point the concept is only roughly described.
The shortcomming of NURBS surfaces is the intrinsic rectangular control point grid, which reduces the flexibility of this kind of description and motivated the idea of trimmed surfaces. T-Splines overcome this drawback by allowing non-rectangular control point nets. The "mesh" formed by the T-Spline conform control points is called a T-mesh. The name already indicates the possible presence og T-junctions in the mesh. In case a T-mesh forms a rectangular grid without any junctions it degenerates into a B-Spline surface. Picture 2.23 shows a T-mesh as control point grid for T-Spline representation.

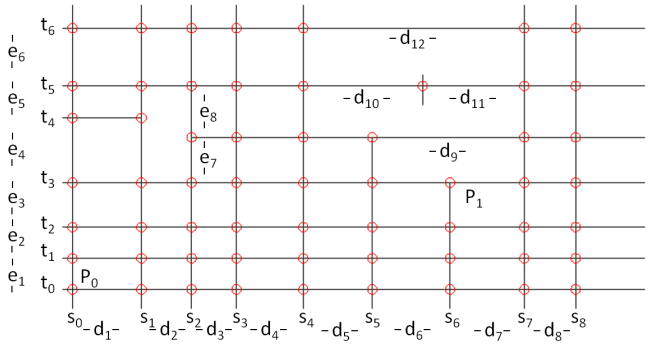


Figure 2.23: Example for a T-Mesh

The mesh is defined by knot intervals d_i and e_i. A valid T-mesh must assure that the sum of all knot intervals on one side equals the sum on an opposed side at any position of the mesh. The mathematical description of the surface equals the one of B-Splines, since the same basis functions and knot vector concepts are used. The difference lies in the procedure to determine the knot vectors for each basis functions at the breakpoints

of the mesh. The knot vector is not globally defined but rather defined by the local coordinate system in the vicinity of position $\boldsymbol{P}_i$ using the following construction rule:
The basis function assigned to control point $\boldsymbol{P}_i$ is determined using the knot coordinates at $\boldsymbol{P}_i$, given by (s_{i2}, t_{i2}). A ray in parametric space $\boldsymbol{R}(\alpha) = (s_{i2} + \alpha,\ t_{i2})$ is constructed. The intersection of that ray with the first two s-isoline yield the knot coordinates s_{i3} and s_{i4}. The other coordinates are found likewise. After all knot coordinates are defined the basis functions can be computed and the parametric formula is determined.

2.3 File formats and exchange of geometric information

In section 2.1 it was shown that the workflow from problem definition, modelling, processing and finally interpretation and visualization of results might include many sub-steps or even sub-systems. For practical engineering problems there are usually several systems involved in the workflow. The systems might differ between the sub-steps as well as between processing routines.
In the automotive industry for example the intrinsic differences of specific sub-tasks motivate the usage of various different CAD/CAM/CAE systems. Systems for car body design might be very different from those used for electronic or interior design. The same holds for civil engineering. Architectural design and engineering analysis like structural analysis or HVAC-design is usually performed on different models, different levels of details or even models with different dimensionality.
There is a high demand for efficient documentation of design and production steps as well as a need for efficient exchange of information between the group members and involved systems. The exchange of information requires often an interface, such that two systems or software components can communicate with each other. This holds especially for the exchange of geometric information and models. The exchange of geometric models is related to several problems listed below. Especially the exchange of models based on classical CAD systems is listed amongst the ten most challenging problems for CAD (see [26]).

- Drawing and document conventions: different conventions for organizing drawings or documents might be apparent. This includes grouping of information in layers or groups, or the interpretation of color information.

- Numerical Precision/Inaccuracies: Different systems might work with different numerical accuracy which can lead to significant loss of data or misinterpretation of geometric structures.

- Upward compatibility: In different systems different level of detail might occur or different information in general might be assigned to the entities. It is wise and recommended to create upward compatible data exchange formats. Otherwise exchange formats are not flexible towards specific needs of different systems and even compatibility with older versions of the format might not be guaranteed.

- Specification language: National standards for data exchange might include language specifications leading to misunderstandings or data mismatch.

- Mismatched subsets: Due to differences in functionalities necessary data assigned to entities for a receiving system might not be included in the model or deleted from earlier stages of the model by a prior system in the chain.

- Mapping: It is essential to map data from a system's database in the format of exchange standard.

- Costs: The cost for data exchange should be moderate especially if many different formats are included in the production workflow. Costs include costs for hardware, translators, software and time.
- Development: In general the will to develop translators and data exchange applications is limited. This holds especially for translators to file formats of competing systems. Often software development companies try to establish their own file formats and sell entire solution systems for several sub-steps in the workflow. Therefore neutral file formats are preferred for data exchange.

There are several approaches to realize exchange of geometric information. A trivial remedy is to ensure that all parties involved in the workflow use the same or totally compatible systems as sketched in figure 2.24. Especially component suppliers or sub-contractors might be forced to use systems compatible to those of their clients. The disadvantage is that huge and complex systems have to be developed and maintained for the entire cycle. Furthermore this concept is contra productive concerning market competition or highly specialized systems.

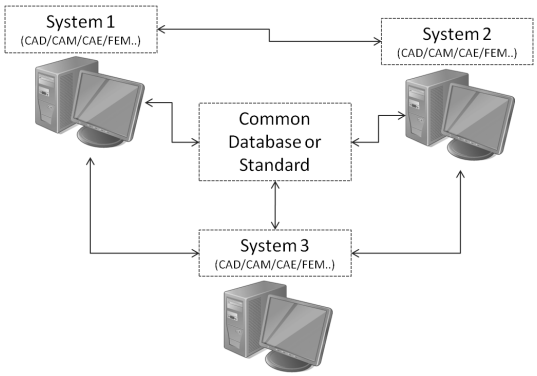


Figure 2.24: Information exchange via common systems and standards

Usage of direct translators is another approach as seen in picture 2.25. They enable communication between exactly two standards or systems. Only small and rather simple software components are necessary. A drawback is that many translators are needed in case various systems or standards are used in the workflow. In this case the costs might increase dramatically. The costs for one translator can be distributed on only two systems. Furthermore development of direct translators between competing systems might be difficult since access to information about data structure or conventions might be restricted to third party developers or competitors.

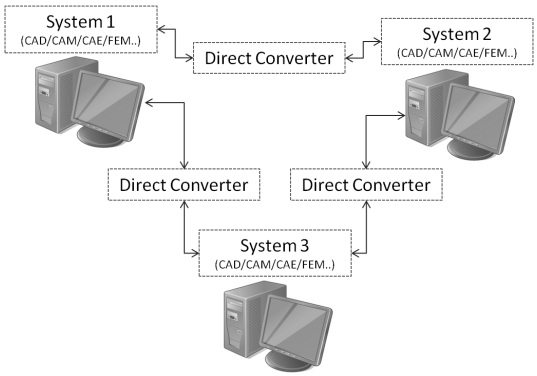


Figure 2.25: Information exchange via direct translators

Neutral exchange formats for system communication and exchange of information implies that all information can be translated into a system-independent standard. Every system or software component has to provide so-called half translators, translating its data structure into the neutral standard. Advantageous is the fact that confidential information about a software structure is not to be published for development of direct translators. Furthermore the software developers of a system have a deep insight in their own structure to develop high-quality translators. The problem is that the definition of neutral file formats might be quite complex and cumbersome. Often boards or institutions have to be created to administrate and regulate standardization and development of the exchange standards. Administration and research has to be funded by different companies.

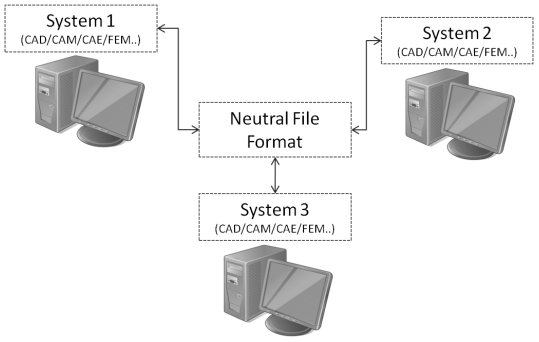


Figure 2.26: Information exchange via neutral file formats

The development of neutral file formats was pushed by public clients in the area of defense amongst others. Suppliers and service provider for engineering tasks in those fields where forced to provide information in neutral file formats. Out of that widely used exchange standards like IGES (Initial Graphics Exchange Specification, see [1]) or STEP (Standard for the Exchange of Product Model Data [7]) were developed.
The development of IGES started around 1980 and is since then under frequent development. This format is motivated by exchange of geometric information between CAD-software. Almost every CAD-system providing a half-translator can work with the IGES format. The field of application for the IGES format is defined by the U.S. Product Data Association like that: "This Specification provides the framework for communicating the essential engineering characteristics of physical objects called products."(see [47]). A product is defined by the following characteristics according to IGES-specification:

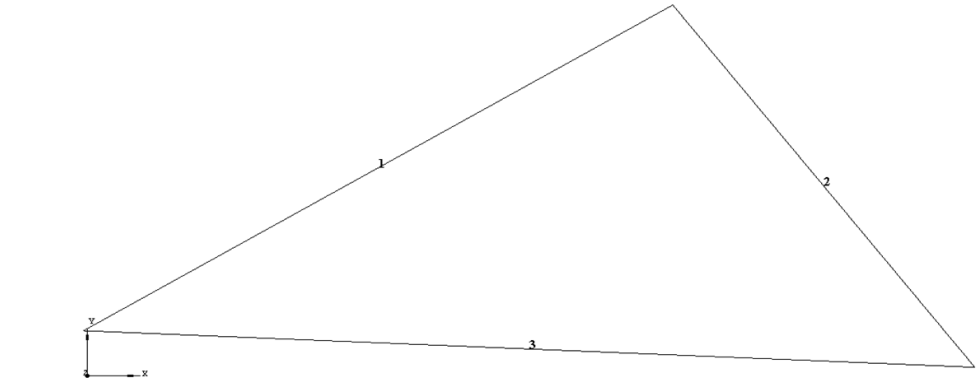
- ADMINISTRATIVE
  - · Product Identification
  - · Product Structure
- DESIGN/ANALYSIS
  - · Idealized Models
- BASIC SHAPE
  - · Geometric
  - · Topological
- AUGMENTING PHYSICAL CHARACTERISTICS

  - Dimensions and Tolerances
  - Intrinsic Properties
- PROCESSING INFORMATION
- PRESENTATION INFORMATION

The exchange files are organized in six sequentially numbered sections:

1. **Flag:** Optional section used only when remainder of file is in compressed ASCII or binary form. This section is not used if written in ASCII format.
2. **Start:** Sender comments. Free form to pass comments and any kind of information to another system.
3. **Global:** General file characteristics and information of post- and preprocessors like accuracies or units.
4. **Directory Entry:** Index of all entities used in the file, like circles, lines or surfaces. References to the geometric specification are assigned to the entities. Additional common attributes (non-geometric attributes like color) are defined here.
5. **Parameter Data:** Geometric attributes for all entities defined in directory section are stored here.
6. **Terminate:** Check sums for all data in each section

Picture 2.27 sketches a typical IGES file




```
1                                                                  S      1
2 1H,,1H;,,                                                        G      1
3 40HC:\Users\Felix\Desktop\IGES_examplel.igs,                     G      2
4 26HRhinoceros ( May 12 2010 ),31HTrout Lake IGES 012 May 12 2010,G      3
5 32,38,6,308,15,                                                  G      4
6 ,                                                                G      5
7 1.0D0,2,2HMM,1,0.254D0,13H111014.085348,                         G      6
8 0.001D0,                                                         G      7
9 14.0128878387454 4D0,                                            G      8
10 ,                                                               G      9
11 ,                                                               G     10
12 10,0,13H111014.085348;                                          G     11
13      314       1       0       0       0       0       0       000000200D     1
14      314       0       1       1       0       0       0  COLOR        0D     2
15      406       2       0       0       1       0       0       000000300D     3
16      406       0      -1       1       3       0       0LEVELDEF      0D     4
17      126       3       0       0       1       0       0       000000000D     5
18      126       0      -1       8       0       0       03d BsCrv      0D     6
19 314,0.0,0.0,0.0,20HRGB(   0,   0,   0 );                       0000001P     1
20 406,2,1,8HStandard;                                             0000003P     2
21 126,3,1,1,1,1,0,0.0D0,0.0D0,14.59575457152352D0,                0000005P     3
22 25.25664388353793D0,44.03885963921933D0,44.03885963921933D0,    0000005P     4
23 1.0D0,1.0D0,1.0D0,1.0D0,-14.0128878387454 4D0,                  0000005P     5
24 -3.493409013020573D0,0.0D0,-1.609323253189251D0,                0000005P     6
25 4.199941172957315D0,0.0D0,4.749466186241452D0,                  0000005P     7
26 -4.356948319609928D0,0.0D0,-14.01288783874544D0,                0000005P     8
27 -3.493409013020573D0,0.0D0,0.0D0,44.03885963921933D0,0.0D0,     0000005P     9
28 0.0D0,1.0D0;                                                    0000005P    10
29 S0000001G0000011D0000006P0000010                                T      1
30
```

Sender comments

Global: General file characteristics

Directory Entry

→ Definition of 3 curves

Parameter Data

→ Geometric attributes of curves

Check sums

Figure 2.27: Typical structure of an IGES file

The IGES file format definition suffers from several intrinsic problems.

- Reading of IGES files can be quite slow, since the file format holds forward pointers from the directory section to the parameter section. Therefore the preprocessor has to jump within the file while reading in.
- Early versions did not provide the functionality to compress data, resulting in huge files.
- In practical application it turns out that many half-translators (pre- and postprocessors) are not totally compatible with the IGES specification. Therefore geometric models might be modified during import or export or loss of information might occur. Translation of volumetric information into or out of IGES is not possible in many investigated modelling systems, although theoretically possible in the IGES format. Therefore this information is simply lost after exchange via IGES. Another problem occuring frequently is the decomposition of rational curves into shorter non-rational segments. This problem will be described in detail with the help of examples later in this thesis. Picture 2.28 shows a geometric surface in the original modelling software and after the exchange via IGES.

STEP is another widely used exchange standard developed since approximately 1990. The development of STEP was motivated by the will to establish an international format based on

international standards (ISO) [2]. Furthermore the point of view is significantly different for the STEP format. The origin of IGES clearly lies in communication between CAD-systems. The STEP format in contrast addresses the entire life-, design- and production-cycle. All data and information within this cycle are to be represented sufficiently and efficiently. Communication with database systems was an important aspect for this format.

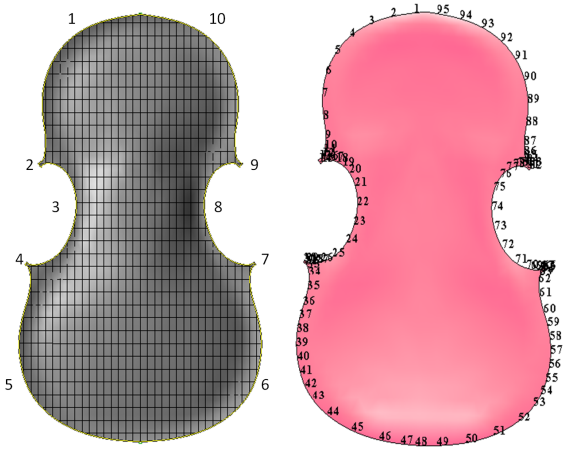


Figure 2.28: Boundary decomposition due to exchange via IGES

STEP combines the physical and functional description of a product. The product description is realized with the standardized specification and data modelling language EXPRESS. The standardization is defined in ISO 10303 and simplifies the exchange and communication of information. Access to the data is realized with the SDAI interface (Standard Data Access Interface). So-called application protocols (AP) form the interfaces and data type layouts to describe one specific problem or product. Nearly all types of problems and applications in engineering and technical environment have an application protocol. AP 214 used in the automotive industry or AP 203 for CAD-data are important protocols for example. The specification of STEP includes several parts as listed below in picture 2.29.

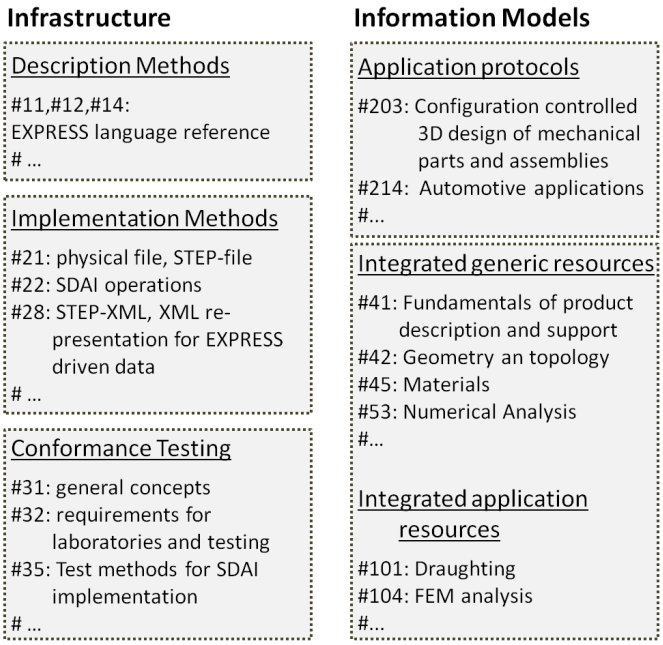


Figure 2.29: STEP specification

For a detailed description of STEP the reader is referred to [7] or [8]. An illustrative comparision of IGES and STEP is found at [38]. The idea to document and include all information within a life-cycle in the file format might be interesting for healing purposes since information about geometric manipulations due to healing routines can be stored alongside with the original geometric model such that at any time in the workflow the original geometric data can be

retrieved.

2.4 Meshing and discretization

As described in section 2.1 problems of modern engineering are often solved using numerical methods. Numerical methods provide a solution technique for the equations of motion related to mechanical systems. To apply numerical methods a discretization step of the problem formulation is required, including a discretisation of the physical domain unless meshing-free techniques are applied. The total potential energy of the system $\Pi = U - W = 0$, with the internal energy $U = \frac{1}{2}\int_\Omega(\sigma\varepsilon\ d\Omega)$ and external energy W can be decomposed into contributions of finite elements. We finally end up with the well-known FEM equations (see also [25]):

$$\begin{aligned} \delta\Pi &= \delta U^{(e)} - \delta W^{(e)} = 0 \\ &= \frac{1}{2}u^{(e)}K^{(e)}u^{(e)} - u^{(e),T}f^{(e)} = 0 \end{aligned} \tag{2.28}$$

with:

$$K^{(e)} = \int_\Omega B^T E B\ d\Omega \tag{2.29}$$

and:

$$B = DN \tag{2.30}$$

Where E holds the constitutive law and D is a differential operator applied on the so-called shape functions N assigned to the finite elements. Therefore in classical FEM the domain has to be approximated by a finite number of elements. In isogeometric analysis the representation of the geometric model itself provides a discretization and shape functions N. In any case the resulting discretized domain holds degrees of freedom at discrete points in space (nodes of the FE mesh) and is equipped with shape functions to describe the deformation patterns over the entire domain.
The mesh generator has to decompose the domain in such type of elements requested by the FEM formulation. There is a wide range of FEM-formulations sketched in picture 2.30 with the corresponding shape and topology of element.

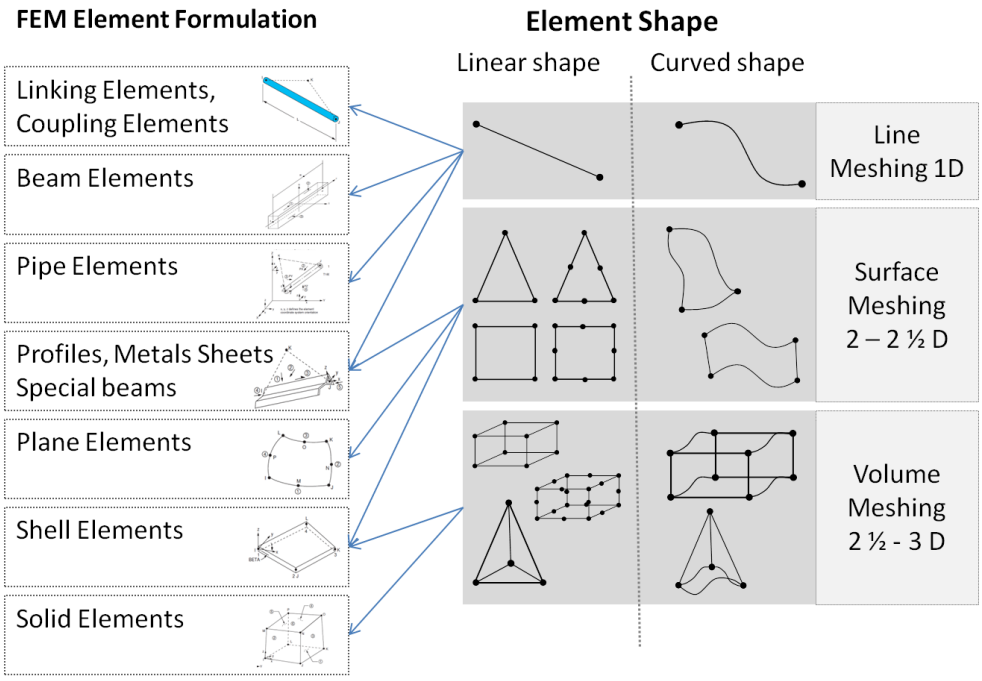


Figure 2.30: FEM formulation and element shapes

Furthermore meshes can be grouped into:

- Structured meshes: regular connectivity, which can be expressed by a 2D or 3D array
- Unstructured meshes: irregular connectivity
- Graded meshes: gradient in mesh density apparent
- Uniform meshes: uniform mesh density / element size
- Radial meshes: radial alignment of elements

Meshing is a critical step in the engineering workflow for numerical simulations. The mesh size might have a significant influence on the accuracy of the results. Furthermore the mesh generator acts directly on the geometric model. Therefore it is directly influenced by errors and inconsistencies in the model. In fact this is quite often the case for complex models in industrial applications and therefore mesh generation is still a time-consuming and costly process.
The kinematic characteristics of the model have to be included in a suitable discrete formulation. Let us consider problems of civil, mechanical or marine engineering were thin-walled structures often appear. Thin-walled structures can be modelled as shells. Due to the practical relevance there exist closed-form FEM formulations for shell elements (see [14]). For a suitable formulation the dimensionality of the geometric model as well as the FEM-elements plays an important role. Shell elements can be formulated as plane elements or solids as sketched in picture 2.31. To construct the elements the geometric model has to provide the necessary information. The same dimensionality is not necessarily required if additional information is assigned to the model. In our example if plane elements are used in the FEM formulation a surface mesh is sufficient. The information about the shell thickness has to be provided separately. On the other hand if 3D solids are used as shell elements it does not imply necessarily a geometric model in 3D. If the model is of 3D volumetric type (BREP volumes) volume meshing can be applied. But there exist also methods like extrusion or sweeping where a $2\frac{1}{2}$ D model is sufficient. The plane surface mesh can be transformed into a 3D solid mesh by different types of extrusion:

- Normal extrusion: all nodes on a midsurface are extruded in normal direction by a global offset.
- Global extrusion vector: all nodes on a midsurface are extruded in direction of a global vector by a global offset.
- Intersection extrusion: all nodes are projected in normal direction from a midsurface onto an upper and lower shell.

Picture 2.31 sketches possible approaches for a shell FEM formulation.

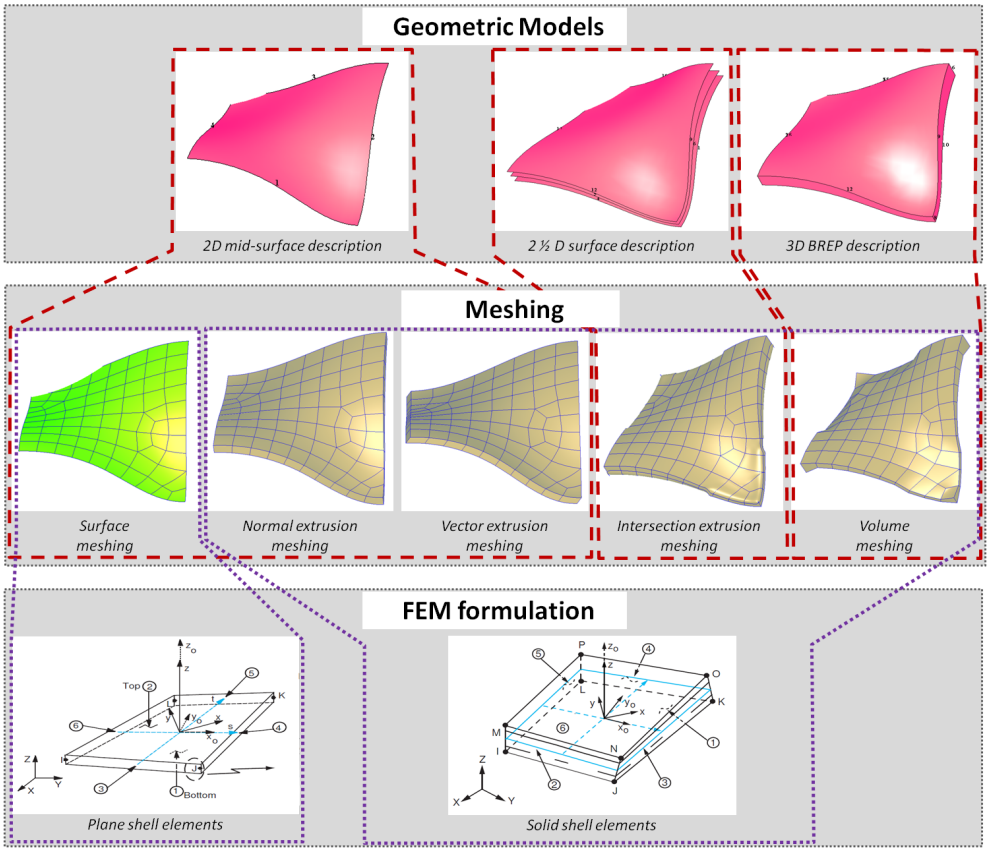


Figure 2.31: FEM elements and meshes for shell-like structures

It is important to mention that errors and inconsistencies in the model are likely to cause the meshing algorithm to fail or perform poorly. Therefore healing routines might function as pre-processor to supply the mesher with suitable geometric representations. On the other hand the meshing routines might introduce errors themselves resulting in an defective or poor approximation of the model. Therefore the healer can have the function of a post-processor after meshing. Even an interactive and iterative approach between healing and meshing routines is possible. The possible relations between healer and mesher are depicted in picture 2.32.

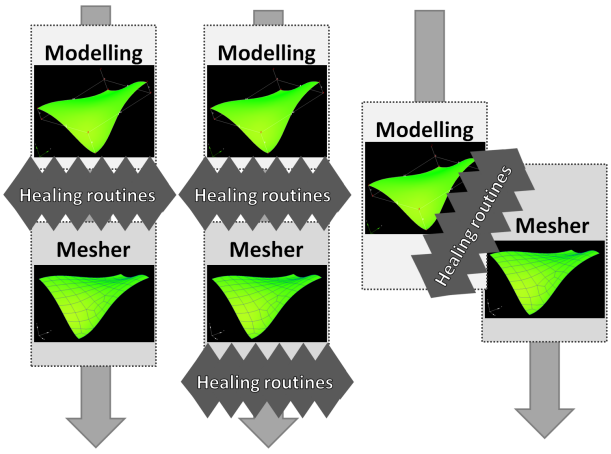


Figure 2.32: Location of the healer in the discretization process

2.5 Healing routines in the engineering workflow

There are several positions in the engineering workflow where healing routines might be located to ensure the absence of errors and defects in the model. A more detailed classification of healing routines and algorithms will be given in chapter 3. At this point it is already mentioned that healing can refer to topological and geometrical errors and defects. Healing routines might be applied to models for geometric representation or discrete models. Within the engineering workflow sketched in section 2.1 several exchange file formats and systems might be used. Translating and mapping of geometric information into different formats might be one reason for inconsistencies and errors. Therefore after each transformation or translation step the models might be checked against conditions of validity. All interface points are possible locations for healing routines, depicted in picture 2.33.

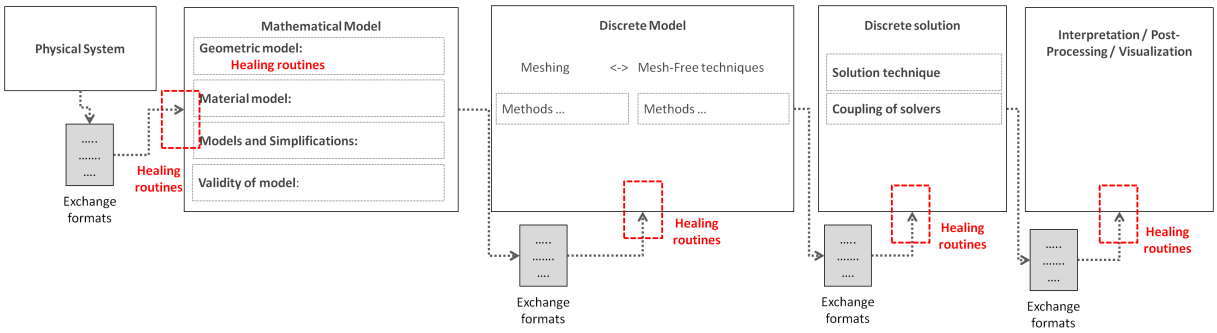


Figure 2.33: Healing routines in the workflow

Other sources of geometric errors throughout the workflow might arise from the algorithms itself. Modelling software often makes intensive use of geometric operations like trimming or projections. Each routine of that kind can result in model defects which violate conditions of validity or just decrease the accuracy of the results. Errors or inconsistencies are often not visually detectable. Therefore healing and checking routines are necessary to reveal insight on possible model errors and inconsistencies. The same holds for meshing algorithms or other discretization techniques which might result in defected models and require healing routines. Especially in meshes for very big objects, the user is dependent on robust routines due to the

amount of data. Visualization can be another step in the engineering workflow where healing algorithms might be applied to produce meaningful visual outputs to interpret the obtained results.
This section closes this general chapter on the engineering workflow, geometric modelling, data exchange and the reference of those aspects to healing routines. The purpose was to motivate the necessity of healing algorithms and routines for efficiency in modern engineering applications.

Chapter 3

Theoretical aspects of healing routines and algorithms

3.1 Modelling errors and healing approaches

The purpose of this chapter is to illustrate the types and characteristics of geometric errors and inconsistencies in detail combined with existing approaches to overcome gaps in the engineering workflow due to a defective model. This chapter sets the basis and motivation for the implementation of own healing routines into the existing TUM.GeoFrame framework as it will be presented in the following chapters. Defects resulting in a "dirty" geometric model will be listed in detail. In addition this chapter provides the reader with a theoretical background on the problems, operations and healing approaches related with model errors and inconsistencies.

3.1.1 Errors and inconsistencies of geometric models

Preparing geometric models for numeric simulations and analysis routines is an important field of research and industrial development. The need for healing routines became apparent alongside with the development of CAD systems. The proceedings achieved over the last years related to computer aided design and modelling software did not solve the problems of defective models but rather introduced more problems motivating the heavy interest in robust healing routines. One source for model errors is related to data exchange illustrated also in section 2.3. Translation problems or precision mismatches might change the structure and characterisitcs of the geometric model. Other reasons for defective geometric models are modelling operations itself or errors introduced by the user. In any case the model has to be cleaned from errors and inconsistencies in so far as they influence the downstream analysis processes. The importance of healing routines becomes apparent in many practical applications where the data repair and healing routines often turn out to be the bottleneck while downstream routines like meshing or analysis perform stable and efficient as stated on basis of an example in [19]:"Ken Morgan of the University of Wales, Swansea, showed impressively detailed meshes [...] for fighter aircraft [...]. He presented the following typical breakdown of the effort in a realistic CFD analysis: 1-4 weeks for geometry repair and preparation, 10-20 minutes for surface meshing, 3-4 hours for volume meshing, and about 1 hour for the actual flow analysis."
Problematic is the fact that the consequences of errors or inconsistencies might not be foreseeable at creation time of the model. Furthermore many errors are hard to detect since geometric models might not reveal embedded defects by visual inspection. To judge the consequences of errors it is necessary to know at creation time of the model which type of analysis and routines are to be applied on a model. While this is usually the case in research projects this might not be so clearly defined within a complex industrial analysis workflow. The exact methods and

analysis routines might not be known at creation time of the model. The cycle can cover a long period and within it might turn out that the analysis workflow or the model itself has to be adjusted. Therefore the existence of errors and inconsistencies in the model is a critical problem although not always critical with respect to specific processing routines.
A major aspect related to healing routines is to define when a geometric model is defective or vice versa when a model can be considered to be correct, consistent and clean. Although intuitively rather easy to formulate this question might not be trivial. The problem of error analysis has produced a huge body of literature over the last decades. A detailed formulation of the problem can also be found in [23] and [10]. Modelling errors are to be seen with respect to modelling concepts, which motivated the review on different schemes presented in chapter 2. The knowledge which rules and conventions a correct model description has to follow enables us to analyse the presence of violations resulting in a defective and inconsistent model. As illustrated in section 2.2 a modelling concept includes on the one hand the questions which modelling scheme to use. The indirect BREP concept is such a modelling scheme for example. In addition the way to describe geometric shapes is covered by the modelling concept. In this thesis we will make heavy use of parametric NURBS descriptions. Furthermore the modelling concept is also influenced by the downstream applications which the model should be used for. Especially the last point can cause problems within healing routines.
Let us consider the case of a BREP modelling scheme. The conditions of validity are clearly defined and require manifold cell complexes. A closed region defined by a closed boundary wire might form such a valid cell complex. Things become more complicated if this region is modified in a way to be able to carry information needed during the analysis process. For example inner lines might be required to assign boundary conditions or loadings. From the viewpoint of BREP modelling however these inner lines are not necessary to describe the model and are to be seen as isolated entities not contributing directly to the model unless these lines are closed and form an inner hole. This example is rather simple but it demonstrates the problem that healing routines purely related to conventions and conditions of the modelling schemes might not result in the desired and required output. The healing routines rather have to be adjusted to the model characteristics and modelling concept. Unfortunately there is no clear definition in such a sense that healing routines are clearly and unambiguously definable. This ambiguous character of geometric model information is the most challenging aspect of healing routines.
Before approaches to overcome the bottlenecks in the workflow due to "dirty models" will be presented in the next section, a collection of well-known errors and inconsistencies within geometric models are presented to the reader in a first step. It will turn out that the defects can be grouped and classified in relation to the used modelling scheme. Since the BREP scheme is used within the implemented routines the classification will strongly be related to the BREP concept in the following.
The reason to concentrate on BREP schemes clearly lies in the wide range of application in nowadays tools related to engineering analysis since this scheme is a flexible and powerful method. However many CAD systems make use of the constructive solid modelling (CSG) scheme. This scheme is quiet intuitively to use during modelling by means of basic Boolean operations and possible extended operations like scaling or rotations. CSG models can be stored in trees holding the information about primitives and operations resulting in a final spatial object. Although intuitive for modelling this scheme is not well-suited for exchange of geometric information. Therefore even CAD systems often have a hybrid system or transfer CSG model into a BREP structure for data exchange.
Following the conventions in the BREP scheme model errors and inconsistencies can be grouped into geometric errors and topological errors. Geometric errors are universal errors which might appear in all types of modelling schemes. Topological errors in contrast have a stronger link to the underlying modelling concept. The following table 3.1 gives an overview over typical errors

classified into geometric and topological defects.

Modelling defect	Specification of defect	Source	Classification of error	Consequences
Non-consistent node disjointedness (Double nodes)	Nodes of neighbouring entities like edges are disjoint but actually should be one and the same node	Often related to precision conflicts, model exchange or introduced by the user	Geometric	Violation of validity conditions (VC) if cell complexes are not closed anymore; loss of consistency
Non-disjoint but distinct nodes (Double nodes)	Nodes are not disjoint (within a defined tolerance) but topologically distinct.	Often related to translation problems	Topologic	Violation of topologic VC; loss of consistency
Non-consistent edge disjointedness (Double edges)	Edges of neighbouring faces are disjoint but actually should be one and the same edge	Often related to precision conflicts, model exchange or introduced by the user	Geometric	Violation of VC if cell complex closure are destroyed or non-consistent edge intersections appear.
Partially non-consistent edge intersections (Overlapping edges)	Distinct edges have non-consistent intersections, therefore are partially non-disjoint; harder to detect than tottaly identical double edges	Often related to modelling errors introduced by the user or due to model exchange	Geometric	Violation of geometric VC; loss of consistency
Singular non-consistent edge intersections	Edge intersection between to edges in a singular point where no vertex is placed	Often related to modelling problems introduced by user or modelling software	Geometric	Violation of geometric VC
Endpoint sloppiness	End points of edges geometrically not lying on edge path	Data exchange or failed modelling operations (trimming)	Geometric	Critical for geometry related operations within workflow; Violation of geometric VC (not topologically)
Edge self-intersections	Self-intersections along edge path due to geometric representation	Problems related to modelling operations	Geometric	Critical for geometry based operations within workflow; Violation of geometric VC (not topologically)
Non-disjoint but distinct edges (Double edges)	Edges are not disjoint (within a defined tolerance) but topologically distinct.	Often related to translation problems	Topologic	Violation of topologic VC; loss of consistency

Non-consistent face disjointedness	Faces of neighbouring volumes are disjoint but actually should form one and the same face	Often related to precision conflicts, model exchange or introduced by the user	Geometric	No violation of VC! But loss of model consistency since neighbouring relations are lost
Non-consistent face intersections	Face intersections due to geometric representation	Problems related to modelling operations	Geometric	Critical for geometry based operations within workflow; violation of geometric VC
Non-consistent face self-intersection	Face self-intersection due to geometric representation	Problems based to modelling operations	Geometric	Critical for geometry related operations within workflow; violation of geometric VC
Non-disjoint but distinct faces (Double faces)	Faces are not disjoint (within a defined tolerance) but topologically distinct.	Often related to translation problems	Topologic	Violation of topologic VC; loss of consistency
Infinite faces	Faces with infinite area due to missing bounding wires	Often related to data loss during model exchange	Topologic	Loss of consistency and problematic for operations in workflow
Sliver surfaces	Faces with very small area and no meaningful modelling purpose	Leftovers of geometric modelling operations	Geometric	No violation of VC but increase of model volume and problematic for geometry based operations
Boundary-wire sloppiness	Wires forming face boundaries geometrically not lying on the face	Often related to modelling errors or translation problems	Geometric	Critical for geometry based operations within workflow; violation of geometric VC
Gaps and overlaps between faces	Presence of geometric gaps or overlaps between neighbouring faces due to geometric descriptions	Often related to modelling operations or user-introduced errors	Geometric	Violation of geometric VC, problematic for geometry based algorithms (isogeometric analysis, ray casting)
Spurious entities	Edges or faces with infinite or not defined underlying length or area	Encountered often after exchange via neutral file formats	Geometric	Problematic for geometric algorithms

Spare entities	Entities like nodes, edges or faces not contributing to a meaningful model: dangling edges	Leftovers from modelling operations or user introduced	Topologic	No direct violation of VC if all distinct entities are disjoint, but blowing up the model and maybe critical for topologic analysis
Missing entities	Missing entities to form closed complexes, entities not present in model	User-introduced problems or translation problems	Geometric	Violation of geometric VC, problematic for operations like inside-outside tests
Entity assignment mismatch	Mismatch of topologic assignments or even missing assignments, for example topologically missing faces which are geometrically present in the model	Related to the user or translation problems	Topologic	Violation of topologic VC
Excess details	Model information that can be expressed by much less data or simpler representation	User related or introduced by geometric operations of modeller, due to missing geometric representation schemes of kernels	Geometric	No violation of VC but blow up of model volume
Orientation problems	Orientation of entities like faces not computable, for example due to improper parameterization	Problems of geometric representation scheme	Geometric	Violation of basic requirements for BREP models
Orientation mismatch	Wrong geometric orientation of entities like edges or faces, for example inward pointing face normal vectors forming a volume	Modelling errors not considering model schemes conventions	Geometric	Critical for geometry based operations like mesh generation
Orientation mismatch	Wrong topology related orientation of entities, for example orientation of a wire differentiating holes or boundaries	User introduced or translation problems	Topologic	Mismatch of model interpretation

The list of errors does not claim to be complete. It can be assumed that almost everybody who ever worked with geometric models encountered some of the listed defects above. The consequences for downstream applications can be quite different regarding the various defects and errors and are worth having a closer look.
There are defects purely increasing the problem size and therefore might blow up the processing time or cause inefficiencies. These defects do not violate conditions of validity of the modelling scheme and therefore they might be described by having a "weak" influence on the workflow. Weak is to be understood in the sense of not directly endangering the success of downstream applications. Nevertheless the consequences can be significant in case the processing time increases dramatically. An example might be a linear surface described parametrically by NURBS, whereas a purely linear description would be much easier. If the representation information is carried along the workflow the excessive description might lead to an increase of data volume or influence the processing time of geometry based algorithms. In case a pFEM analysis is applied the representation is also a question of minimizing the problem size, respectively the degrees of freedom. However excessive shape representations might not only be a question of efficiency, they might cause serious numeric problems within the workflow as illustrated with an example in section 6.3. This example demonstrates the influence of spatial boundary decomposition and enforced mesh refinement on discretisation and analysis. This example reveals that an insight on applied downstream applications is necessary to judge on the influence of modelling defects.
In contrast to weak defects there are errors and inconsistencies clearly violating crucial conditions of validity in a strong sense. Non-consistent intersections and self-intersections of edges and faces are classical examples for this type of defects. Such errors will cause problems throughout the workflow in any case and have to be erased to prepare the model for numeric simulations. Unfortunately this type of defects is usually hard to detect due to geometric complexity although clearly defined. Intersections or overlaps of trimmed NURB surfaces for example are a huge challenge for healing algorithms.
Gaps between adjacent faces due to trimming or modelling defects are of the same type than overlaps but however much more difficult to handle. A detection algorithm struggles with geometric complexity on the one hand. On the other hand the problem is of an intrinsic ambiguous character. A void area between faces can be a desired hole or an undesired gap and an algorithm has to decide with respect to defined tolerances.
The situation is even more complicated in case of double objects like double nodes, edges or faces in the model. Theoretically they violate the conditions of validity and therefore a clearly defined rule is available to detect them and heal them. However double entities might be desired by the problem formulation to make neighbouring relations of objects inconsistent. Applications for inconsistent relations might be contact problems or material nonlinearities. This problem can only be solved by user interaction. The same holds for missing entities in the model not directly violating the conditions of validity. For example a topologically missing inner wire in a region which is at the same time a boundary of another region might be such a case. Without the inner wire the consistency is not guaranteed between the two objects. In a strict sense also conditions of validity are violated since there is a non-consistent object intersection without sharing common entities. The true model as desired by the user is however unrecognizable to a computational algorithm and the healing routine will be confronted with an ambiguous situation.

3.1.2 Classifications of healing approaches

The previous section listed a range of frequently occurring defects in geometric models. An insight on the character of the defects was provided. A robust healing routine has to consider all of these intrinsic characteristics. This section will present an overview over different approaches for healing routines as found in literature.

Chapter 2 section 2.5 illustrated that healing routines might be necessary or useful at different locations throughout the workflow. Cleary one location might be within the modeller at creation time of the geometric model. The model can be tested against errors or inconsistencies resulting from failed operations or introduced by the user immediately. Other locations are possible basically whenever model data is exchanged to another system. If the defects arise due to data exchange and not to modelling itself this is the only possibility to address them. Classically in a workflow with included meshing routines the healer is located before or around the mesher. In this case there are three locations possible as sketched in section 2.5. The healer can be placed before or after the mesher or interactively perform during meshing routines. There are strengths and weaknesses of every possible approach sketched in the following. The question which concept to apply most efficiently often depends on the characteristics of the model itself.

Pre-processing healing routines

In this concept the healing routines are applied directly on the geometric model before meshing routines are called. Strength of this system is that the healer is solely dependent on the geometric model and not on robustness of another algorithm like the meshing routine for example. Meshing might introduce new errors not related to the original model description. Furthermore the number of entities in the geometric model is smaller than after the discretization step.One disadvantage of this method is related to the complexity of the model. Usually the original model description is far more complex than the discretized model after meshing. Healing algorithms and routines have to handle complex geometric operations like for example intersections of trimmed NURBS surfaces. If the complexity of the original geometric model is rather low this concept is for sure the most efficient approach.
The majority of approaches published in the scope of scientific research and also implemented in commercial software are related directly to geometric models. An important motivation for this approach is that the model might be used for different purposes throughout the workflow and therefore a correct model is desirable. A collection of healing routines on CAD surfaces is presented in [33]. The routines are limited to surface descriptions since a BREP model is used as underlying geometric modelling scheme. The presented tools work directly on the geometric model obtained from neutral file formats like IGES and STEP. The model defects resulting from exchange of geometric information via neutral file formats is in the focus of this publication. The encountered problems of data exchange correlate to the experiences obtained during implementation of healing routines within the scope of this thesis. The user contributes substantially to the toolbox of healing routines described in [33]. Therefore an interactive and efficient user interaction is proposed.

Post-processing healing routines

In this concept the healing routines perform on the resulting mesh (discretized model) after meshing routines were applied on the unhealed original geometric model. Advantageous is the fact that a discretized model is usually represented by less complex shapes. In classical FEM analysis only linear shapes are used to approximate the spatial objects. Even in case of high order meshes the degree is usually lower than in the original model for complex shapes. The reduced model complexity results in easier and more robust geometric algorithms within the healing workflow. Intersection algorithms for example simplify dramatically if trimmed elements are approximated by lower dimensional untrimmed objects. Another advantage is that the healer can also address defects introduced by the meshing algorithm.
One disadvantage is that the number of errors and inconsistencies might even increase after meshing due to the meshing algorithm itself and the information about the true model character might be even harder to reconstruct for a healing routine. Furthermore this system can only

work if the mesher is able at all to mesh the original model. Therefore the requirements towards the meshing system concerning stability and robustness are higher. A huge drawback of that method is the fact that rather simple topological defects can influence the mesh dramatically and cause a huge healing effort for the discretized geometry. For example inconsistencies due to partially non-disjoint edges in the original model which have to be healed on basis of a mesh might be a big challenge for this configuration. The non-consistent intersection is detected easily on basis of a mesh but manipulation might be difficult since the mesh has to be modified and reconnected. Furthermore it might be disadvantageous that the original model remains defective. If somewhere in the downstream applications the original model has to be used the information about defects might not be included. In general it is complicated to transfer information about healed errors on the mesh to information on defects in the original model.
Since the number of elements usually increases significantly during meshing the healing routines on a mesh might have to be more efficient than the one on the original model, especially concerning mutual entity checks. [15] presented such a healing technique based on a meshed geometric model addressing the following defects:

- Gaps and overlaps
- T-joints
- Simple holes or missing surfaces
- Slivers

The routines make use of the fact that these defects can rather easily be detected on base of the mesh. For a mesh on solid models for example holes or gaps can be detected by free mesh edges. Small gaps can be closed by merging neighbouring edges and nodes. Beyond a defined tolerance holes have to be filled up by creating additional mesh elements. [15] proposes an algorithm for mesh modification to close small gaps and a tolerance above which a hole is filled up with new elements. Desired holes have to be marked within the data structure to apply this concept since the algorithm would try to close them otherwise.
Detection of sliver surface which are usually long and slender can be done by analysing the element quality within the mesh. It is assumed that sliver faces result in a poor mesh quality. The publication by [15] also presents an algorithm to close larger gaps in the structure. Nevertheless this approach is limited to models where the structure of the objects is able to be reconstructed even on basis of the defective model. Within this thesis routines will be presented to heal topologically missing faces in a BREP volume. It will turn out that this step might be rather easy to perform if the geometric information is complete but topologically defective. For a meshed structure on this kind of defective model however reconstruction might not succeed at all since model information is already significantly lost. The presented algorithms in [15] therefore might not be able to reconstruct the desired geometry.

Interactive healing routines

The approach to interactively integrate healing routines in the workflow can be seen as a hybrid version of the two systems described before. The mesher and healer perform interactively on the model and its approximated discretized form. By that the advantages of both approaches can be exploited. The sequence of healing and meshing routines can be of great importance. In a first step simple and basic defects have to be healed such that the geometric model is meshable or partially meshable. Complex problems can be detected on the discretized model but healed in the original one. As an example the intersection of two trimmed surfaces can be determined on basis of the mesh on these two surfaces and in a next step the intersection information is transferred to heal the original model.

Advantageous is the fact that the original model is healed and additionally a post-processing healing step can guarantee that defects introduced by the mesher are addressed. However the implementation effort increases since the stability of the entire configuration might depend strongly on the sequence of routines.
In case geometric models are discretized with elements of higher order (high order meshes) pre- and post-processing approaches might not differ so significantly anymore. Therefore the focus must be on how geometric and topologic relations and possible defects can be detected efficiently, in particular taking into account the state of the art parametric representation schemes. Topologic relations must be exploited and analysed in geometric models to obtain an insight on the model characteristics.

Healing routines and user interaction

We can conclude from all of the three presented approaches that user interaction might be an important factor to overcome ambiguous situations in the healing workflow. The importance of user interaction on the healing robustness is mentioned in [33]. A similar statement towards user interaction can be found in [31]. The healing framework presented in this publication is based on stl files. Models in the stl format are rather simple with respect to the geometric representation since the surfaces are tessellated. In that sense these models are similar to meshes on geometric models. The main conclusion of [31] is to built-up a powerful user interaction environment to tackle healing aspects in geometric models. Although rather simple the number of errors in geometric models based on the stl format with a high decomposition of the surfaces motivates an efficient user interaction.
Until now no proposed concept for healing routines is able to overcome the ambiguous character of geometric modelling uncertainties and therefore perform without any user interaction during the healing workflow. It might be even doubted if a fully automatic healing code exists at all.

3.2 Theoretical background on selected healing routines

After a brief introduction into defects of models and different approaches of healing implementation the following sections will provide the reader with some theoretical background on the several healing operations. The purpose is to make the practical implementation of healing routines and outlook for further development of healing routines more understandable. The distinction of topology and geometry in the modelling scheme will be used to classify the healing operations and algorithms. Therefore this section is separated into two parts for geometric and topologic algorithms.

3.2.1 Theoretical review on geometric healing

Geometric healing refers to the attempt to reveal insight on the models characteristics by its geometric shape. Furthermore geometric healing tries to detect model defects related to geometric shape properties. For example the existence of a surface bounded by a sequence of edges where all endpoints of the edges are distinct objects provides no insight on the geometric character of the surface and is purely topologic information. If none of the distinct point is also disjoint the surface results in a zero span and it will be geometrically meaningless. This little example indicates that we need geometric information within healing routines and have to analyse the geometric shape to detect model defects.
The basis for geometric operations is a metric space, which is in our case given by the Euclidean 3-space $\mathbb{R}^3$. Furthermore we need methods to measure lengths, areas and volumes, as well as distances between geometric objects in the model. The computation of most of those properties is subject of basic geometry courses. However applied on parametric shape descriptions things

might not be so trivial anymore. Additionally the algorithms have to be numerically stable since they might be confronted with inaccuracies resulting from round-off problems. Intersection problems are of significant importance for many geometric healing routines. In particular in case of intersections between parametric surfaces or even trimmed parametric surfaces the complexity of the problem increases dramatically. Therefore the following sections give a short overview on intersection aspects.

Intersection problems

Intersection problems can clearly be classified according to the dimension of the entities to be tested against each other. Therefore we have to address the following types:

- Point / Point
- Point / Curve
- Point / Surface
- Curve / Curve
- Curve / Surface
- Surface / Surface

The solutions of the intersection pairs listed above are the following:

Intersection pair	Solution set
Point / Point	{Empty, Point}
Point / Curve	{Empty, Point}
Point / Surface	{Empty, Point}
Curve / Curve	{Empty, Point, Curve, Curve segments}
Curve / Surface	{Empty, Point, Curve, Curve segments}
Surface / Surface	{Empty, Point, Curve, Curve segments, Surface patches}

Another classification of intersection problems is based on the mathematical representation of the geometric shapes. The representations scheme has significant influence on the complexity of the problem. Often the classification is structured like the following:

- IA: Implicit algebraic representation $f(x, y, z) = f(\boldsymbol{r}) = 0$
- IP: Implicit procedural, shapes derived from implicit expressions by offsetting, blending, sweeping etc.
- RPP: Rational polynomial parametric
  $r(u, v) = [X(u, v)/W(u, v),\ Y(u, v)/W(u, v),\ Z(u, v)/W(u, v)]^T$
- PP: Parametric procedural, shapes from parametric expressions obtained by offsetting, ...

For the rational polynomial parametric representation the concept of homogeneous coordinates is used, which was already explained in section 2.2.7. Number systems also play an important role for intersection problems since they influence the achievable accuracy. Another classification might therefore be done on basis of number systems.
A detailed description of algorithms for the intersection pairs with algorithms for different shape representations can be found in [32]. In the following only a very general overview on selected pairs is presented to illustrate the difficulties related to intersection problems.

Point/Point intersections The solution of point-point intersections is either a point or empty. The problem is easy to formulate and solve. A distance function D for the Euclidean space is needed to compute the distance between the points $r_0(x_0, y_0, z_0)$ and $r_1(x_1, y_1, z_1)$. Analytically it holds:

$$D(\boldsymbol{r}_0, \boldsymbol{r}_1) = 0 \tag{3.1}$$

Since we deal with floating point systems, numerically stable results require the reformulation in such a way:

$$D(\boldsymbol{r}_0, \boldsymbol{r}_1) \leq \epsilon \tag{3.2}$$

Point/Curve intersections The solution of this intersection pair is either empty or a point. In case the curve shape is given by an implicit algebraic representation (IA) the problem is rather simple. The point-coordinates $\boldsymbol{r} = [x_0, y_0, z_0]^T$ can be inserted directly into the curve's function $g : f(x, y, z) = 0$:

$$f(\boldsymbol{r}) \stackrel{!}{=} 0 \tag{3.3}$$

Due to floating point errors it is however necessary to solve:

$$|f(\boldsymbol{r})| \ \leq \ \epsilon \tag{3.4}$$

Things become more complicated in case the curve shape is given in a parametric representation $g(t) = [X(t)/W(t),\ Y(t)/W(t),\ Z(t)/W(t)]^T$. For a given point $\boldsymbol{r} = [x_0, y_0, z_0]^T$ we have to solve the equations:

$$\begin{aligned} X(t) - x_0 &= 0 \\ Y(t) - y_0 &= 0 \\ Z(t) - z_0 &= 0 \end{aligned} \tag{3.5}$$

The equations can be solved using the Newton method for example. However the practical implementation of the Newton method might be inefficient for intersection problems.
Another possibility is to perform a so-called **subdivision** of the problem. We can make use of the convex hull property of parametric curves described in section 2.2.7. The bounding box (BBox) defined by the control point set $\{x_i, y_i, z_i\}$ contains the curve:

$$\begin{aligned} min(x_i) \ \leq \ x(t) \ \leq \ max(x_i) \\ min(y_i) \ \leq \ y(t) \ \leq \ max(y_i) \\ min(z_i) \ \leq \ z(t) \ \leq \ max(z_i) \end{aligned} \tag{3.6}$$

The convex hull property allows us to reduce the "search domain" on the curve which has to be tested against the point. If we obtained a small enough search interval I we can formulate a minimization problem:

$$F(t) = min\{|\boldsymbol{r} - \boldsymbol{r}(t)|^2,\ t \in I\} \tag{3.7}$$

Starting from an initial $t_s \in I$ we can apply numerical techniques to converge to the minimum t_0. If $\sqrt{F(t_0)} \leq \epsilon$ we have found an intersection point.
Another efficient method is to formulate a distance function $D(t)$ and check its derivative $\dot{D}(t)$ for stationary points $t_{s,i}$. If the distance $D(t_{s,i}) \leq \epsilon$ we have found an intersection point.

From the previous derivations it became apparent that the intersection problems are much easier to formulate on basis of an implicit representation. Therefore a very convenient method for parametric shapes is the so-called **implicitation**. The goal is to reformulate the problem in an implicit manner.
For two polynomials $p : p(t)$ and $q : q(t)$ with:

$$p(t) = a_m t^m + a_{m-1} t^{m-1} + \cdots + a_1 t + a_0 = x \tag{3.8}$$

which possesses the roots $\{\alpha_1, \cdots\}$; and the second polynomial:

$$q(t) = b_n t^n + b_{n-1} t^{n-1} + \cdots + b_1 t + b_0 = y \tag{3.9}$$

with the roots $\{\beta_1, \cdots\}$ we can reformulate the equations like:

$$\begin{aligned} a_m t^m + a_{m-1} t^{m-1} + \cdots + a_1 t + a_0 - x = 0 \\ b_n t^n + b_{n-1} t^{n-1} + \cdots + b_1 t + b_0 - y = 0 \end{aligned} \tag{3.10}$$

This form allows us to set-up an equation system in matrix form based on the so-called Sylvester matrix:

$$Syl(p,q) = \begin{bmatrix} a_0 - x & a_1 & \cdots & a_m & 0 & \cdots & 0 \\ 0 & a_0 - x & a_1 & \cdots & a_m & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & a_0 - x & a_1 & \cdots & a_m \\ b_0 - y & b_1 & \cdots & b_n & 0 & \cdots & 0 \\ 0 & b_0 - y & b_1 & \cdots & b_n & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & \cdots & 0 & b_0 - y & b_1 & \cdots & b_n \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ t^{m+n-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \cdots \\ \cdots \\ \cdots \\ \cdots \\ 0 \end{bmatrix} \tag{3.11}$$

The resultants of the polynomials $\rho(p,q)$ are equal to the determinant of the Sylvester matrix. Therefore we obtain a function of x and y, in this example: $\rho(p,q) = \rho(x,y)$. The vanishing of the resultants $\rho(x,y) \stackrel{!}{=} 0$ is a necessary condition for the polynomials to have common roots. This means that x, y values satisfying $\rho(x,y) = 0$ lie on the parametric curve and therefore $\rho(x,y) = 0$ is the implicit form of the parametric curve representation. Once the implicit form is given we can solve $\rho(x_0, y_0) = 0$ for a point $\boldsymbol{r} = [x_0, y_0]$ to verify if the point is on the curve. If this is the case in an inversion step we have to determine the corresponding parameter value t related to $[x_0, y_0]$.
This implicitation method is very powerful especially for low degree polynomials, but might suffer from floating point problems.
In case the curve description is given in form of a procedural representation the solution of the intersection problem is more difficult. There is no general procedure to reduce the search interval with help of a BBox and subdivision techniques. The minimization problem $F(t) = |\boldsymbol{r}(t) - \boldsymbol{r}_0|^2$ has to be solved over the entire domain $[0, 1]$.

Point/Surface intersections The point-surface intersection problem is very similar to the point-curve problem with the difference that the dimension of one member is increased. Basically the same solution techniques are available than for the point-curve pairs.

Curve/ Curve intersections The complexity of this intersection pair depends strongly on the representation of the curves.

In case the intersection pair is IA/RPP, denoting that one curve is given in algebraic form and one curve in parametric form, we end up in a set of equations. The algebraic curve is defined by the equations:

$$\begin{aligned} f(x,y,z) &= 0 \\ g(x,y,z) &= 0 \end{aligned} \tag{3.12}$$

and the parametric curve by:

$$\begin{aligned} x &= \frac{X(t)}{W(t)} \\ y &= \frac{Y(t)}{W(t)} \\ z &= \frac{Z(t)}{W(t)} \end{aligned} \tag{3.13}$$

The problem formulation reads:

$$\boldsymbol{r} = r(t) \cap \ f(\boldsymbol{r}) = g(\boldsymbol{r}) = 0 \tag{3.14}$$

By substituting $x = \frac{X(t)}{W(t)}$, $y = \frac{Y(t)}{W(t)}$ and $z = \frac{Z(t)}{W(t)}$ into f and g and multiplying by $W^m(t)$, where m is the degree of the algebraic functions, we end up with polynomial equations. Resultants can be used to compute the common roots.
In case the pair reads RPP/RPP, indicating both curves are given in parametric form, like $r_1(t)$ and $r_2(\sigma)$, we end up with 3 nonlinear polynomial equations from the equality $r_1(t) = r_2(\sigma)$. These equations have to be solved for the two unknowns t and σ. There are different methods to solve these equations, like for example bounding box and subdivision methods followed by a minimization problem.

Surface/Surface intersections Surface - surface intersection pairs are by far the most complex problems. However in many fields this type of intersection appears frequently like for example in computer graphics or modelling. Accordingly there exists a huge body of literature on this topic. The solution of surface / surface intersections is either empty, consists of points, intersection curves or consists of surface patches.
Intersection curves for two surfaces can be grouped in two different types illustrated in picture 3.1. Open components are intersection curves arising from intersections of surface boundaries. In contrast also so-called inner components are possible forming intersection loops or inner lines.

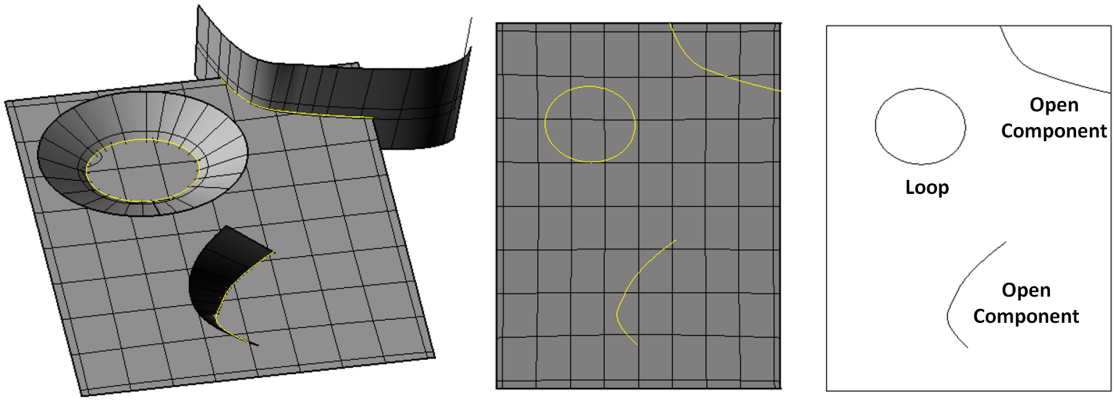


Figure 3.1: Intersection curve components from S/S intersection pairs

These intersection curves can degenerate to singular intersection points which are in general hard to detect.
There are several general approaches to address the surface intersection problem given in table 3.2. Some concepts were already illustrated for other intersection pairs.

Name	Basic Idea
Triangulation	Intersection is performed on approximated geometry. The geometry is approximated via triangulation.
Lattice methods	Dimensionally reduced problem formulation. Performing C/C intersections of a net of iso-parametric curves on one surface with the curves on the other surface. The discrete intersection points on the set of curves have to be resembled to obtain the final solution.
Subdivision	Decomposition of the original problem into simpler problems. Decomposition is done until problem is easy enough to obtain direct solutions for the intersection problem.
Marching methods	Starting from one point on each intersection component (loop, open component) tracing methods allow us to define the intersection path by local differential geometry properties.

Table 3.2: S/S Intersection approaches

Every method has its intrinsic advantages and disadvantages which are outlined shortly here:

Name	Advantages and drawbacks
Triangulation	The intersection problem is rather easy to formulate and solve on a linear triangulated shape. The accuracy of the results depends significantly on the approximation (triangulation). Therefore a robust triangulation algorithm is needed and the resolution has to be adjusted to the shape characteristics.
Lattice methods	The problem is much easier to be solved on the dimensionally reduced system. The initial grid resolution for search curves influences the results significantly. Small loops or singular intersection components might be missed.
Subdivision	Subdivisions can restrict the search domain significantly and the problem formulation might be much easier on the subdivided domain. In the limit case this method is convergent. Especially in combination with other methods this procedure can increase efficiency dramatically. High accuracy requirements of the results require high levels if subdivision and might lead to data proliferation. Reversely if a low level of subdivision is chosen the results might be inaccurate.
Marching methods	Marching methods make use of local differential geometry and analytic formulations. Therefore the accuracy is usually rather high for these methods. However the procedure is not complete since a start point for every component of the intersection is needed. Another problem is that the step size for tracing methods following the intersection path along a component might be significant for the accuracy. Especially if components are close the tracing algorithm might jump from one component to another and fail (see picture 3.2).

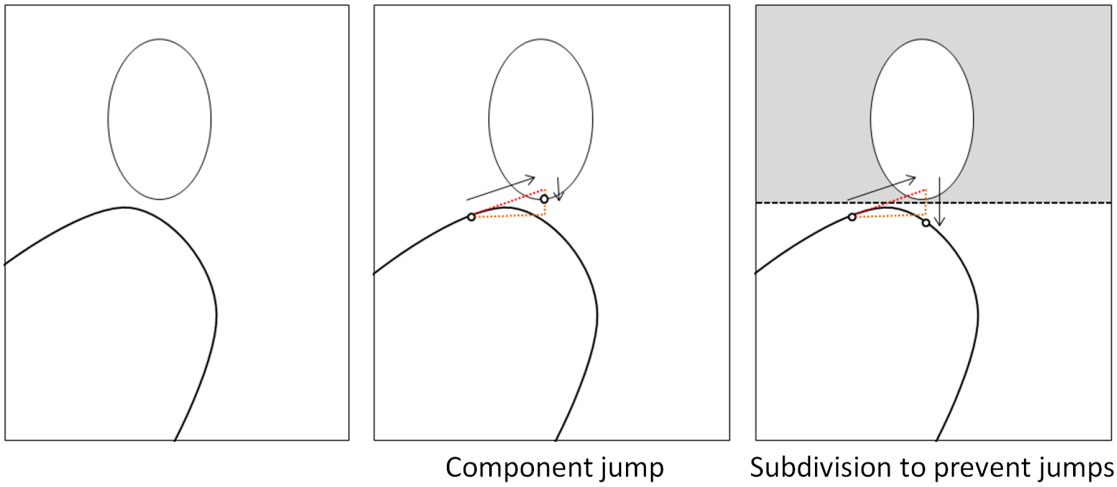


Figure 3.2: Marching methods: component jumping and subdivision remedy

The problem within marching method to adjust the step size for tracing to prevent jumping into

another component's path motivates the combination of different procedures. In [27] a algorithm is presented which combines marching methods, subdivision and algebraic formulation. Basic idea of the algorithm in [27] is to implicitize the problem. Consider two rational polynomial surfaces:

$$F(u,v) = [\frac{X(u,v)}{W(u,v)}, \frac{Y(u,v)}{W(u,v)}, \frac{Z(u,v)}{W(u,v)}]^T$$
$$G(u,v) = [\frac{\bar{X}(u,v)}{W(u,v)}, \frac{\bar{Y}(u,v)}{W(u,v)}, \frac{\bar{Z}(u,v)}{W(u,v)}]^T \tag{3.15}$$

In a first step $F(u,v)$ is transformed to its implicit form $f(x,y,z,w) = 0$. In a next step the parameterization of G is substituted in F leading to:

$$f(\bar{X}(U,V), \bar{Y}(U,V, \bar{Z}(U,V) = 0 \tag{3.16}$$

This implicit formulation can be used to trace the curve. The method needs a start point on every component to perform tracing. Detecting a start point for open components is equivalent to a curve-surface intersection problem. Far more complicated is the detection of loops. For that purpose the algorithm makes use of the loop detection routines described in [24]. The intersection algorithm makes use of subdivision to guarantee a stable tracing without component jumping. For a detailed description the user is referred to the reference literature.

Closing remarks on intersection problems The previous sections give only a very short and general overview over the aspects and approaches to address intersection problems. The implementation into the healing routines is scope of further development as described in the outlook in section 7.2. It turned out that at this point the provided geometric functions from the external library Open CASCADE will have to be extended and embedded in an efficient algorithm to tackle the problem. This is especially the case for trimmed NURBS surface-surface intersection.

3.2.2 Theoretical review on topologic healing

In contrast to geometric defects we can observe that several errors of a model do not depend on the geometric shape at all but on the way objects are related or put together. This type of defects is covered within the field of topology healing. In a first step the concept of topology is illustrated roughly before the applications within healing routines is described.

Topology and topologic spaces

Topology is the study of continuity and connectivity and investigates the qualitative properties of objects. Topologic characteristics are preserved under continuous deformation of objects, like stretching or rotation. Topology is based on a topologic space which allows us to formulate the concepts of convergence, connectedness and continuity.
A topological space is based on a set X together with a collection of subsets of X denoted as τ.τ is called the topology on X, whereas elements of X are usually referred to as points or vertices. The following axioms have to hold:

- The empty set $\{0\}$ and X are contained in τ
- τ is closed under arbitrary union
- τ is closed under finite intersection

Many theorems in the field of topology hold under the assumption that the topologic space is equipped with a separation axiom T. There are many separation axioms. The most frequently used axiom is the so-called Hausdorff separation axiom, transforming a topologic space into a Hausdorff space. This space guarantees that distinct points have disjoint neighbourhoods. Two points x and y of the topologic space are disjoint if there is a neighbourhood U of x and V of y such that U and V are disjoint:

$$(U \cap V = \emptyset) \tag{3.17}$$

Examples for Hausdorff spaces are:

- Open intervals on the set of real numbers $\mathbb{R}$.
- Open balls on the Euclidean space $\mathbb{R}^n$.
- Manifolds since they inherit locally the topologic properties of the Euclidean space.
- Linear graphs have a natural topology.

Topologic spaces - Manifolds Manifolds play an important role in the field of geometry and topology. Manifolds are topological spaces "which on a small enough scale resemble the Euclidean space of a specific dimension" (Wikipedia: manifolds). In [30] we find an illustrative definition for two-dimensional manifolds. A 2-D manifold is a mathematical abstraction of a surface made of a sheet of paper with the same local properties than the plane of Euclidean space. "An intelligent bug crawling on a surface could not distinguish it from a plane if he had a limited range of visibility." A similar statement holds for higher dimensional manifolds. Low-dimensional topology is a branch of topology dealing with four or fewer dimensional manifolds since they are of special interest in various mathematical fields.
More formally a definition for n-manifolds is also provided by [30] as follows: For an positive integer n a n-dimensional manifold is a Hausdorff space where each point has an open neighbourhood homeomorphic to the open n-dimensional disc $U^n = \{x \in R^n \ : \ |x| < 1\}$.
Connected n-manifolds can be grouped into two kinds: orientable and non-orientable. If an orientation is defined for example by the orientation of the coordinates system then an orientable manifold preserves this information while moving along its domain. In other words every closed path on the manifold is orientation preserving. The well known Möbius strip, see picture 3.3, is an example for non-orientable manifolds. The orientation information is not preserved in case we travel along its circle connecting the two ends of the strip.

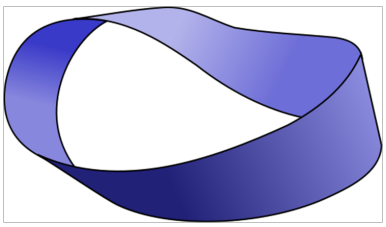


Figure 3.3: Non-orientable manifold: Möbius strip

Topologic spaces - connected graphs Another very important topological space is given by linear graphs also called path graphs. As illustrated in section 2.2.1 a graph in general is a set of so-called vertices and a collection of edges that connect pairs of vertices. Graphs can be undirected or directed, depending on the fact if the order is relevant for the vertex pairs. Graphs are often visualized in a way illustrated in figure 3.4.

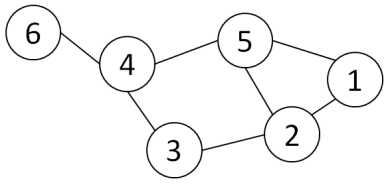


Figure 3.4: General graph example

A linear graph or path graph is a special graph where no branches are present in the graph structure. The degree of a node or vertex is given by its connections meaning its edges bounded by that particular vertex. In a linear graph all vertices have a degree of one or two, so only one or two edges are connected to that vertex as in the graph in picture 3.5.

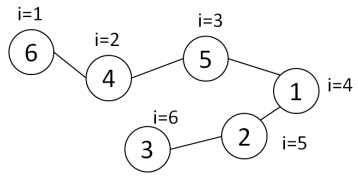


Figure 3.5: Linear graph example

A path is a sequence of vertices. In case a path is finite there is always a start vertex and an end vertex. A cycle is described by a path where start and end vertex are the same and the path degenerates into an infinite path. Nevertheless the definition of a start and end vertex is still valid in an infinite path although in a cycle the definition of these particular vertices is arbitrary, see picture 3.6.

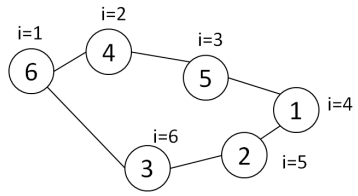


Figure 3.6: Infinite linear graph example

If the number of vertices and edges is finite we talk of a finite graph. Furthermore the absence of multiple edges or graph loops is resulting in a sub-graph of a complete graph. A complete graph is a graph where all nodes are interconnected by an edge.

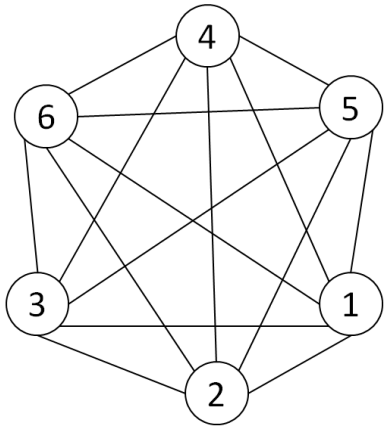


Figure 3.7: Complete graph example

Basic concepts of topology

A detailed review on topologic properties can be found in [30] and [17]. This section wants to mention only some very basic concepts which will have a relation to topologic healing routines. A basic theory which for example differentiates geometric and topologic viewpoints is the study of homotopy. Tow continuous functions from one topological space to another are said to be homotopic if one function can be continuously deformed into the other. More formally we define a homotopy between two continuous functions f and g from a topological space X to a topological space Y by a continuous function $H: X \times [0,1] \to Y$ such that:

$$\text{If } x \in X: \; H(x,0) = f(x) \text{ and } H(x,1) = g(x) \tag{3.18}$$

The parameter H can be seen as a time. At time $H = 0$ we have the function f which will be transformed into the function g at time $H = 1$. The theory of homotopy is used to define homotopy groups which are used to classify topological spaces. There is a special group, the fundamental group, within homotopy groups revealing information about loops in a space.
Another central aspect of topology is the concept of homeomorphism. A Homeomorphism is a continuous function between topological spaces with a continuous inverse function. A space is called homeomorphic to another if both are the same from a topologic viewpoint. A mapping between them preserves all topologic properties. Homeomorphism is the specification of the general concepts of abstract algebra, namely isomorphism and homomorphism, for topologic spaces. Homomorphism describes a mapping of one structure into another that preserves the algebraic structure. If this holds in a bijective sense we speak of isomorphism. Isomorphism is exploited in graph theory. An isomorphism of graphs G and H is a bijective mapping of the vertex set G and H given by $F: V(G) \to V(H)$ such that any two vertices u and v of G are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in H. We also speak of an edge preserving bijection.
Abstract algebra can be applied to analyse topologic spaces. We then talk of algebraic topology. One of the main aspects of this field of mathematics is to find groups of topologic spaces where statements about structure of the space are easier to define. An important theory exploited in algebraic topology is the theory of homology. Homology theory motivates the definition of homology groups which are in general much easier to compute than homotopy groups. The concept to describe shapes by their holes or translated to topologic spaces, describe spaces by means of cycles might be an intuitive definition of homology theory.
A formal derivation of homology groups is found in [30]. The theory of homology groups is only roughly presented to the reader at this point.

Let $\mathbb{R}$ be a real line, I a closed interval $[0,1]$ and $\mathbb{R}^n = \mathbb{R} \times \mathbb{R} \cdots \ (n - factors)$ the Euclidean n-space. Then a unit n-cube is given by:

$$I^n = I \times I \cdots \ (n - factors) \tag{3.19}$$

I^0 consists of a single point. A singular n-cube is a map $T: \ I^n \to X \ , (n \geq 0)$. $Q_n(X)$ denotes the free abelian group generated by the set of all singular n-cubes in X. Any element in X is given by a finite linear combination of n-cubes in X.
A n-cube $T: \ I^n \to X$ is degenerate if there exists an integer i with $1 \leq i \leq n$ such that $T(x_1, x_2, \cdots, x_n)$ does not depend on x_i (the function is independent of one of the coordinate values). If $D_n(X)$ is the subgroup of $Q_n(X)$ given by degenerated cubes then the quotient group $C_n(X) = \frac{Q_n(X)}{D_n(X)}$ is referred to as n-chains in X.
For a n-cube front and back faces $A_iT, B_iT: \ I^{n-1} \to X$ can be defined. The concept of faces is left to the reader's intuition, especially when considering 2-cubes. Faces enable us to define a boundary operator ∂, given by the homomorphism $\partial_n: \ Q_n(X) \to Q_{n-1}(X), \ n \leq 1$ by:

$$\partial_n(T) \ = \ \sum_{i=1}^{n} (-1)^i [A_iT - B_iT] \tag{3.20}$$

In some sense this operator represents the oriented boundary of an n-cube. Two important properties are related to the boundary operator:

$$\begin{aligned} &(i) \quad \partial_{n-1}(\partial_n(T)) = 0 \quad , (n > 1) \\ &(ii) \quad \partial_n(D_n(X)) \subset D_{n-1}(X) \quad , (n > 0) \end{aligned} \tag{3.21}$$

From that follows that there exists a induced homomorphism: $C_n(X) \to C_{n-1}(X)$ also denoted by ∂. Furthermore we define:

$$\begin{aligned} Z_n(X) \quad &= \quad kernel(\partial_n) \ = \ \{u \in C_n(X) \mid \partial(u) = 0\} \quad , \ n > 0 \\ B_n(X) \quad &= \quad image(\partial_{n+1}) \ = \ \partial_{n+1}(C_{n+1}(X)) \quad , \ n > 0 \end{aligned} \tag{3.22}$$

Since it holds that $B_n \subseteq Z_n$ we can define the quotient set:

$$H_n(X) \ = \ \frac{Z_n(X)}{B_n(X)} \tag{3.23}$$

$H_n(X)$ is called the n-dimensional homology group of X. $Z_n(X)$ is also called the group of n-dimensional singular cycles of X (group of n-cycles) and $B_n(X)$ the group of n-dimensional boundaries or group of bounding cycles. It is necessary to define $H_0(X)$. Since $Z_0(X) = C_0(X)$ the definition is given by:

$$H_0(X) \ = \ \frac{C_0(X)}{B_0(X)} \tag{3.24}$$

The reason why the theorem on homology and homology groups was described more precisely is that this concept has a quite useful application for finite graphs and can be exploited within healing routines.

Reminder on some basic Linear Algebra

In the following we want to investigate the structure of homology groups. To make use of homology theory the user is shortly reminded in basic concepts of Linear Algebra. In particular

the reader is reminded about the four fundamental subspaces of a $m \times n$ Matrix A:

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ & & \cdots & \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \tag{3.25}$$

which are the following:

Subspaces of $\mathbb{R}^n$
- Nullspace of A: $Null(A) = \{x \in \mathbb{R}^n : Ax = \{0\}\}$
- Rowspace of A: $R(A^T) = \{x \in \mathbb{R}^n : x = A^T y \text{ for some y}\}$
- all linear combinations of the rows of A

Subspaces of $\mathbb{R}^m$
- Left Nullspace of A: $Null(A^T) = \{y \in \mathbb{R}^m : A^T y = \{0\}\}$
- Column Space of A : $R(A) = \{y \in \mathbb{R}^m : y = Ax \text{ for some x}\}$
- all linear combinations of the columns of A

The relation of the Nullspace of a matrix A and the Left Nullspace of this matrix is of special importance for the following. However often we are not directly interested in the Nullspace of a matrix itself but in a basis for the Nullspace. The two following examples are illustrating the meaning of the bases for a Nullspace.
For a matrix A:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \end{bmatrix} \tag{3.26}$$

the Nullspace is given by:

$$Null(A) = c \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \tag{3.27}$$

In comparison we take the matrix $\hat{A}$

$$\hat{A} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \end{bmatrix} \tag{3.28}$$

The Nullspace is then given by:

$$Null(\hat{A}) = c_1 \begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix} + c_2 \begin{bmatrix} -3 \\ 0 \\ 1 \end{bmatrix} \tag{3.29}$$

We can see that the Nullspace of a matrix lives in $\mathbb{R}^n$, more precisely is a subspace of $\mathbb{R}^n$ spanned by the bases which are vectors $\{\boldsymbol{b}\} \in R^n$. The dimension of the Nullspace is determined by the number of base vectors to span the Nullspace. For the matrices above there is also an illustrative explanation for the Nullspace. For these 2x3 matrices the Nullspace describes the intersection of two planes trough $(0, 0, 0)$. In the first case all the solution lie on the line spanned by the vector $\{\boldsymbol{b}^T\} = [1, -2, 1]$. In the second case the intersection is a plane spanned by the bases given above.

There is an important relation of the Nullspace to the row space, given by the Rank-Nullity theorem. If the dimension of the Nullspace is given by $nullity(A)$, and n is the number of columns of a matrix, then the following holds:

$$rank(A) + nullity(A) \;=\; n \tag{3.30}$$

The computation of the Nullspace can be performed by using elementary row operations. Row operations of this kind do not change the Nullspace of the matrix. Therefore the matrix is transferred into its Row-Echelon form to obtain the free variables and form a basis for the Nullspace.
This method however is not well-suited for numerical computations since it might suffer from round-off errors and numerical instabilities. A state-of the art approach is based on singular value decomposition. The singular value factorization of a matrix reads:

$$\underset{(m\times n)}{A} \;=\; \underset{(m\times m)}{U}\;\underset{(m\times n)}{\Sigma}\;\underset{(n\times n)}{V^T} \tag{3.31}$$

The matrix Σ holds the singular values in the following form:

$$\Sigma \;=\; \begin{bmatrix} \sigma_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & \sigma_2 & 0 & \cdots & 0 & 0 \\ & & & \cdots & & \\ 0 & 0 & 0 & \cdots & \sigma_r & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{3.32}$$

with $\sigma_1 \leq \sigma_2 \leq \cdots \sigma_r$. The rank r of the matrix can be found by detecting the non-zero entries on the main diagonal in a numerical sense. The matrix U holds the eigenvectors of AA^T in its columns, whereas the matrix V holds the eigenvectors of A^TA in its columns. The rank-nullity theorem can be used to finally find the Nullspace of the matrix. The following relation holds:

$$\underset{(m\times m)}{U}\;\underset{(m\times n)}{\Sigma}\;\underset{(n\times n)}{V^T}$$

$$\begin{bmatrix} & & & & & \\ [u_1] & [\cdots] & [u_r] & [u_{r+1}] & [\cdots] & [u_m] \\ & & & & & \end{bmatrix} \begin{bmatrix} \sigma_1 & & & & \\ & \cdots & & & \\ & & \sigma_r & & \\ & & & 0 & \\ & & & & 0 \end{bmatrix} \begin{bmatrix} [& v_1^T &] \\ [& \cdots &] \\ [& v_r^T &] \\ [& \cdots &] \\ [& v_n^T &] \end{bmatrix} \begin{matrix} \} Row(A) \\ \\ \\ \\ \} Null(A) \end{matrix}$$

$$\overbrace{Col(A)} \qquad \overbrace{Null(A^T)}$$

(3.33)

So by performing a singular value decomposition we obtain the Nullspace and Left Nullspace of a matrix at once. Moreover the rank of the matrix can be computed from the matrix Σ. The singular value decomposition is well-suited for numerical algorithms and integrated in the Linear Algebra library Lapack [3] for example.

Homology of Finite Graphs

Equipped with the concept of fundamental matrix subspaces and the homology theory we can turn towards the practical application of homology groups for finite graphs.
Consider a finite, regular graph consisting of the pair X and X^0, where X is a Hausdorff space and X^0 a finite subspace denoting the vertices in X. The following conditions are defined:

- $X - X^0$ is a disjoint union of a finite number of open subsets $\{e_1, e_2, \cdots, e_k\}$ called edges. Each e_i is homeomorphic to the open interval $(0, 1)$ of the real line.
- The point set boundary of an edge e_i is given by two distinct vertices obtained from $\overline{e_i} - e_i$, where $\overline{e_i}$ is homeomorphic to $[0, 1]$. The pair $(\overline{e_i}, e_i)$ is then homeomorphic to the pair $([0, 1], (0, 1))$.

The graph in this form might be connected or disconnected and might have isolated vertices. Our interest is to determine the homology groups of the graph X. A detailed description of the following derivations can be found in [30]. At this point only the rough outline of the procedure and results is presented.
In a first step the pair (X, X^0) is considered which allows us to determine the structure of so-called relative homology groups $H_n(X, X^0)$. It can be shown that knowledge of the structure of the groups $H_n(X, X^0)$ give rise to information about the homomorphism $i_* : H_n(X^0) \to H_n(X)$ and vice versa. Again the reader is referred to [30] for a detailed description. The purpose of this definition is to describe the so-called exact homology sequence of pairs, which is denoted in our case by the following:

$$0 \to H_1(X) \xrightarrow{j_*} H_1(X, X^0) \xrightarrow{\partial_*} H_0(X^0) \xrightarrow{i_*} H_0(X) \to 0 \tag{3.34}$$

Our final interest is to determine $H_1(X)$. To reveal this knowledge about the homology groups of X we have to know the structure of the other groups in the sequence above. It can be shown that the rank of the free abelian group $H_0(X)$ is equal to the number of vertices. Furthermore we make use of the Euler characteristics. For a graph the Euler characteristic reads:

$$\sharp \text{ vertices} \; - \; \sharp \text{ edges} \; = \; \text{Euler characteristic} \tag{3.35}$$

Starting from these theorems it is possible to derive the main theorem related to homology groups of a graph, which can then be stated as follows:
If (X, X^0) is a finite regular graph then $H_q(X) = 0$ for $q > 1$, $H_1(X)$ is a free abelian group and:

$$rank(H_0(X)) - rank(H_1(X)) \; = \; \text{Euler characteristics} \tag{3.36}$$

We can use this theorem to determine the structure of the graph. We know the rank of $H_0(X)$ simply by the number of vertices and we can compute the Euler characteristics for a graph if we know the number of edges. The homomorphism $\partial_* : H_1(X, X^0) \to H_0(X)$ is determined by the kernel and respectively the cokernel of this homomorphism. We have to express this homomorphism by choosing convenient bases for the groups $H_0(X^0)$ and $H_1(X, X^0)$. A basis of $H_0(X^0)$ consists of the set of vertices. Choosing bases for $H_1(X, X^0)$ is more complicated. By orienting the vertices $v_{i,\alpha}$ and $v_{i,\beta}$ for an edge e_1 we can define a basis for $H_1(X, X^0)$ and the following holds:

$$\partial_*(e_i) \; = \; v_{i,\beta} - v_{i,\alpha} \tag{3.37}$$

The homomorphism ∂_* is therefore defined and we are able to determine the kernel of this homomorphism. By doing so we will find out that cycles in the graph actually form a basis for

the kernel of ∂_* or what is equivalent for the homology group $H_1(X)$. In particular the set of linear independent cycles in the graph form the basis for the 1-dimensional homology group. The entire homology group is then given by linear combinations of the basic cycles.
We therefore have determined the structure of the graph by means of the homology groups. A more intuitive description for these results and relations can be found in [18]. The main statement anyhow remains the same. For a graph $G = (V, E)$ with n edges and m vertices. The following subspaces are defined:

$$\begin{aligned} V(G): &\quad \text{vertex space;} \quad dim(V(G)) = m \\ E(G): &\quad \text{edge space;} \quad dim(E(G)) = n \end{aligned} \tag{3.38}$$

The cycle space of a graph is spanned by all cycles in G, in particular by the edge set of all cycles in G.
If T is a spanning tree on the subgraph on G, i.e. a tree containing all vertices of the graph and a certain number of edges such that this subgraph T on G has no cycles, we introduce a fundamental cycle C_e by adding exactly one edge e to the graph as illustrated in picture 3.8.

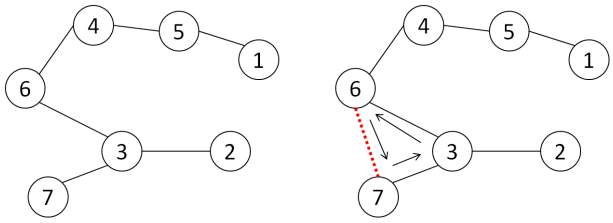


Figure 3.8: Spanning tree and fundamental cycles

These **fundamental cycles** are a basis for the cycle space on G.
Furthermore we can define a so-called **incidence matrix** (vertex-edge) of the graph. It is a matrix $B = (b_{i,j})_{m \times n}$ of a graph $G = (V, E)$ with the vertices $V = \{v_1, v_2, \cdots, v_m\}$ and edges $E = \{e_1, e_2, \cdots, e_n\}$ defined as follows:

$$b_{i,j} = \begin{cases} -1 & \text{if } v_i = v_\alpha \\ 1 & \text{if } v_i = v_\beta \\ 0 & \text{otherwise} \end{cases} \tag{3.39}$$

Where v_α is the start vertex of an oriented edge and v_β the end vertex of an edge e_j. This incidence matrix is in fact nothing else than the transpose matrix form of the homomorphism $\partial_*(e_j)$ defined above. With these definitions the fundamental result for finite graphs can be formulated:
The kernel of B is $C(G)$ and the fundamental cycles form a basis for $C(G)$.
Nothing else is stated by the theorem that basic cycles in the graph form a basis for the 1-dimensional homology group on the graph.

Applications of Homology Theory of Finite Graphs for Healing Routines

In a first step we will apply the theorems derived in the previous section to simple graph examples and then show the relation to healing routines. Let us consider the following graph G.

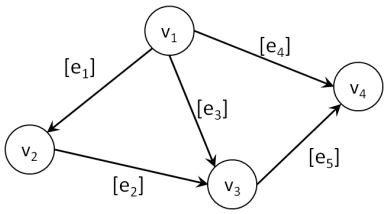


Figure 3.9: Simple graph example

For this graph we now set up the so-called edge-vertex incidence matrix $\hat{B}_{n\times m}$ with number of edges n and number of vertices m. Notice that this matrix is the transpose of the previously defined incidence matrix B. The matrix has the following structure:

$$\hat{B}(G) = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \tag{3.40}$$

The Nullspace of this matrix is given by:

$$Null(\hat{B}(G)) = c * \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{3.41}$$

The Nullspace of $\hat{B}$ lives in $\mathbb{R}^n = \mathbb{R}^4$, which means it is related to the vertices. So the dimension of the Nullspace of $\hat{B}$ is: $dim(Null(\hat{B})) = 1$. With the help of the rank-nullity theorem we can determine the rank of $\hat{B}$: $rank(\hat{B}) = 3$. This refers to the dimension of the column space of this matrix.
There is a convenient mechanical analogy to this result. If the graph would describe a pipe system with hydraulic potentials at the nodes, corresponding to the vertices in the graph, the results states that the potential at nodes can only be computed up to a constant c. If the potential at every node is equal to c no hydraulic flow will occur. Therefore this result refers to smth. like a balance equation. In contrast we set up a second graph G' like depicted in picture 3.10.

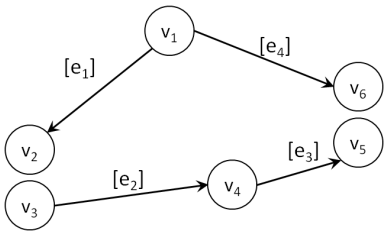


Figure 3.10: Simple graph example with two disconnected substructures

The edge-vertex incidence matrix $\hat{B}$ reads:

$$\hat{B}(G') = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.42}$$

Computing the Nullspace $Null(B(\hat{G'}))$ of this matrix we obtain the following result:

$$Null(\hat{B}(G')) = c_1 * \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} + c_2 * \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.43}$$

In terms of the previously introduced analogy this result indicates that there are two totally independent subgraphs where in each subgraph the potential at the nodes can be computed up to a constant.
We introduce a third Graph G'' as sketched in picture 3.11, where the graph is "open" on one side.

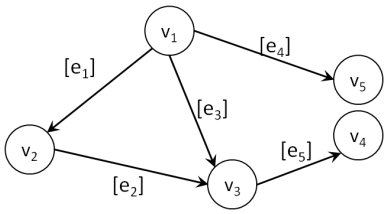


Figure 3.11: Simple graph example with one disconnected end

The edge-vertex incidence matrix $B(\hat{G''})$ and the Nullspace $Null(B(\hat{G''}))$ are given as follows. Again we can detect that all nodes contribute to a "balanced state".

$$\hat{B}(G'') = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 \end{bmatrix} \tag{3.44}$$

$$Null(\hat{B}(G'')) = c * \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{3.45}$$

In a next step we investigate the Left Nullspaces of the matrices, which is euivalent to computing the Nullspace of the transpose of the matrix. The transpose of the matrix is called vertex-edge incidence matrix and equals the originally defined matrix B.

For graph G the Nullspace of G: $Null(B)$ reads:

$$Null(B(G)) \;=\; c_1 * \begin{bmatrix} 1 \\ 1 \\ -1 \\ 0 \\ 0 \end{bmatrix} + c_2 * \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 1 \end{bmatrix} \tag{3.46}$$

The Nullspace is spanned by two bases. These bases are equal to fundamental cycles in the graph as illustrated in figure 3.12.

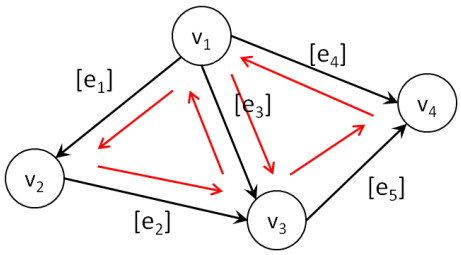


Figure 3.12: Fundamental cycles in the graph

From the Nullspace base vectors we can determine the pivot columns of the matrix B which are the columns 1, 2 and 4 in our case for example. These columns have an important meaning. The form a spanning tree T on the graph G illustrated in picture 3.13.

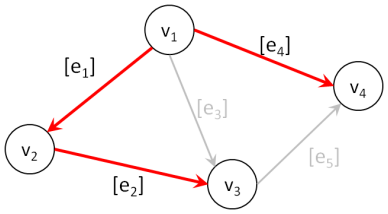


Figure 3.13: Spanning tree on the graph

Furthermore we can test and observe that the Euler formula holds for that graph:

$$\begin{aligned} \sharp \text{ loops} &= dim(Null(B)) \\ \sharp \text{ loops} &= \sharp \text{ edges} - (\sharp \text{ nodes} - 1) \end{aligned} \tag{3.47}$$

Again a mechanical analogy holds for these results. The cycles represent cyclic paths in the pipe network such that no potential is accumulated at the nodes.
Since the Nullspace of graph G' : $Null(B(G'))$ has dimension zero it means that there are no cycles present in the graph G'.
The Nullspace of the graph graph G'' : $Null(B(G''))$ has dimension 1. Visually we can easily

detect on cycle given by the only Nullspace vector:

$$Null(B(G'')) \;=\; c_1 * \begin{bmatrix} 1 \\ 1 \\ -1 \\ 0 \\ 0 \end{bmatrix} \tag{3.48}$$

Region Healing Routines based on Finite Graphs If we want to make use of these concepts within healing routines we first of all need a graph representation for the topology of the geometric shapes to be investigated. At this point the advantage of the BREP modelling scheme becomes apparent since it is well suited for a graph representation. Moreover the fact that edges are oriented is important to be able to apply the theorems, since the boundary operator ∂_* was defined on oriented vertex pairs.
Lets us consider the following topology for three adjacent regions bounded by a sequence of edges, given in picture 3.14.

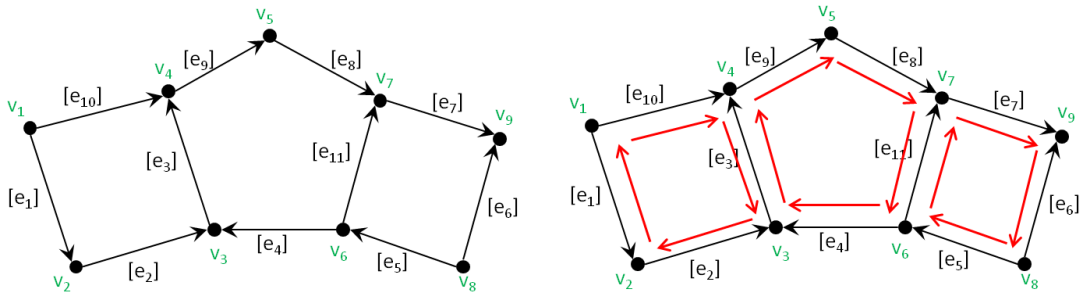


Figure 3.14: BREP model with three regions

For a conform BREP representation it has to hold that a valid region is bounded by a closed boundary wire, i.e. a sequence of edges. Therefore the homology theorem gives us a procedure to investigate the graph determined by the object's topology and make conclusions about the structure and validity with respect to the BREP scheme. In particular the theorem reveals information about cycles in the graph which might be used to form closed boundary wires in the model.
The graph G of the topological structure in picture 3.14 is transferred to the corresponding edge-vertex incidence matrix $\hat{B}$ which reads as follows:

$$\hat{B}(G) \;=\; \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \end{bmatrix} \tag{3.49}$$

The Nullspace $Null(\hat{B}(G))$ has dimensions 1. The cycles obtained from the Left Nullspace on $\hat{B(G)}$, namely $Null(\hat{B}(G)^T) = Null(B(G))$ are the following:

$$Null(B(G)) = c_1 * \begin{bmatrix} -1 \\ -1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + c_2 * \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} + c_3 * \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \\ -1 \end{bmatrix} \tag{3.50}$$

In the graph the fundamental cycles clearly define the boundaries of the three distinct regions as illustrated in picture 3.14. Problematic is the fact that also different bases are able to span the Nullspace, like for example:

$$Null(B(G)) = c_1 * \begin{bmatrix} -1 \\ -1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + c_2 * \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} + c_3 * \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ -1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.51}$$

Therefore suitable cycles corresponding to the "smallest complexes" have to be extracted from the results, namely the Nullspace of the incidence matrices. The next example covers a periodic structure as illustrated in picture 3.15.

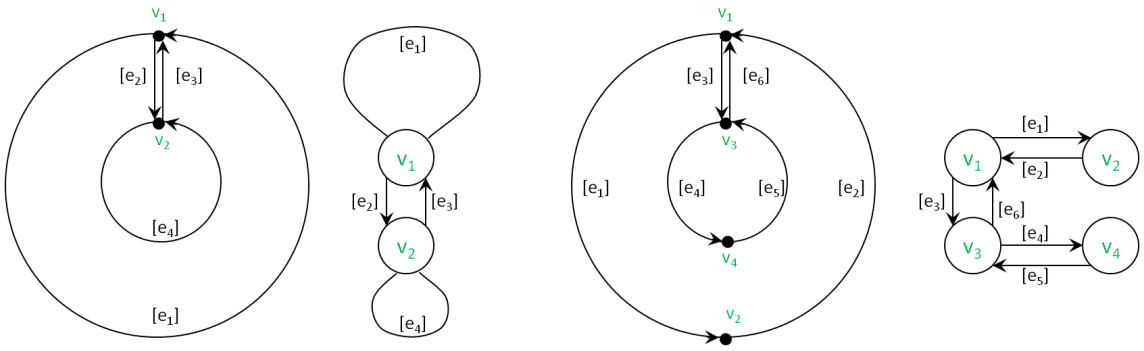


Figure 3.15: BREP model of regions with periodic boundaries

The corresponding graph of this structure suffers from the fact that it possesses loops on the

nodes, meaning start and end vertex are identical for an edge. The structure can be however modified like also sketched in picture 3.15 to break the periodic elements. The corresponding graph on the modified structure is also shown in the picture. On the modified structure we can apply our theorem without problem and obtain the following cycles from the Nullspace on B.

$$Null(B(G)) \;=\; c_1 * \begin{bmatrix}1\\1\\0\\0\\0\\0\end{bmatrix} + c_2 * \begin{bmatrix}0\\0\\0\\1\\1\\0\end{bmatrix} + c_3 * \begin{bmatrix}0\\0\\1\\0\\0\\1\end{bmatrix} \tag{3.52}$$

We detect three fundamental cycles. Again the information we need has to be extracted from these results. All edges can form one structure by taking the linear combination of all cycles: $C_1 + C_2 + C_3$. However with respect to the BREP scheme edge e_2 and e_6 are distinct but not disjoint.

Volume Healing Routines based on Finite Graphs Let us consider a simple BREP volume as sketched in picture 3.16. Clearly the volume is topologically closed and 6 faces are visually detectable bounding the volume.

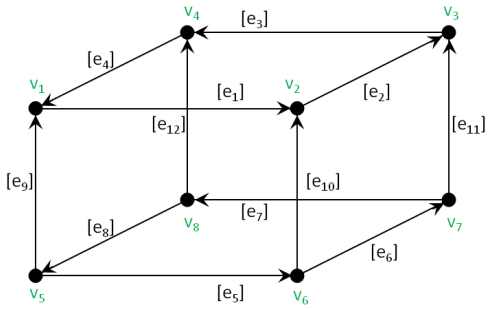


Figure 3.16: Simple BREP volume model

The corresponding graph of vertices and edges looks as sketched in picture 3.17:

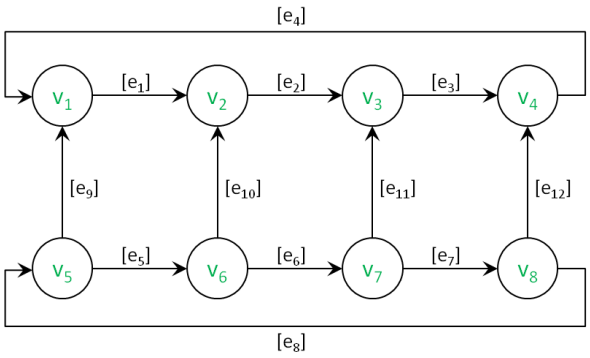


Figure 3.17: Node-Edge graph of a BREP volume

The Nullspace on the edge-vertex incidence matrix B (not written out) of this graph is given

by the following formula:

$$Null(B(G)) = c_1 * \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_2 * \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_3 * \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \end{bmatrix} + c_4 * \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \\ 0 \end{bmatrix} + c_5 * \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{bmatrix} \tag{3.53}$$

The Nullspace is spanned by 5 vectors indicating fundamental cycles in the graph. The meaning of the cycles is illustrated in picture 3.18. They form faces of the BREP volume. Clearly the volume consists of 6 faces, however only 5 cycles span the Nullspace. The missing face, corresponding to a cycle C_6, is a linear combination of all other cycles namely $C_6 = C_1 - C_2 - C_3 - C_4 - C_5$.

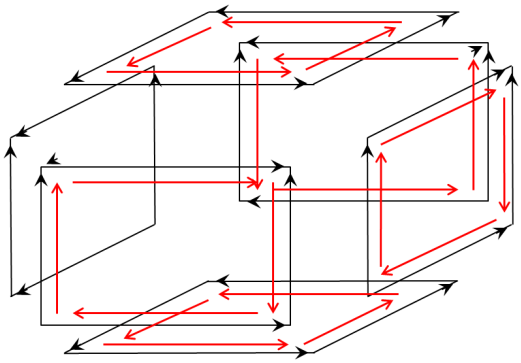


Figure 3.18: Cycles and faces in a BREP model

There exists also another way to analyse the volumetric structure. Consider every face is given by one region. These regions are formed by exactly one closed wire consisting of edges. Assume the consistency and validity is guaranteed for the regions. The regions can therefore be expressed by means of their bounding wires which are translated topologically to cycles in the graph of vertices and edges. Therefore the BREP volume of this example consists of 6 cycles for the 6 bounding faces. Adjacent faces are connected via common edges. Every edge is bounded by exactly two faces in a valid BREP model. This structure can be transformed into a graph structure where the cycles, which form one face, are the graph edges meeting at vertices of the graph, which are the edges in the structure. The problem appears that one graph edge (a cycle in the structure) can have more than two vertices. In our case every graph edge (cycle) has 4 graph vertices for the 4 bounding edges. The graph therefore cannot be drawn in a classical way since we are confronted with a so-called hypergraph.
In a first step we illustrate the structure in a different way. Picture 3.19 sketches again the BREP volume with all of its cycles.

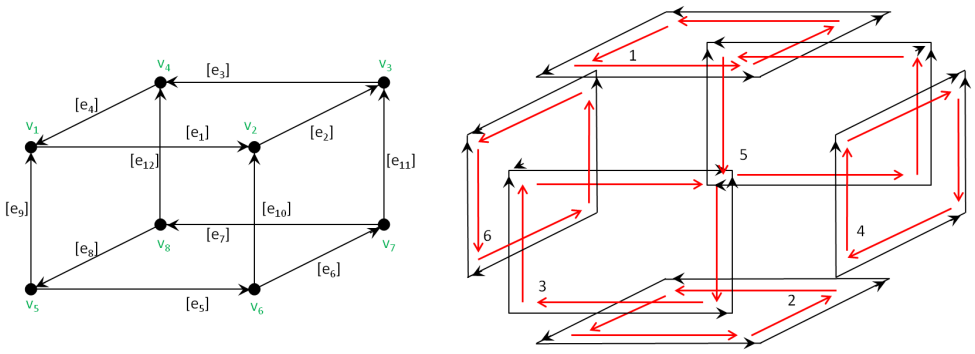


Figure 3.19: BREP model and bounding cycles

We translate this structure in a different graph than we are actually looking for. The cycles form the vertices and the edges form the relations of the cycles.

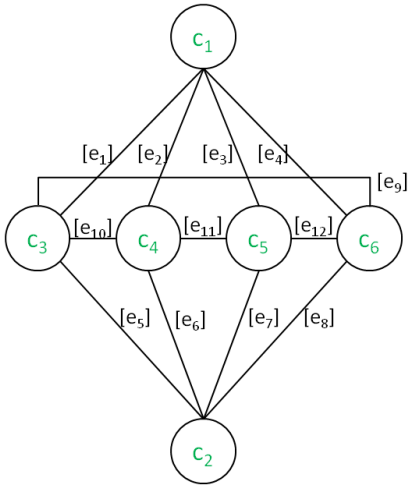


Figure 3.20: Alternative Cycle-Edge graph representation of volume (cycles as vertices)

Picture 3.21 in contrast illustrates the graph we are actually looking for. The cycles form the graph edges in this picture. Since 4 graph vertices are connected to every graph edge, the graphical representation is not intuitive.

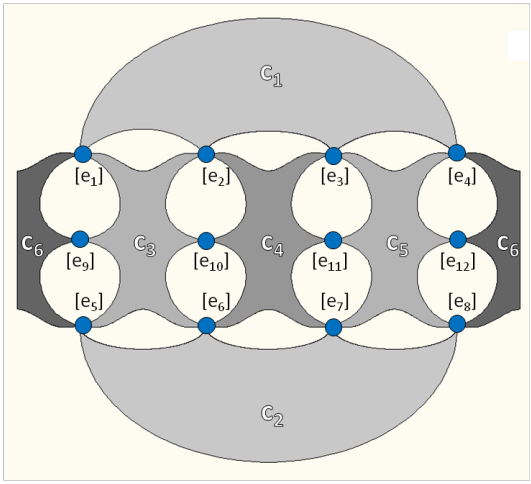


Figure 3.21: Hypergraph of BREP volume

However we can investigate the homology of this graph analogously to the edge-node graph on the same structure. To apply the homology theorem used before we have to orient the vertices in the graph. The vertices refer to the edges in the structure. A orientation convention is derived from the cycle orientation, for example a counter clockwise orientation of bounding edges is defined as positive. If an edge is conform to the orientation of the cycle it holds a positive sign and otherwise a negative sign respectively.
This allow as to formulate an edge-cycle incidence matrix $C = (c_{i,j})_{m \times n}$ of the graph $G = (E, C)$ with graph-vertices $E = \{e_1, e_2, \cdots, e_m\}$ related to edges in the structure, and graph-edges $C = \{c_1, c_2, \cdots, c_n\}$ related to cycles in the structure. The matrix entries are defined as follows:

$$c_{i,j} = \begin{cases} -1 & \text{if } e_i \in c_j \quad \text{and } O(c_j) \neq O(e_i) \\ 1 & \text{if } e_i \in c_j \quad \text{and } O(c_j) = O(e_i) \\ 0 & \text{if } e_i \notin c_j \end{cases} \tag{3.54}$$

Where $O(\cdots)$ is the orientation of the cycle and edge. An initial orientation for each cycles has to be defined. The definition is arbitrary since the procedure will tell us the suitable orientation as one element of the solution. The kernel of the transpose of this matrix $Null(C^T) = Null(\hat{C})$ analogues to the node-edge problem rises information about "balanced" structures in the graph. Balanced means here that every edge belongs to two regions and has a opposite orientation in every of those two regions following the regions cycle orientation. By this a valid and close BREP volume can be constructed.
The regions belonging to one of the volumes in the model are given by the cycles obtained from the Nullspace $Null(C)$.
In our case the incidence matrix $\hat{C}$ is given as seen in the following. The orientation of edges and cycles is chosen as sketched in picture 3.19.

$$\hat{C}(G) = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 1 \end{bmatrix} \tag{3.55}$$

The Nullspace $Null(C) = Null(\hat{C}^T)$ has the following result:

$$Null(C(G)) \;=\; c_1 * \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{3.56}$$

This indicates that all regions contribute to one closed volume and the orientation of the cycle 1 has to be flipped. The nullspace $Null(\hat{C})$ has the following result.

$$Null(C(\hat{G})) \;=\; c_1 \begin{bmatrix} -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ -1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_2 \begin{bmatrix} -1 \\ 0 \\ 1 \\ 0 \\ -1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_3 \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_4 \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} + c_5 \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} + c_6 \begin{bmatrix} 0 \\ 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} + c_7 \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{3.57}$$

We see that actually $Null(\hat{C})$ reveals the cycles in the graph. So again the information we are looking for has to be extracted from the results. We are in this case not looking for the cycles but for the complexes of cycles forming one valid volume.

Chapter 4

TUM.GeoFrame Framework - Geometry Kernel and Mesh Generation Algorithm

4.1 The Existing TUM.GeoFrame Framework

The routines and algorithms developed in the scope of this thesis form a semi-standalone code due to the fact that all the healing algorithms depend on an underlying geometric and topological kernel to describe geometry and topology of the objects to be healed. Implementation of an efficient and robust kernel for geometry and topology is not trivial and could not be covered within the scope of this thesis. A possible remedy would have been to restrict the algorithms and applications to a simple self-implemented two dimensional kernel. By that approach the field of applications would be restricted to simple test examples, excluding almost all practical geometric models.
Since the application of the healer code on more complex models was one of the objectives of this thesis the healer project makes use of an existing framework providing the topological and geometric kernels. Therefore the healer does not form a standalone program, but rather an attachable extension to an existing framework. In general there is a wide range of open source and commercial external libraries usable as geometry and topology kernels. The healer written within the scope of this thesis is an extension for the TUM.GeoFrame framework (formerly Visual DoMesh) developed on the chair of Computation in Engineering (CiE), Technische Universität München. The advantage of using this "inhouse" framework is that a full access to the code is guaranteed. A detailed description of TUM.GeoFrame and its functionality can be found in [45] and [44].
TUM.GeoFrame is a framework environment developed for purpose of research in the field of computational mechanics. Its original functionality covers pre-processing routines, in particular meshing algorithms. It is under constant development and recently extended to form a basis for further processing and post-processing routines. The framework provides a kernel for geometric and topological description of solid models. The geometry kernel of TUM.GeoFrame is based on the external open-source library Open CASCADE [5], implemented into the framework via wrapper classes. Open CASCADE is able to represent almost the entire range of geometric shapes available in modern CAD and CAM systems, including freeform shapes and parametric descriptions. The access via wrapper classes guarantees a uniform and "framework-conform" access to geometric information of the shapes. The topology kernel in contrast to the geometric kernel is developed and implemented at the chair CiE (see [45]), in particular by Mr. Ch. Sorger. Furthermore the TUM.GeoFrame framework provides a graphical user interface (GUI),the so-called TUM.Visual Engine, based on the open-source cross-platform application and UI frame-

work Qt [6]. The post-processor functionalities implemented in the scope of recent development is based on the visualization toolkit VTK [9].
The TUM.GeoFrame framework is providing the core kernel framework and user interfaces and can be used for several routines based on geometric model information. For its original meshing purpose a sub-project called MeshGen is available. Two basic mesh algorithms are integrated in the meshing code in form of external libraries. One library is called DoMesh and was also developed at the chair CiE in the early nineties, see [44]. DoMesh is a surface meshing algorithm for generation of unstructured triangular and quadrilateral meshes. The DoMesh algorithm is based on a boundary decomposition and recursive domain-subdivision procedure as described in [41]. The following steps are included:

1. Form closed loops: all internal holes, lines or nodes are connected to the boundary in form of a closed wire.
2. Boundary edge division: Divide the boundary edges
3. Surface triangulation: Perform recursive domain-subdivision procedure by either choping of a triangle connecting three nodes, which fulfils certain quality criteria or subdividing the domain and retry to find a triangle to be chopped of.
4. Element conversion: in case a quadrilateral mesh is required the triangles are converted to quad-elements.

The MeshGen project is on the one hand simply a wrapper around the two included meshing libraries DoMesh and NetGen (see [4] and [40]). On the other hand MeshGen is able to create volumetric hexahedral meshes for shell-like structures with the help of surface mesh extrusion techniques. The basis for this kind of meshes is a quadrilateral mesh constructed by DoMesh. Volumetric tetrahedral meshes for BREP volume models are constructed in contrast to that by the third-party library NetGen, for which MeshGen provides wrapper functions. NetGen itself is also available as a standalone program with a wider range of functionality. Within TUM.GeoFrame this external library is used to compute tetrahedral meshes starting from an existing surface triangular mesh provided by the DoMesh algorithm.
Picture 4.1 gives an overview over the kernels and routines within the TUM.GeoFrame framework, including the healer sub-project implemented in the scope of this thesis. The code of TUM.GeoFrame is mainly written in C++ and FORTRAN 95.

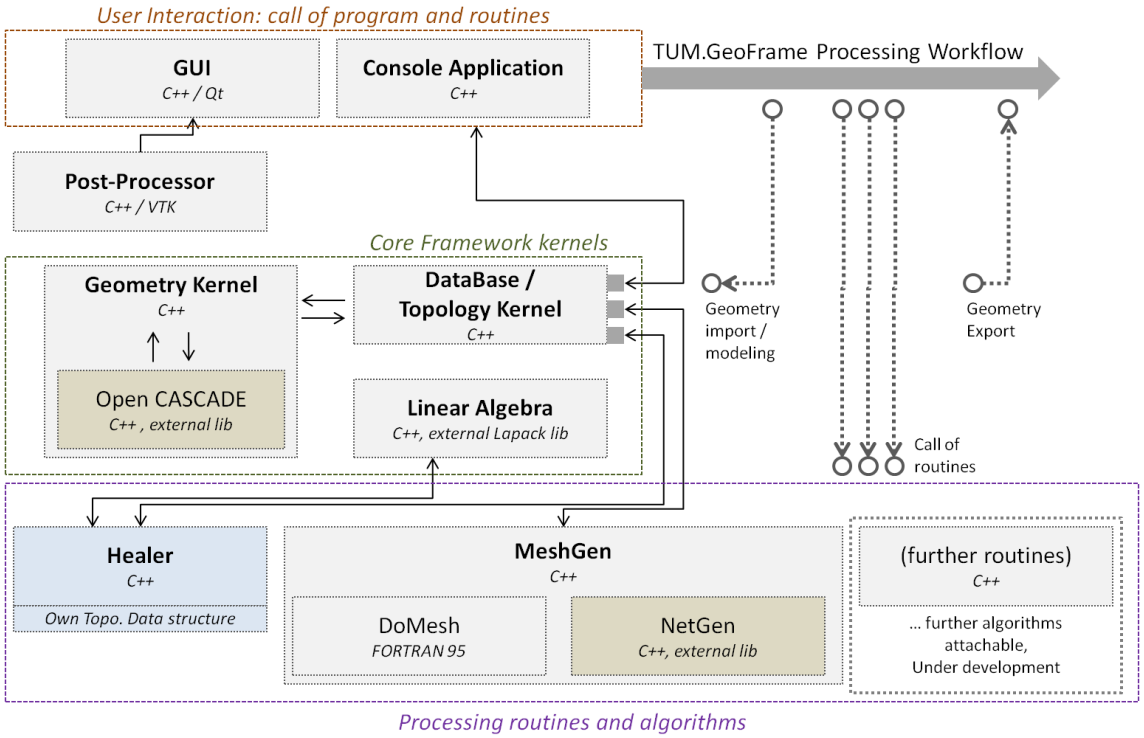


Figure 4.1: Structure of the TUM.GeoFrame framework

The advantages of using an in-house framework developed at the chair became apparent when implementing the healer sub-project. On the one hand the geometry and topology kernels provided by the TUM.GeoFrame framework (called TUM.Geometry Engine within the framework) were able to represent all geometric models on which the healing routines were tested. On the other hand an own framework was flexible to be extended according to specific needs detected during implementation time. The following sections will give a short overview over the most important entities within the topological and topology kernels of TUM.GeoFrame.

4.1.1 The Topology Kernel - DataBase

The geometric representation scheme realized with TUM.GeoFrame is strongly related to a boundary representation scheme for solid models. The terms and notations for the entities and relations in the BREP model as found in literature might differ slightly from the implementation within TUM.GeoFrame. Nevertheless the general concept is equivalent. The topology kernel is included in the so-called DataBase sub-project within the core framework. Besides topological entities also general project information is stored and managed in this subset of the framework. DataBase and geometry kernel are roughly composed of four parts.

- General information, data management and project settings.
- Settings for steering sub-frameworks.
- Entities for topological description.
- Attributes to be assigned to Entities (link to geometric shape description).

The implementation is realized in an object oriented style. For optimized storage and efficient access to set of entities like for example the set of all nodes in the model, the code makes use of containers provided by the C++ STL-library.

The user interaction with the processing routines like healer or mesh generator is realized via setting classes. Those classes are stored in the core DataBase project such that the interfaces between the GUI and the core kernel are clearly defined and dependencies between the GUI and routine sub-projects are minimized.
As mentioned above the topological entities (also called topological primitives) forming the topology kernel of a BREP model are included in the DataBase sub-project whereas the geometry kernel forms an own sub-project. Picture 4.2 sketches the topology entities within TUM.GeoFrame with their corresponding attributes. The existence of classes for edge, face and solid attributes in the DataBase kernel is one of the mentioned differences to a strict BREP structure. These classes hold geometric shape information but not the geometry itself. They provide fast access to shape information exploited for example in model import and export functions.

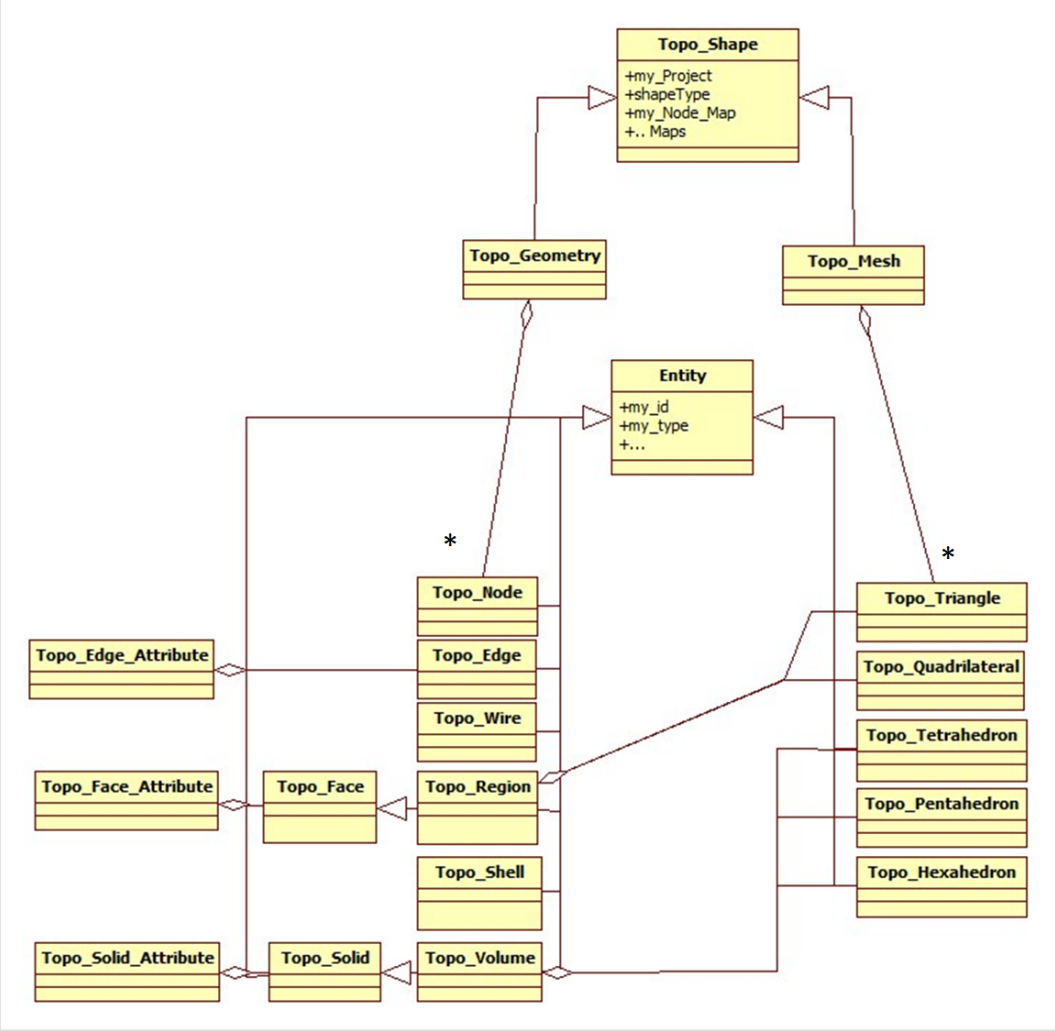


Figure 4.2: Overview over topological entities in the TUM.GeoFrame framework

4.1.2 The Geometry Kernel

The geometry kernel holds the geometric shape information of the entities within the geometric model in form of Open CASCADE objects (referred to as DS objects). The difference to the attribute classes in the DataBase kernel is that the geometric entities (referred to as Geo objects) have underlying geometric Open CASCADE objects. These geometric objects can be used in

operations and algorithms for geometry manipulation or analysis also provided by Open Cascade. On the other hand the graphical visualization is realized with the geometric information in the geometry kernel. The visualization itself is also performed by the Open CASCADE library. In this sense the geometry kernel can be seen as a wrapper around the external geometry library. The reason to provide wrapper classes is to realize a uniform code design within TUM.GeoFrame. Furthermore the usage of frequent geometric operations is optimized within the wrapper classes. Missing operations of the external library can easily be implemented with this concept.
Picture 4.3 sketches the relation of the topology kernel entities - attributes and primitives - and geometry kernel entities, illustrated with the help of an edge element.

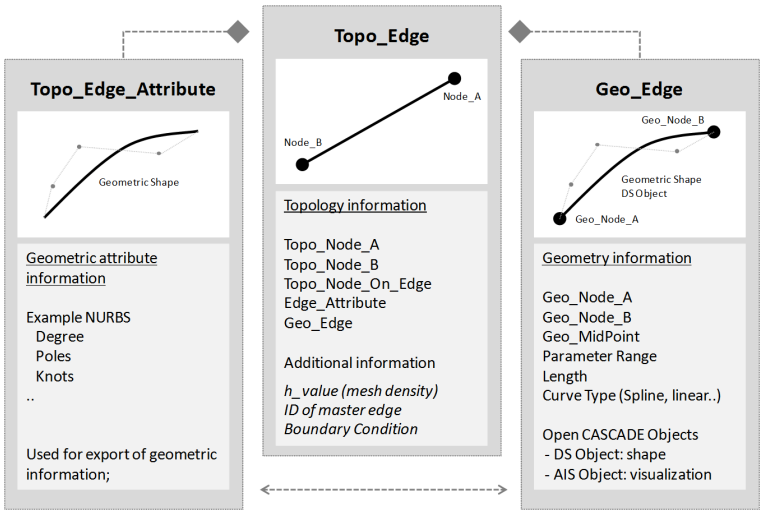


Figure 4.3: Relation between Topo_Objects, Geo_Objetcs and Attributes

In the scope of this work the geometric kernel was extended by several geometric operations needed for geometric healing purposes. Details on the implemented functionalities are given at a later point.

4.2 Implementation of Healing Routines in TUM.GeoFrame

The healer sub-project is only a semi-standalone code as mentioned already in the previous section. A standalone healer code would require an entire local kernel framework for geometrical, topological and general data. To prevent the coding and implementation of local healer kernels identical to the global framework kernels, which would also result in double maintenance effort, the healer sub-project is dependent on the core framework. Nevertheless it was a goal of the implementation concept to create separated sub-projects with clearly defined interfaces. Furthermore the healer code has a local topology kernel but not a geometrical kernel. The reason is that the geometry kernel of TUM.GeoFrame is a wrapper around the Open CASCADE library. Structure and concept of the kernel is motivated by concepts of geometric representation suitable for all routines in the framework. The structure of the topology kernel in contrast is strongly related to the specific needs of the related routines. Therefore a local topology kernel was implemented within the scope of this thesis, optimized for the specific needs of the healer project and healing routines. Furthermore during the healer implementation this structure allowed a more flexible handling of the topology kernel without influencing other routines in the framework. This advantage itself justified the additional implementation effort.
To realize this concept the healer imports the necessary data into its local kernel from the global topology kernel at the start of main healing routine calling. After processing it exports the

information to the global database. During processing time the healer sub-project only has to communicate with the geometry kernel and is therefore not a pure standalone code. Picture 4.4 sketches the interface of the healer project with the TUM.GeoFrame core framework.

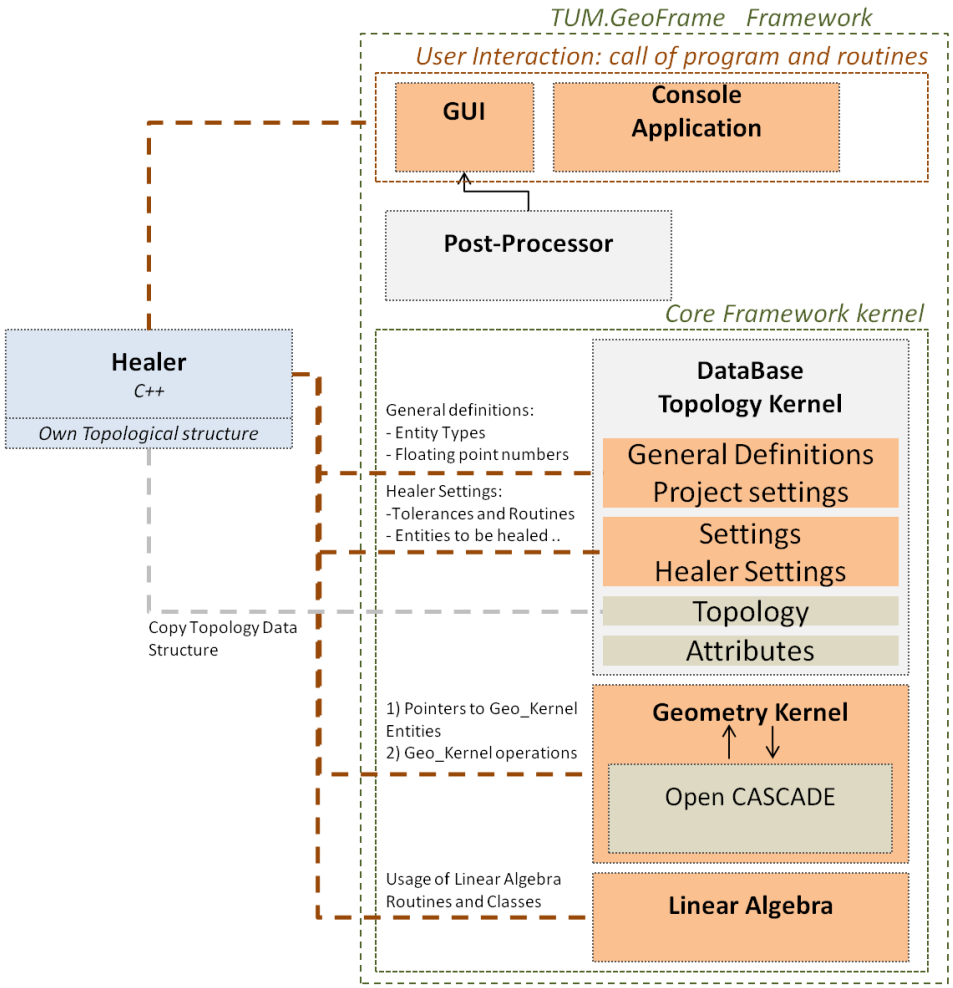


Figure 4.4: Interfaces of the healer sub-project with the core of TUM.GeoFrame

Additional to the healer project added to the framework within the scope of this thesis also further implementations were necessary. First of all the graphical interface (GUI) was adjusted to be able to interact with the healer. Furthermore the global geometric kernel was extended by necessary operations for several healing routines but also useful in the entire framework. An external linear algebra library was attached to the framework via a wrapper project. Before the healer implementation intensive usage of a linear algebra library was not necessary. Picture 4.5 sketches at which point in the framework code was implemented within the scope of this thesis.

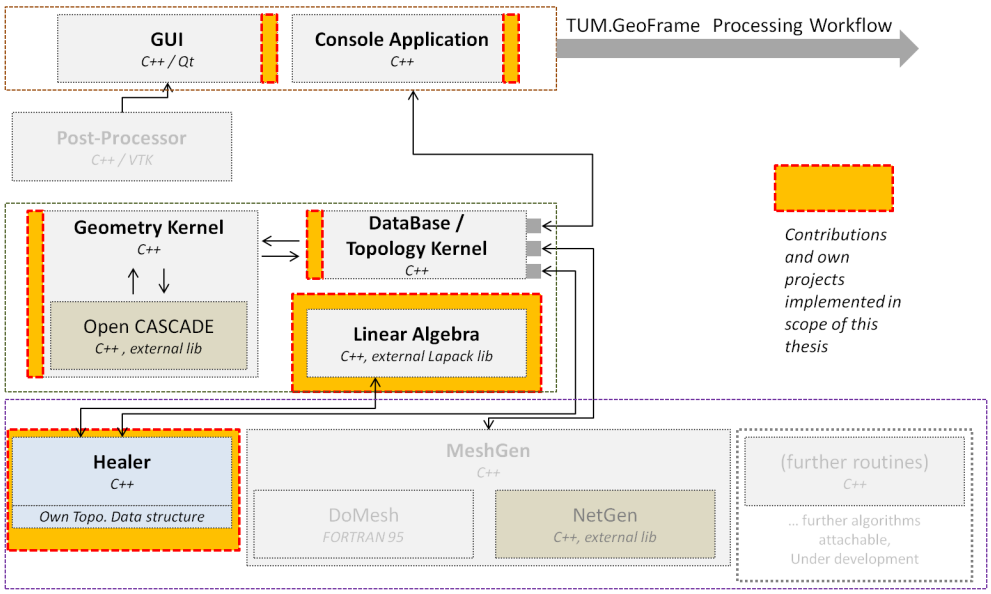


Figure 4.5: Contributions of this thesis to the framework

This concept of a framework with core framework kernels and attachable half-standalone extension sub-projects can be realized efficiently and easily due to the fact that the subroutines in TUM.GeoFrame do not work in parallel. Conflicts accessing global geometric entities cannot appear. The next section will provide a better understanding where and how the healing routines are included in the workflow.

4.2.1 Healer in the TUM.GeoFrame Processing Workflow

Theoretically several positions of healing routines within the processing workflow are possible as illustrated in chapter 2.5. For example the healer might perform before, after or interactively during meshing routines. The healer performs as pre-processing step before downstream applications like meshing directly on the geometric model in the concept realized in scope of this thesis. Other downstream applications on the healed model might be meshing free solution techniques.
Although the mesh is geometrically and topologically described in a similar manner than the original geometry within TUM.GeoFrame, the healer is optimized to perform efficiently on the original model. During meshing the number of entities usually blow up. At the same time the shape description is simplified. Concepts for healing routines on meshes have to take into consideration these characteristics concerning data volume and representation. The pictures 4.6 and 4.7 depict the healing-meshing relation according to the current concept.

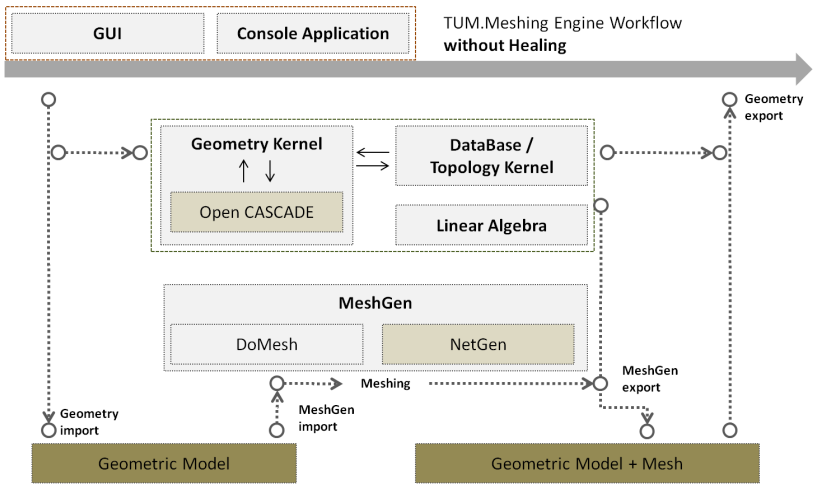


Figure 4.6: TUM.GeoFrame workflow without healer

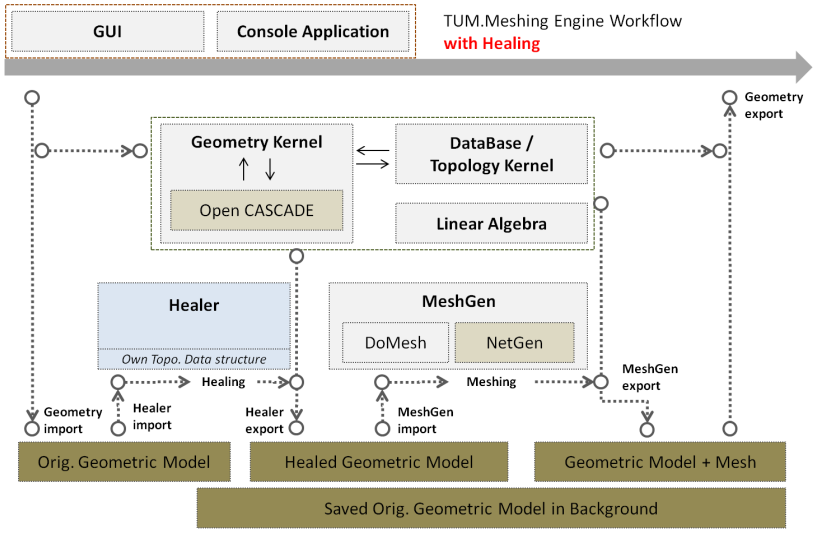


Figure 4.7: TUM.GeoFrame workflow with healer

Picture 4.7 demonstrates that the original geometry is stored in the background when healing routines are called. Since the desired outcome of healing routines might not be achieved due to the ambiguous character of geometric operations or due to improper tolerances the original model is not deleted. Single entities and the whole initial model are able to be retrieved from the background in case healing operations failed.
The user interaction via the GUI is realized by a healing dialog to specify healing tolerances, defining the routines to be applied as well as selecting the entities to be healed. Picture 4.8 shows the current state of the healing dialog. So far the following can be defined by the user:

1. Selection of region-, extrusion volume- or BREP volume-entities to be healed or kept during healing.
2. Value of healing tolerances.
3. Enabling of tolerance adaption.
4. Enabling and Disabling healing operations and functionalities.

5. Enabling of an optimized workflow.

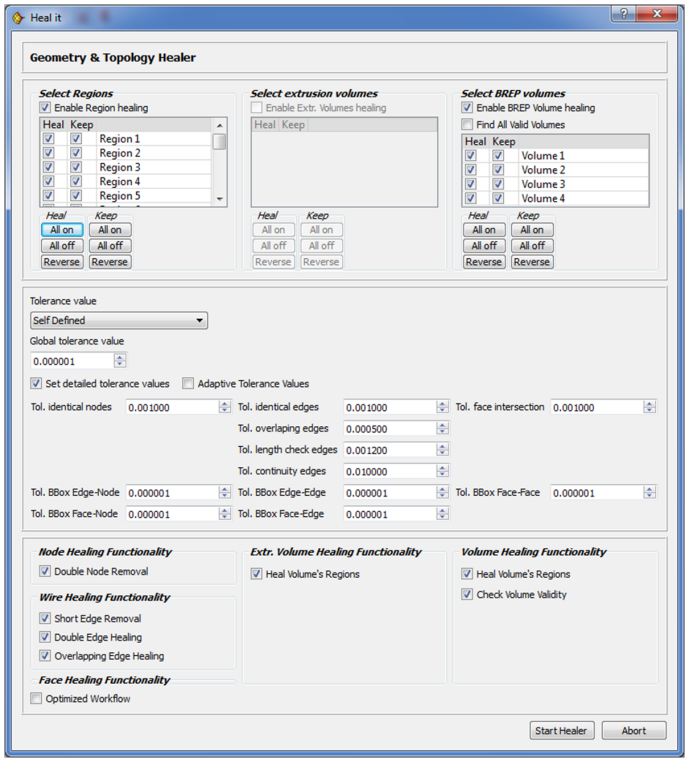


Figure 4.8: User interaction via Healing Dialog within the framework

All major entity groups (region, extrusion and BREP volumes) available for meshing routines are provided with healing routines. It is possible to apply the healing routines only to a subset of the entities and either delete the others from the model or keep them. The "keep" flag does not necessarily mean that geometry or topology of this entity remains unchanged. Since minor entities (e.g. nodes or edges) might be shared with entities where healing routines apply a modification is possible. Nevertheless entities with the "keep" flag are not explicitly passed to healing routines. Chapter 5 refers to this fact more in detail.
The current concept allows a user interaction at the start when healing routines are called and after the routines terminated by calling healing routines iteratively on the healed intermediate models or again on the original model with different settings. Efficiency and robustness of the code might be increased dramatically especially in case of ambiguous situations in case a user interaction within the healing process is realized as depicted in picture 4.9. This kind of user interaction is not yet implemented and described in the outlook in section 7.2.

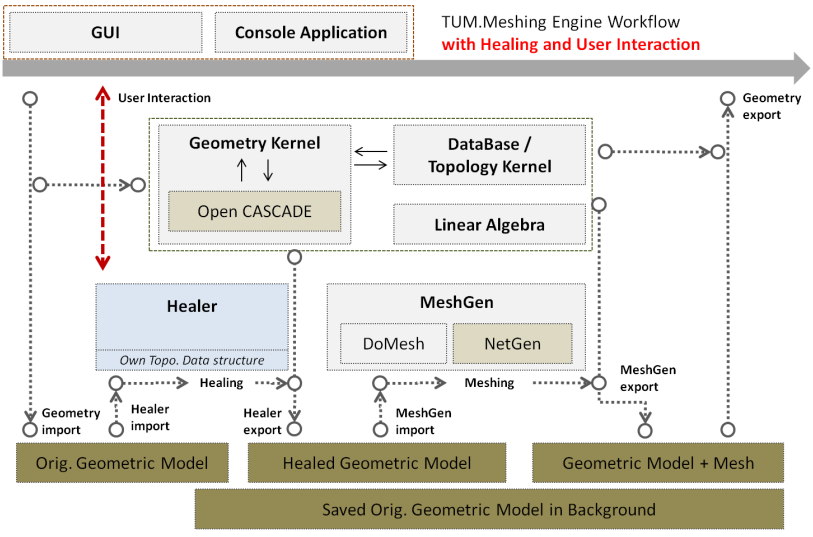


Figure 4.9: TUM.GeoFrame workflow with healer and user interaction

Chapter 5

Implemented Healing Routines and Algorithms

5.1 General Structure and Healing Settings

The structure of the healing sub-project is motivated by two major aspects. On the one hand it is influenced strongly by the original purpose of TUM.GeoFrame, which is to generate meshes for numerical simulations. Due to that fact the same entities which can be selected for meshing can also be passed to healing routines. There are three major groups of entities selectable, namely regions (faces), extrusion volumes ($2\frac{1}{2}$ D volumes) and BREP volumes. All other model entities like edges, nodes etc. are considered only in so far as they contribute to one of these three entity groups. At this point the hierarchical character of the BREP modelling scheme is exploited. This hierarchy is the second major influence on the structure of the healer routines. The hierarchical composition of spatial objects in the BREP scheme is used for the object oriented code design. The details will be presented in detail in the next section.
Every entity within one of these three major groups can be passed to the healing routines and for that purpose two flags can be set via the GUI dialog (or via the console application). One flag controls if this particular entity is to be healed ("bool IsToHeal") and the other controls if this entity is to be kept ("bool IsToKeep") in the healed geometric model. The "IsToKeep" flag is necessary since the healer is importing the topological information of the model into its local database. After terminating of healing routines the global topological database is cleared and the model is reconstructed from the healer's local data structure. Since only entities where either the "IsToHeal" or "IsToKeep" flag is set are imported into the local database at the start of healing all other entities are deleted from the geometric model. Therefore all entities on which no explicit healing routines should be applied have to be marked to be kept. Another consequence of this implementation concept is that all minor entities not contributing to a shape of one of the elements in the major groups (regions, extrusion volumes and BREP volumes) will be deleted from the model. Minor entities are the following within hierarchical BREP structure:

1. Regions:
   - Wires (boundary and internal wires)
   - Edges in wires
   - Nodes (bounding edges and inner fixed nodes)
2. Shell-like structures / $2\frac{1}{2}$D Volumes :
   - Shells (midsurface shell plus top- an bottom shell in case of intersection volumes)
   - Regions in shells

- Wires (bounding regions and internal wires)
- Edges in wires
- Nodes (bounding edges and inner fixed nodes)

3. BREP volumes :

- Shells (boundary plus internal shells)
- Regions in shells
- Wires (bounding regions and internal wires)
- Edges in wires
- Nodes (bounding edges and inner fixed nodes)

The fact that all minor entities which do not contribute to one of the major elements will be deleted is desirable for classical BREP models. Like that the model consistency can be guaranteed. In case those strict BREP validity conditions are not applicable for a certain model deleted entities can be retrieved from the saved background geometry.
The user interaction is realized via a healing dialog implemented in the GUI of TUM.GeoFrame. The interfaces between healer, GUI and core framework kernels were sketched in section 4.2. It is desirable to minimize the interface points between the processing routines and the core framework. One reason is to decouple the dependencies between the sub-processes. Like that single processing routines can easily be excluded from the framework on compilation time if desired or necessary. On the other hand this concept makes maintenance and debugging easier. For that reason the entry point into the healer subproject is one main file.
This main file is grouped into three major sections referring to the three major entities in the model (regions, $2\frac{1}{2}$D and BREP volumes). The code design follows wherever possible the idea to split processes in three steps: pre-processing, processing and post-processing. This holds for the entire main file as well as within each section for one of the major entities. In other parts of the code this concept is applied in a similar manner. A continuous and consistent code design significantly simplifies debugging and maintenance as well as the understanding of algorithms and routines for programmers entering the project at a later time, in particular for large projects. Conversely code without any continuous design might even lead to a deadlock situation where future development and maintenance is impossible and the code has to be completely redesign. Besides the grouping into major entities and pre-, post and processing steps the implemented algorithms are classified into topological and geometrical types. Again the reference to the BREP model was tried to transfer to the code design with the goal to make the code more understandable.
Since many healing algorithms depend on search routines the efficiency can be increased by flags assigned to all types of entities to identify their status concerning:

- Visited by an algorithm
- Temporary occupied by some routine
- Invalid according to a BREP model representation
- Successfully healed

These flags are only used and moreover meaningful during processing of healing routines which motivated the usage of a local topology kernel in the healer sub-project. This distinction of global and local topology increases the flexibility to adjust the data structure for certain processes.
Tolerances defined for various geometric healing routines can have a significant influenece on the outcome. Therefore the next section will focus on the implementation of tolerances within the healer code.

5.1.1 Concept and Implementation of Healing Tolerances

At the current state there are twelve different tolerance values available within the healer code. These values are defined and accessible globally in the entire TUM.GeoFrame framework. The global definition in the core framework simplifies the interaction with the GUI enabling the user to modify these values. The different tolerances are exploited in various healing operations. Table 5.1 gives an overview over the values and their related healing algorithms.

Name	description	Related entity	Default value
DEFTOL_GLOB	Global tolerance value if no specific value defined	-	1.0E-6
DEFTOL_IDENT_NODES	Defines if two nodes are identical	Nodes	1.0E-3
DEFTOL_IDENT_EDGES	Defines if two edges are identical	Edges	1.0E-3
DEFTOL_OVERL_EDGES	Defines if two edges have a partial overlapping section	Edges	0.5E-3
DEFTOL_LNGTH_EDGES	Checking zero lengths of edges and length comparisons	Edges	1.2E-3
DEFTOL_CONTI_EDGES	Checking continuity when merging adjacent edges	Edges	1.0E-2
DEFTOL_INTERSECT_FACES	Checking intersection of faces	Faces	1.0E-3
DEFTOL_BBOX_EDGE_NODE	BBox intersection check: edge to node	Edge - Node	1.0E-6
DEFTOL_BBOX_FACE_NODE	BBox intersection check: face to node	Face - Node	1.0E-6
DEFTOL_BBOX_EDGE_EDGE	BBox intersection check: edge to edge	Edge - Edge	1.0E-6
DEFTOL_BBOX_FACE_EDGE	BBox intersection check: face to edge	Face - Edge	1.0E-6
DEFTOL_BBOX_FACE_FACE	BBox intersection check: face to face	Face	1.0E-6

Table 5.1: Tolerance values

The default values are defined statically in the framework. However the code has a dynamic tolerance policy. Therefore in the pre-processing section of the healer's main file an instance of a global tolerance class is created. This global tolerance class can be seen as the access to tolerance values for all algorithms and routines in the healer code. The global tolerance class holds a mirrored local member variable for each of these twelve tolerance values. At creation time the user can define the initialization of these values as follows:

- Set to global default: the default values listed above are assigned to the memeber variables in the tolerance class.
- Defined by domain size: According to the real domain size of the objects the tolerance values are adjusted.
- Self-defined values: The user can set (via the healing dialog) each tolerance value himself.

Adjusting the tolerance values according to the domain size is currently related to the average and shortest edge length. The proper values for stable results are still to be determined by testing the code against a wider range of practical examples. This instance of the global tolerance class is the gateway to the tolerance values whenever a tolerance is requested by some algorithm. Besides this top level tolerance settings the user is able to assign to every primary entity (nodes, edges, regions and volumes) an instance of an entity tolerance class. This class is basically a mirror of the global tolerance class, but its influence is restricted locally to a single entity.
When a tolerance value is requested by an algorithm the global tolerance class returns the tolerance value. Therefore the global tolerance class searches in a strictly hierarchical manner the most locally defined tolerance value. Therefore all local instances for entity tolerances are known to this global class. By this concept it is possible to react to local model characteristics and accuracies. Picture 5.1 depicts the concept with its intrinsic hierarchical order.

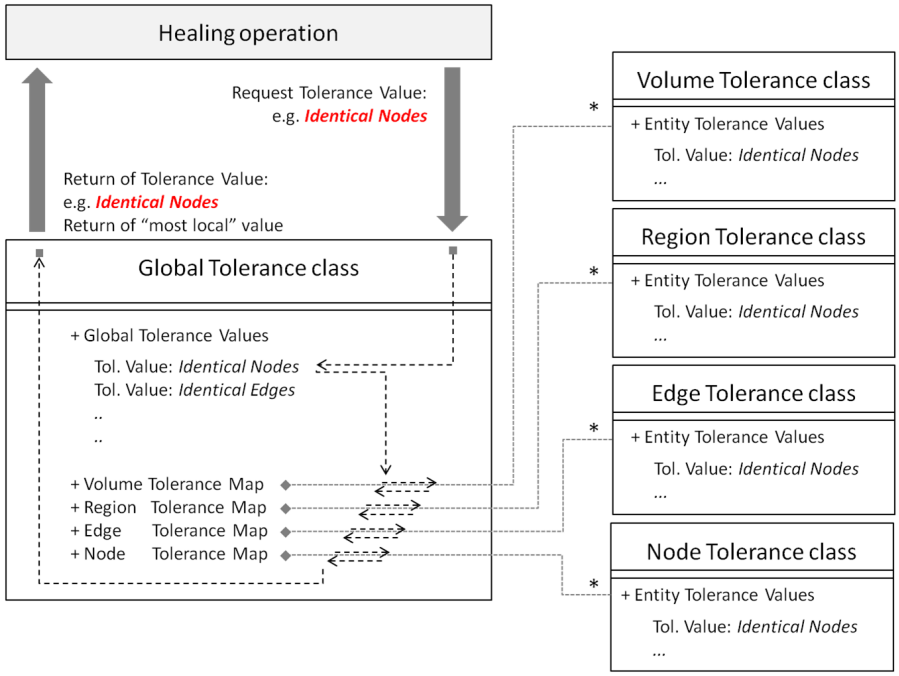


Figure 5.1: Hierarchy of tolerance values

The entity tolerance classes enable a local tolerance policy. In addition an adaptive approach is realized in the healer code. The user can activate this function via the GUI dialog. The goal of this concept is to automatically adjust tolerances dynamically to local characteristic of the model within the healing process. Therefore each tolerance value is equipped with a so-called **adaption factor**. If adaption is enabled the code checks for several routines if a certain operation was not performed due to a tolerance value, but would be performed with a lower tolerance value. The adapted tolerance value is simply the original tolerance multiplied by the adaption factor.
Most healing routines have a recursive structure which means that a specific healing operation is called only if various simpler tests succeeded and the outcome of the operation is likely to be the desired model result. Bounding Box checks are an example for such kind of simpler test. This structure increases efficiency and robustness. At a certain point in this recursive structure the success of a specific healing operation is simply determined by the assigned tolerance value. At this position the code checks if the healing operation was not conducted according to the current tolerance settings and in consequence enforces an adaption of the tolerance value. A flag in the data structure indicates that healing routines are to be called again since a tolerance

adaption was performed. A second flag within the data structure checks if iterative calls due to tolerance adaption actually have any influence on the model to stop the iteration in case not. As in the reverse direction the global tolerance class has the function of a gateway to assign the adapted tolerance value most locally. Picture 5.2 sketches the concept.

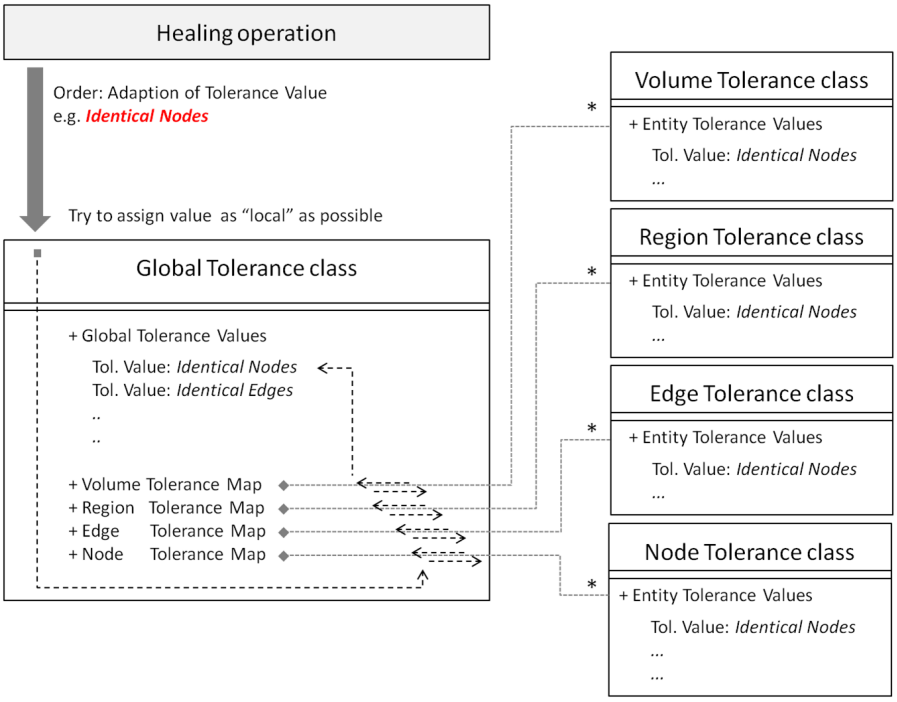


Figure 5.2: Hierarchical tolerance adaption workflow

The implementation of tolerance adaption within the overall healing workflow is described in detail in the following section.

5.2 Healing Routines and Processing Workflow

The implementation of the healer sub-project can be divided into two major aspects. First of all several healing algorithms and operations had to be implemented. Input and output arguments as well as an error handling concept were defined for each algorithm.
In a next step the single algorithms had to be bundled to form an efficient entire healing routine. It turns out that this step is actually the crucial and time consuming part. The reason is that efficiency and robustness of many healing algorithms might depend significantly on the sequence of calling. Algorithms which make intensive use of costly geometric checks between entities in the model for example are more efficiently implemented after the model size is decreased due to previous applied healing operations. Another example is the concept how topological information is updated in the data structure. If the topological relations (e.g. neighbouring relations) are costly to be determined it might be more efficient to update them only if requested by an algorithm. These two examples indicate that the concept how to structure the healing routines are of great importance for the overall efficiency, stability and robustness.
Section 5.1 illustrated that entities belonging to one of the three major groups (regions, $2\frac{1}{2}$ and BREP volumes) are provided with specific healing routines. Starting from these major entities healing routines on all minor entities are called recursively following the intrinsic hierarchy of the BREP model. Picture 5.3 sketches the implementation in the code graphically.

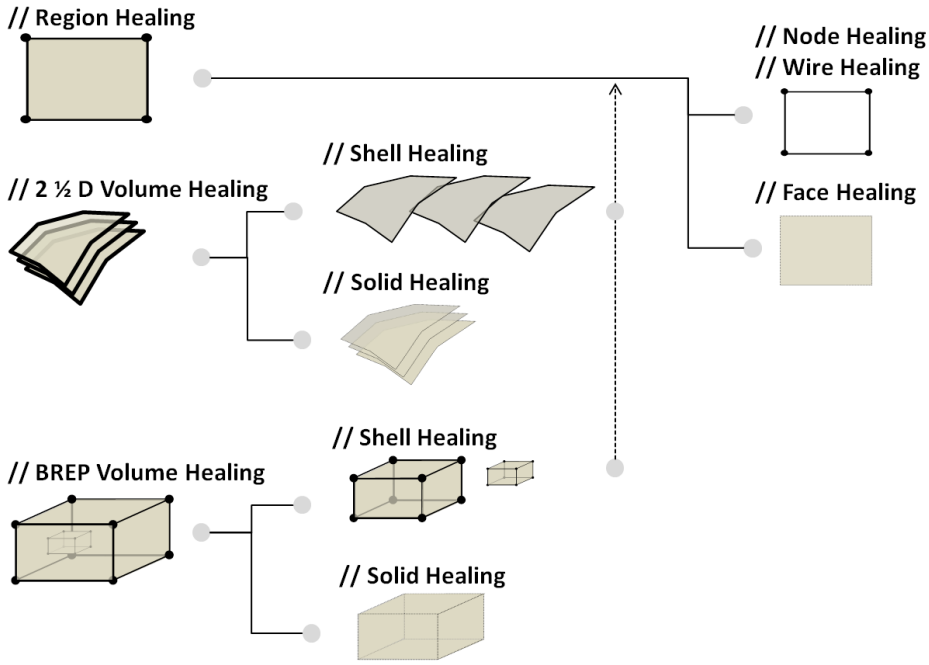


Figure 5.3: Hierarchical recursive implementation concept of healing routines

As illustrated in 5.1 this concept was chosen to translate the BREP scheme in an object oriented code. The healer is therefore optimized and adjusted for a BREP modelling scheme as it is also necessary for meshing routines in TUM.GeoFrame. However the healing algorithms in contrast to the routines are more independent from the modelling scheme. In the scope of future development the code can therefore be extended to other schemes by providing suitable processing workflows.
It was also mentioned in 5.1 that the routines are divided in pre-, post- and processing sections. The processing steps are described in the upcoming sections. The purpose of pre- and post-processing steps is the following:

1. Pre-processing:
   - Import topology of entities to be healed or kept into the healer.
   - Initialize flags for all imported entities (not visited or temporary occupied, not yet healed and valid).
   - Backup each entity in a local back-up copy. (not to be mixed with the global background geometry which holds a copy of the entire original geometric model).
   - Define tolerance values according to the domain of the model to be healed, if this function is enabled (otherwise set to default or self-defined values).
   - Derive required bottom-up topological information, for example setting:
     · to which regions an edge or node belongs
     · to which volume a region belongs
   - Delete minor entities not contributing to the BREP model, for example isolated nodes or edges. By this many algorithms work more efficient, especially as it turned out that some modelling systems add a significant amount of this spurious entities.
2. Processing: Healing operations as described in the following section in detail

3. Post-processing:

   - Final setting of entity-flags: no entity temporary occupied, all entities visited.
   - Final check of validity. This is necessary since the entity based implementation of algorithms in a loop over all entities might lead to the fact, that a entity (a region for example) modifies a sub-entity (like an edge) and therefore also modifies the topology of other entities (regions) sharing that object but visited earlier. Therefore a final loop over all entities checks their validity concerning the BREP representation.
   - Invalid entities after performed healing routines are replaced by the local back-up copy of them. In case the initial entity was invalid according to the BREP scheme and could not be healed the result is an invalid model. Otherwise if the validity was affected by failed healing operations the result is a valid but possibly inconsistent model. The user is informed about this step to be able to react accordingly.
   - Delete backup topology and free memory for all valid entities.

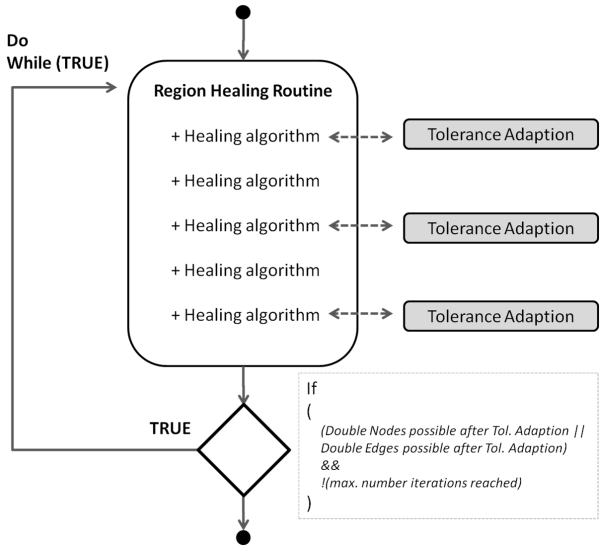


Figure 5.4: Adaptive tolerance concept in the region healing workflow

As already mentioned the result of many healing operations is ambiguous and might depend significantly on the defined tolerances. This fact motivated the dynamic and adaptive tolerance policy described in the previous section 5.1. It was illustrated that tolerance adaption are useful to react on local model characteristics and accuracy. However it was also indicated that the tolerance adaption requires an iterative approach. In the healer code iterative call of healing routines is disabled by default and has to be enabled by settings a so-called "optimized workflow" flag. As a result the healing routines are called iteratively until either the model is not changed anymore or no adaption is performed anymore during healing. In this final iteration state it holds:

- Double nodes and edges are not likely to be present in the model even if tolerance values are adjusted by their adaption factors. ("near-double" nodes or edges are not likely anymore)
- All short edges are merged if they fulfil the continuity conditions (and additional topological checks) or continuity conditions with adapted tolerance.
- A maximum number of iterations is not excessed.

All conditions have to be seen with respect to the tolerance settings. The combination of adaptive tolerances in a optimized workflow is a powerful iterative approach dynamically adjusting the tolerance settings to local model characteristics. Picture 5.4 sketches this workflow.
However workflow optimization and tolerance adaption can be selected independently from each other. An iterative calling after tolerance adaption might be undesired and the user can manually check the adapted tolerances and initiate a second healing run himself.
On the other hand an optimized workflow without tolerance adaption can also be chosen. This configuration will result in an iterative healing calling. The reason why an optimized workflow

without tolerance adaption might be desirable is that the workflow optimization includes more than simply an iterative approach. It turns out that several healing operations perform more stable and efficient if other operations where already successfully applied on all entities in the model. In general the sequence of operation calling is defined in the basic workflow. Therefore the overall stability and robustness might be improved if certain routines are disabled in a first run and enabled in a second or later run. The most efficient implementation of workflow optimization is still to be figured out by applying the code to a wide range of practical examples. The following example should illustrate the motivation for workflow optimization. To determine if edges have partially non-consistent intersections with other edges costly geometric checks against all edges in the model have to be performed. This operation is more efficiently and robustly applied if the model size is reduced and as many edges as possible are merged or deleted from the model in prior runs. Therefore the so-called edge overlapping healing is disabled at the beginning and enabled in a second or later run when double edges or short edges are already healed in the model. Details on the algorithms mentioned in this example are given in the next section. Picture 5.5 sketches the idea of an optimized workflow with or without tolerance adaption.

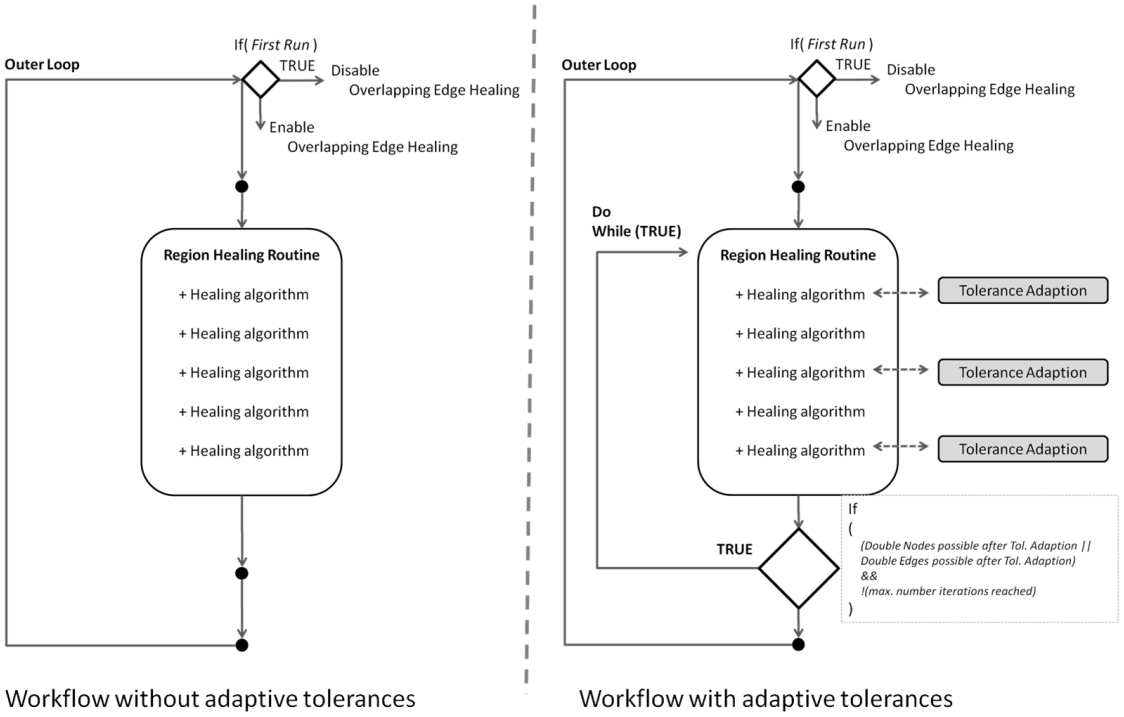


Figure 5.5: Optimized healing workflow concept

The following sections illustrate and describe the specific processing routines related to the three major entities: regions, $2\frac{1}{2}$D and BREP volumes. Each entity is equipped with a corresponding healing routine. Due to the hierarchical decompositions region healing routines are recursively called during volume healing routines.

5.3 Region Healing Routines and Algorithms

Regions of a BREP model are bounded by one closed wire and might consist of internal wires describing holes or internal lines. The geometry of the region's face is defined by the geometric shape attributes. As illustrated in section 2.2.7 the geometric shapes are often given in terms of NURBS for complex models.

5.3.1 Region Healing Workflow

The region healing workflow is related to the hierarchical object composition. In a first step node healing algorithms are called to merge all double nodes. In a next step wire healing routines for the boundary wire and all existent internal wires are called. Only after that the face information itself is considered in face healing routines.
The region healing loop is included in the healer's main file in the following way:

```

programm Healer_main(Shape)
{
  ..

  // Start section Region Healing

    // preprocessing
    call Region_Healing_Preprocess(..);

    // processing
    loop(over all regions in shape)
    {
      if(region is to heal)
        call Region_HealIT( aRegion, ..);
    }

    // postprocessing
    call Region_Healing_Postprocess(..);

  //

  ..
} // end Healer_main
```

Code Region 5.1: Region Healing Workflow

For a single region the healing workflow has the following structure:

```

programm Region_HealIT(HealIT_Region* aRegion)
{
  // -- Node Healing --
    // delete double nodes from region
    call DeleteDoubleNodes(..);

  // -- Wire Healing --

    // set boundary wire and heal it
    call Healing_Wires(Boundary_Wire);

    // loop over internal wires
    loop(over all internal wires)
    {
      // set internal wire and heal it
      call Healing_Wires(Internal_Wire_i);
    }

  // -- Face Healing --
    // not yet implemented

```

```
23 } // end Region_HealIT
```

Code Region 5.2: Region Healing Routines

5.3.2 Wire Healing Routines

Wire healing routines are recursively included into region healing routines. Each Wire part of a region description is passed to the wire healing routines. A given sequence of geometric and topological healing operations is performed for the wire and all its minor entities (edges and nodes). The specific operations and their sequence of calling are the following:

```
1
2  program Healing_Wires(HealITWire* aWire)
3  {
4    // delete edges with zero length
5    call RemoveAllZeroLengthEdges(..);
6
7    // sort edges in wire
8    call Sort_Wire(..);
9
10   // remove short edges from wire by merging
11   call Wire_ShortEdgeRemove(..);
12
13   // delete double edges from wire
14   if(call DeleteDoubleEdges(..) = true )
15   {
16     //perform only if a double edges was detected and deleted
17
18     // second sorting run
19     call Sort_Wire(..);
20
21     // second run short edge removal
22     call Wire_ShortEdgeRemove(..);
23   }
24
25   // 3rd run sorting if necessary (steered by a flag)
26   If(sorting_necessary)
27     call Sort_Wire(..);
28
29   // loop to heal overlapping edges
30   do
31   {
32     if(call DeleteOverlappingWires(..)= true)
33     {
34       // perform only if a overlapping edges manipulated in wires
35       call RemoveAllZeroLengthEdges(..);
36       call DeleteDoubleEdges(..);
37       call Sort_Wire(..);
38     }
39
40   } while(overlapping edges possible)
41 }
```

Code Region 5.3: Wire Healing Routines

In the next sections a short overview over purpose, structure and performance of the specific sub-routines mentioned above is given.

Remove Zero-Length Edges

The operation {RemoveAllZeroLengthEdges(..)} is deleting edges of zero length. Since we work in a floating point environment zero length is defined by a tolerance value which determines the minimal acceptable edge length. Zero length edges can be present due to the following reasons:

- Due to healing operations: routines like for example the healing of overlapping edges might decompose edges in short sections below the zero length tolerance. Therefore these edges are deleted from the model by merging with neighbouring edges.
- Due to modeller software: it turned out that several third party systems for geometric modelling which were used to produce the models introduce spurious edges with no physical length. These edges cause problems not only in further healing operations but also in meshing routines.

Wire Sorting

The operation {Sort_Wire(..)} has the purpose to sort the sequence of edges in a wire according to their connectivity. In the original BREP description the ordering is decisive because it defines if a wire describes a hole or an outer boundary. The convention is that sequences in counter-clockwise order describe boundaries and in clockwise order holes respectively. This convention is not necessary within the TUM.GeoFrame topology kernel since the wires are equipped with a wire type defining their purpose in a region (hole or boundary). Nevertheless it has to be ensured that the wire is closed (if not describing an internal line) and edges are given in a sequential order. Unordered wires might result from the input model itself. Certain file formats turned out to suffer from that problem. On the other hand healing operations might destroy the sequential order of the edge vectors.
This function orders the edges in a wire without considering the orientation since it is not necessary for the topological kernel. The orientation is important in downstream applications or when exporting the model. The correct sequence is defined and set only when required by functions of the core topology kernel.
This function is also used to determine if healing operations failed and the region description is defective. A defective description is recognizable by the fact that it is not possible to find a closed wire. This holds only for hole and boundary wires. In that case a flag is set to indicate the invalidity. During post-processing the original entity will be retrieved form the background. If the original model was already defective and could not be healed user interaction is necessary to construct a valid model.
The sorting function consists of three main steps.

1. Check if input wire is already in a ordered sequence and forming a closed wire for boundaries and holes. Therefore the endpoint connectivity of the edges in the wire vector in order of their appearance is analysed.
2. Simple sorting algorithm based on endpoint connectivity
3. Sorting using the Nullspace of the node-edge incidence matrix

In the first step the algorithm loops over the edges in the edge vector of the original wire and checks their topological connectivity. Furthermore the wire must be closed to describe a boundary or hole and therefore the first and last edge in the wire must be topologically connected. If this routine fails the wire has to be sorted. A simple algorithm tries to construct a sorted (and closed) wire:

```
1
2  program Simple_Edge_Sorting(HealITWire* aWire)
3  {
4    // set neighbouring relations
5    call set_Neighbouring_Rel(..);
6
7    // a new empty wire
8    new HealITWire* Sorted_wire;
9
10   // add first edge of old wire to new one and delete from old
11   set my_prev_id = First_Edge->id
12   aWire->Delete(First_Edge);
13   Sorted_wire->Add(First_Edge);
14
15   // loop over edge vector until all edges are inserted in the new wire
16   do
17   {
18      loop(over edges in wire)
19      {
20         // set edge and check if neighbored to previous edge
21         Edge_i
22         if(Edge_i->Neighbor->id = my_prev_id)
23         {
24           set my_prev_id = Edge_i->id
25           aWire->Delete(Edge_i);
26           Sorted_wire->Add(Edge_i);
27         }
28      }
29
30   } while((aWire->notEmpty) || !(maxIterations))
31 }
```

Code Region 5.4: Simple Edge Sorting

The algorithm is simple and stable. Nevertheless it will fail for wires which are not closed. For boundary and hole wires this is a sign that the wire is defective and invalid according to the BREP scheme. In case of a internal wire which does not have to be closed this algorithm will fail if the procedure does not start with the first edge by chance. To overcome this problem the code will be adjusted in future development. The basic idea is to try to attach edges to the other side of the wire where the procedure started in case one end is reached.
The algorithm will also fail if a dangling edge is present in the model as shown in figure 5.6.

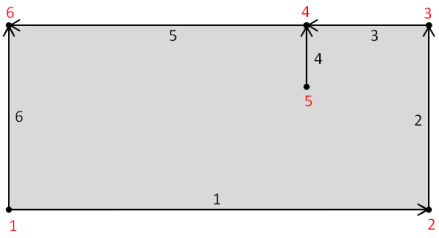


Figure 5.6: Region model with a dangling edge

If the first two sorting algorithms fail a flag will enforce the third applied routine to sort the edges in the wire. This routine is based on the edge-node incidence matrix and its kernel (Nullspace). The theoretical background is given in section 3.2.2. The kernel of the node-edge incidence matrix (NE-Matrix) detects closed cycles in the graph, meaning edge sequences forming closed wires. In contrast the edge-node incidence matrix (EN_matrix) can be used to determine of all nodes in a sequence of edges contribute to one path. This result can also be exploited in case of a non-closed wire The map with the edges and nodes in the wire is past to a BREP_Region_Dectection class. This class is setting up the corresponding matrices and computes the kernels or Nullspaces. Exactly one Nullspace vector of the (NE_matrix) is the optimal result for a closed wire since the solution is

given immediately. As described in section 3.2.2 we will find two vectors for two fundamental cycles in the graph for the example given above in case the edge vector reads $\{1-2-3-4-4-5-6\}$. In contrast to that for a edge vector $\{1-2-3-4-5-6\}$ only one vector, namely the outer wire without the dangling edge is found.
The desired structure has to be extracted from this solution. In the second case the only possible boundary vector is excluding the dangling edge. In the first case the union of both results will form a valid BREP description. Nevertheless within the TUM.GeoFrame framework the desired output is to exclude the dangling edge no. 4 from the wire and include it into the model in form of a internal wire. Chapter 6.2 will refer to the sketched example in picture 5.6 to explain the effects of the algorithm.

Short Edge Merging

The algorithm { Wire_ShortEdgeRemove (..)} has the purpose to delete short edges from the geometric model by merging them with neighbouring edges according to their continuity at the endpoints. The presence of short edges is not a violation of BREP validity conditions as it was also described in section 3.1. Nevertheless the presence of short edges in the wires is often not desired, especially with respect to downstream meshing routines (DoMesh algorithm). The source of this error is often a translation problem when reading geometric information from file formats like IGES. Originally continuous edges might be decomposed resulting in the short edge problem. Example Ex_01 in section 6.2.1 illustrates sources and consequences of this defect more in detail.
The short edges are merged to form longer continuous edges. Therefore two basic tests have to be performed. A geometric test has to reveal information about the continuity between two edges. Furthermore a topological test is necessary. Since the edges of interest might also be bounding edges of neighbouring regions a pure geometric check would not be enough. It has to be guaranteed that the merging step is not destroying the wires of neighbouring regions, which is the purpose of the topologic test. In other words the set of regions to which the edges are assigned has to be the same for both edges in order to merge them. Picture 5.7 is illustrating that graphically.

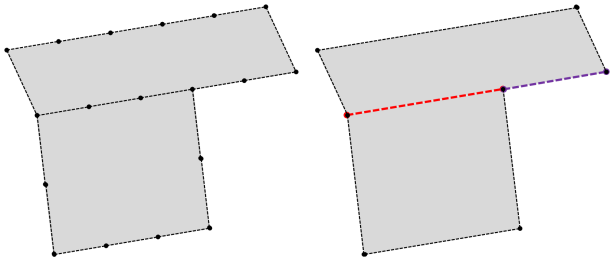


Figure 5.7: Edge merging procedure

In a first step the algorithm loops over the edge vector of the wire to obtain information on the edge types within. Accordingly three different short edge merging algorithms are available. In case all edges in the wire are linear an own merging function is called. The reason is that the linear description of the edges should be preserved and that the geometric test is simplified. The geometric function analysing the continuity of two edges reduces to a interpolation check illustrated in picture 5.8.

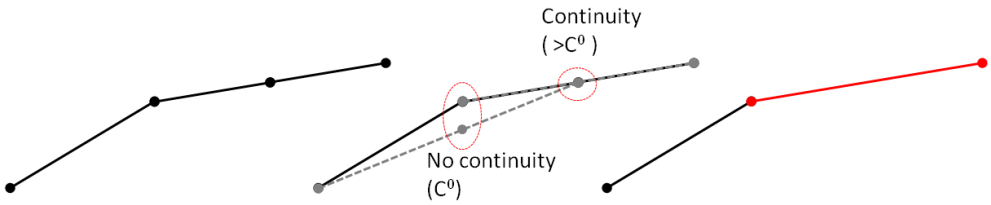


Figure 5.8: Linear edge merging procedure

In case all edges in the wire are given by a NURBS description another merging function is called. Within this function the geometric test as well as the merging is outperformed by Open CASCADE functions determining the continuity and creating one new NURBS edge from two edge segments.
The third available function is called in case the edges in the wire have mixed algebraic and parametric description. This function is not yet implemented and scope of future development. Purpose is to save algebraic information for edges. The basic idea is to form composed edges from the two or several single edge segments to be merged (see picture 5.9). By the corresponding parameter intervals the composed edge retrieves the geometric information from its underlying geometries.

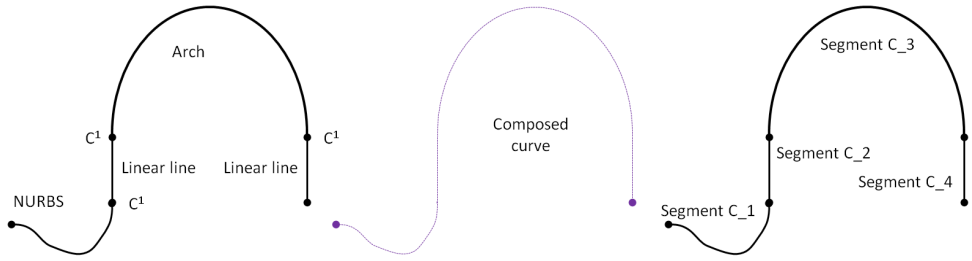


Figure 5.9: Composed edge construction

Delete Double Edges

Distinct edges which are actually non-disjoint are referred to as double edges. Non-disjoint is to be understand in a numerical sense, therefore a tolerance provided by the tolerance class is used to judge on equality in the routine.
Double edges are the result of modelling or translation errors when exchanging geometric models via exchange formats. Another source for that error are the healing routines themselves. The function {DeleteDoubleEdges(..)} is detecting double edges and healing them to construct a valid and consistent BREP model.
The algorithm works as sketched in the pseudo code below. This pseudo code gives a good impression how recursive algorithm design and possible tolerance adaption are realized in the code. Before the more expensive geometric checks are performed to identify two double edges several simpler test are included. Therefore the number of edges to be checked against each other is reduced. The geometric check itself also works in a similar manner. First an endpoint and midpoint check is performed to exclude cases where edges are obviously not equal. And only if this test succeeds a more expensive geometric check is called which is based on functions provided by Open CASCADE and computes the extrema of distances between curves.

```
1
2  program DeleteDoubleEdges(HealITWire* aWire)
3  {
4    // run over all edges in wire
5    loop(over all edges in wire)
6    {
7      // set Edge
8      Edge_i;
9
10     // test it against all edges in shapes
11     loop(over all edges in shape)
12     {
13       // set other Edge
14       Edge_o;
15
16       // check if not the same edge
17       if(Edge_i != Edge_o)
18       {
19         // check if other edge is not bounding the same region
20         // such a edge must be in the same wire and therefore distinct
21         // prior sorting heals double edges in one wire
22         if(Edge_o != bounding same region)
23         {
24           // other edge not of zero lenght and BBoxes intersection
25           if( Edge_o->lenght != 0 || BBox_intersect(Edge_i,Edge_o)))
26           {
27             // edges not with different master edges
28             if( !(different master edges))
29             {
30               // call geometric check routine
31               if(call Is_EdgesIdentical(Edge_i, Edge_o, Tolerance))
32                 //edges are identical
33                 return true;
34
35               // tolerance adaption (if enabled)
36               if(call Is_EdgesIdentical(Edge_i, Edge_o, Tolerance*↩
                     Adaption_Factor))
37                 call Adapat_Tolerances(Edge_i)
38             }
39           }
40         }
41       }
42     }
43   }
44
45 }
```

Code Region 5.5: Delete Double Edges

The algorithm to identify and heal double edges also includes the check of identical nodes. The endpoints might be distinct but non-disjoint according to the defined tolerance values. Therefore no topological endpoint test is included and like this the method is more efficient and flexible to heal two errors at the same time.
In practical examples many double edges pairs were detected where the accuracy differs significantly from one end to the other. Within the workflow these edges might not be recognized as double edges but treated as partially overlapping edges, resulting in edge decomposition. If this is not desired the tolerance adaption in combination with an optimized workflow described in

5.2 should be enabled.
This algorithm might change the geometry and topology of another region already visited earlier within the region loop since double edges are replaced by one common edge. This is one main reason to check the validity of all regions after the region healing processing in a post-processing step.

Delete Overlapping Edges

So-called overlapping wires hold one or several partially non-consistent edge intersections within, what differentiates this function from double edge routines. The sources for this error might be the same than for double edges.
This algorithm detects edges in a wire which have a partial overlapping with other distinct edges. This operation is called after the "delete double edges" operation although it would work also for totally overlapping and therefore double edges. The reason is that this operation is more expensive and is therefore performed most efficiently after double edges are deleted from the model.
The algorithm makes intensive use of bounding box checks to increase efficiency. In a first step a loop over all edges determines if there can be any non-consistent intersection with edges contained in another region. This procedure includes rather cheap geometric distance tests. By this procedure a subset of all regions in the model is obtained which will be used for the actual geometric operations. Like this the number of necessary mutual geometric test between edges is reduced significantly.
In the next step the algorithm performs only on the subset of non-consistent edges in the wire. Every edge is equipped with a map of regions which were detected to share non-consistent intersection with this edge. Only at this point a rather costly geometric operation determines the exact location and length of the intersection interval on this edge and on the corresponding edge in the other region. The edges are healed including an edge decomposition to ensure that intersections only appear along common edge entities.
The algorithm is sketched by the following pseudo-code:

```
1
2  // implementation in wire healing routines
3     // run outer loop until no possible overlapping edges are present ↩
          anymore
4    do
5    {
6      if(call DeleteOverlappingWires(..) = true)
7      {
8        // .. see above: reset neighboring relations ..
9      }
10
11   } while(overlapping edges possible)
12
13
14
15 program DeleteOverlappingWires(HealITWire* aWire)
16 {
17    // Find the subset of edges with non-consitent neighbors
18    // every edge holds a map of regions with non-consistent neighboring ↩
          relation
19    call FindNonConsistent_Edges(..);
20
21   // loop over subset of non-consistent edges
22   loop (over non-consistent edges)
```

```
23   {
24     // set edge
25     Edge_i
26     // check if non-consistent regions of edge are already visited
27     if( all regions in Edge_i->Map_ofNonConsistent_Regions = visited )
28     {
29       // set a non-consistent region
30       Region_i
31
32       // find a edge in wire of other region with overlapping
33       Edge_o = Find_OverlappingEdge(Region_i);
34
35       // decompose edges and merge common parts to form a consistent edge
36       if(Form_Consistent_Edges(Edge_i , Edge_o))
37       {
38         // if merging performed break and force outer loop to run again
39         flag run_outerLoop_again = true;
40         break;
41       }
42     }
43   }
44 }
```

Code Region 5.6: Delete Overlapping Edges

Since this algorithm is much more expensive than the double edge function and furthermore more prone to fail it is disabled in a first run in case an optimized workflow is enabled. Only if double edges are not likely to be present even after the tolerances were adapted locally the healing routines are called again with this function enabled.
The efficiency of this algorithm depends significantly on the size of the model, in particular on the edges to be analysed. The overall efficiency is increased by a simple procedure. A region called earlier in the region healing loop consists most likely of fewer bounding edges after the edge merging operation is performed. Therefore the algorithm makes use of the visited flags to increase efficiency. After for the subset of edges all the non-consistent region neighbours are given the algorithm checks if those regions were already visited within the healing loop. If this is not the case the operation is stopped for that particular edge and region. As soon as the code "reaches" the other region the same non-consistent intersection pair will be detected and healed. The difference is that the model size most likely is decreased by then.

5.3.3 Surface Healing Routines

Healing the face information of regions itself is still to be realized within the healer code. Healing routines for the faces mainly have to address intersection problems and non-watertight trimming curves. As sketched in section 2.2.6 a valid BREP model requires the absence of non-manifold points, which can appear due to self-intersections and intersections between surfaces.

Surface Intersections

Intersection problems are non-trivial and hard to detect in general. This is especially the case for trimmed NURBS descriptions. A stable algorithm would have to work on the following sources of errors and inconsistencies:

- Surface intersection patches due to partially overlapping faces: This error is a strict violation of validity conditions. In a floating point environment the exact intersection patch might be hard to detect.

- Unwanted gaps in the model between neighbouring faces: Gaps can appear due to trimming or modelling inaccuracies. Detection of gaps suffers from the ambiguous character. A tolerance value is required to decide if a gap is a hole or a defect. Gaps might be very small and hard to detect.

- Line intersections between surfaces: The intersection line has to be included in the model in form of a edge entity otherwise it is considered to be a defect violating validity conditions.

- Self-Intersections: They might appear due to the geometric shape description of faces in the BREP model. Detection and healing is very complicated in general.

Section 3.2.1 gives an theoretical overview on intersection problems and computational approaches. For parametric descriptions the intersection problem is non-trivial. In case of trimmed surfaces the situation is much worse. None of the necessary intersection healing operations is yet implemented in the code. If intersections are in the vicinity of the boundary they are healed within the Wire Healing routines and further Face Healing routines are not necessary.

5.4 Extrusion Volume Healing Routines and Algorithms

The TUM.GeoFrame framework provides a special volume description for shell-like structures. They can be modelled conveniently in $2\frac{1}{2}$ dimensions by extruding a reference shell in the direction of the shell thickness. The three different possibilities to construct these $2\frac{1}{2}$D volumes within TUM.GeoFrame are the following:

- Extrusion (sweeping) of a midsurface along a globally defined path (vector) with a defined thickness

- Extrusion (sweeping) of a midsurface along a path defined locally by the normal vectors on the surface

- Projection of the midsurface on a top- and bottom-shell. Top- and bottom-shell describe the volume's boundary.

At the current state there are no healing routines implemented which address the volumetric information directly. Therefore only the minor entities, namely regions in the shells, are passed to the region healing routines.
Volumetric healing routines are scope of future development. Useful algorithms might address the following sources for defects in $2\frac{1}{2}$D volumes:

- Check if projection and sweeping paths do not result in surface self-intersections.

- Check if projection and sweeping paths do not lead to intersections with other volumes in the model.

- Check if all necessary entities are defined, for example bottom and top shells for intersection volumes.

- Check if defined entities are geometrically meaningful. Normal extrusions from the mid-shell have to intersect with the top and bottom shells for intersection volumes.

The advantage of this volume type is the reduced dimensionality. Therefore intersections and connectivity is analysed on basis of regions within the shells. For example two regions forming the mid-shell have to be geometrically and topologically consistent. If this is guaranteed by region healing routines no further volumetric tests are necessary. Example Ex_07 in section 6.3.1 illustrates this approach.

5.5 BREP Volume Healing Routines and Algorithms

Classical BREP volumes consist of a closed boundary shells and might have internal shells describing holes. A shell itself is defined by a patch of regions. The geometric appearance of the volume is ruled by the geometric shape assigned to the faces of the regions. In the following section the BREP volume healing routines will be presented.

5.5.1 BREP Volume Healing Workflow

Besides the region healing routines there are also healing algorithms available addressing the volumetric information of the model itself. The validity of the volumes is checked within the healer code. The validity checks are mainly topological test to analyse if the graph containing edges and regions is topologically able to form a valid BREP volume. Additionally a geometric check should guarantee that internal shells forming holes geometrically are enclosed by outer shells.
The volume healing workflow contains a loop over all BREP volumes. The validity is checked on basis of a single volume. Invalid volumes are marked in the data structure. In a next step the algorithm analyses the whole model to find more valid BREP volumes if this functionality is enabled. In the post-processing step invalid volumes are deleted from the model. The single steps will be explained more in detail in the following. The overall volume healing workflow likes like:

```

programm Healer_main(Shape)
{
  ..

  // Start section BREP healing

    // preprocessing
    call BREP_Healing_Preprocess(..);

    // processing
    loop(over all BREP volumes in shape)
    {
    if(volume is to heal)
      call BREP_HealIT( aBREP, ..);
    }

    // find all valid volumes if enabled
    call Find_AllValid_BREPS(..);

    // postprocessing: delete invalid BREPS
    call BREP_Healing_Postprocess(..);

  //

  ..
} // end Healer_main
```

Code Region 5.7: Overall BREP Healing Workflow

The workflow for healing a single volume is the following:

```

programm BREP_HealIT(HealIT_BREP_Volume* aBREP)
{
```

```
4   // -- Shell Healing --
5
6     // Set Boundary Shell
7     call Healing_Shells(Boundary)
8
9     // loop over internal shells
10    loop(over all internal shells)
11    {
12      // heal internal shell
13      call Healing_Shells(Internal_Shell_i)
14    }
15
16  // -- Volume Healing --
17
18    // check if volume is valid
19    call Is_BREP_valid(aBREP);
20
21    // if not try to make it valid
22    if(!(aBREP -> valid))
23      if(call Validate_BREP(aBREP) -> fails)
24      {
25        // set flag that volume has to be deleted in postprocessing
26        aBREP -> invalid and validating fails
27        aBREP -> mark to be deleted
28      }
29
30    // if volume is valid check if inner shells are enclosed
31    if(aBREP -> valid)
32      call is_Internal_Shell_Enclosed(..);
33
34  }
```

Code Region 5.8: BREP Healing Workflow

The shell healing functions simply pass all the contained regions to the corresponding region healing routines.

5.5.2 BREP Volume Healing Routines

Goal of these routines is to check the validity of BREP volumes. Validity is to be understood with reference to the BREP modelling scheme described in section 2.2.6. The conditions can be summarized stating that the graph containing edges and regions should be able to form a closed volume. Closed in this sense is a topological requirement. Additionally there should not be any inconsistent intersection which refers to a geometric condition. Currently the geometric conditions are not considered in the code.
Invalid BREP volumes are passed to another routine which tries to validate the volume by either adding or deleting edges and regions to the graph. If the volume is still invalid after this routine it is marked to be deleted from the model. As a consequence of this approach all volumes will be either valid or invalid after looping over all of them. A third routine can be enabled to find all possible valid BREP volumes in the model. The detected volumes are checked against the present valid volumes in the model. Only detected volumes not equal to a valid volume already present in the model are added to it. The subroutines are described in the following sections

Check volume validity

The routine {Is_BREP_valid (aBREP ..)} is based on the theoretical backgrounds given in section 3.2.2. The graph of regions (cycles) and edges is analysed for fundamental cycles and connected subgraphs. The workflow to determine if a single BREP volume is topologically valid reads as follows:

```
1
2  programm Is_BREP_valid(HealIT_BREP_Volume* aBREP, ..)
3  {
4    // get set of edges and regions from the volume
5    Region_Map = aBREP->region_map;
6    Edge_Map = aBREP->edge_map;
7
8    // pass maps to a volume detection function
9    call BREP_VolumeDetection(Region_Map, Edge_Map)
10       // within this function the following steps are performed
11       // 1) set-up CE incidence matrix
12       // 2) compute Nullspace of CE matrix
13       // 3) set up maps of regions for every Nullspace solution vector
14   // interpret solution
15   int nValidVolumes = BREP_VolumeDetection -> nSolutions
16
17   case {
18
19       (nValidVolumes = 0){
20           // no valid volume
21           return false;
22           }
23
24       (nValidVolumes = 1){
25           // all regions contained in detected valid volume?
26           if(aBREP->regions = BREP_VolumeDetection->Volume(0)->regions)
27             return true;
28           else
29             return false;
30           }
31
32       (nValidVolumes > 1){
33           // only possible if volume contains internal shells
34           if(aBREP -> nInnerShell = 0)
35             return false;
36           else {
37
38             // every shell has to be investigated separately
39             call
40               BREP_VolumeDetection(BShell->Region_Map, BShell->Edge_Map)
41
42             for all Internal Shells do {
43               call
44               BREP_VolumeDetection(InShell_i->Reg_Map,InShell_i->Edge_Map)
45               }
46             }
47           }
48   }
49 }
```

Code Region 5.9: Check BREP validity

The function returns true if the volume forms a valid BREP volume. Otherwise the valid flag is set to false for this specific volume.

Validate invalid volumes

In a next step invalid volumes are tried to validate with help of this function {Validate_BREP (aBREP ..)}. Two major steps are included within this algorithm.

1. Detect and delete double regions in the model: Double regions refer to non-disjoint regions in the model which violate the validity conditions for BREP schemes. On the other hand these double regions will cause the function {Is_BREP_valid (aBREP ..)} to return false. The reason is that every double region includes one more solution in the volume detection algorithm. For example a simple volume with no inner holes will have as a result two Nullspace solution vectors if a double region is present.

2. Add missing regions to the model: This step is only performed if the volume is still invalid after step 1). Missing regions might be a reason why a volume is topologically not closed and therefore invalid.

The single steps are described in detail in the following.

Delete double regions All possible valid volumes are extracted from the solution vectors obtained from the edge-cycle incidence matrix. Valid is to be seen in a topologic manner. Therefore non-disjoint but distinct regions form topologically a valid volume consisting only of two regions. Only geometric test reveal their invalidity.
However in this function it is sufficient to detect these volumes consisting only of two regions within every shell of the BREP volume. One region is deleted from the regions map assigned to this specific shell. The decision which region to delete is arbitrary. Nevertheless the one region is deleted which is assigned to fewer or in the best case to no other volumes. Like that the consistency with neighbouring volumes should be preserved.
In the end the volume detection function is called again to analyse if the volume is valid after this steps.

Add missing regions If the volume is still invalid after deleting double regions this function is called. The basic idea is to extend the region and edge map among which valid volumes are detected. All neighbouring regions with their corresponding edges are added to the maps which share at least one common edge with regions of this volume. Example Ex_08 in section 6.4.1 illustrates this approach graphically.
From this extended search maps the algorithm tries to extract the desired information. The extended map might only contain one valid volume which is this volume and therefore reveals information about missing regions to validate it. On the other hand the result might also contain further volumes. The reason might be that entire neighbouring volumes were added to the map. Another reason might be that double regions where included into the search map.
The algorithm forms all valid volumes from the search map and refers to every shell in the volume one solution vector which has the largest region intersection set. Like that every shell can be validated by adding missing regions. This procedure fails if a region is simply missing in the model. In this case the volume is still invalid when this function terminates.

Find all valid volumes

There are two motivating aspects for the function {Find_AllValid_BREPS(..)}. First of all the user might need information if the geometric model contains more BREP volumes than actually

present in it or in the extreme case to find all volumes if only regions are present in the geometric model. Another reason might be that single existing volumes are invalid after the check functions described above. Therefore these volumes will be deleted in the post-processing step and valid volumes might be included by this function.
To understand why this function is necessary to validate invalid BREP volumes one has to consider again the hierarchical approach. The extended search map in the function {Validate_BREP (aBREP ..)} described previously can only contain regions which are part of shells in neighbouring volumes. If the missing regions are not assigned to any volume the validation procedure will fail. In this case this function finding all valid volumes might succeed. The steps are the following:

- Include all regions to the shape, also those not part of BREP volumes
- Heal regions not yet healed due to the prior region healing routines
- Extract all valid volumes from the graph of all regions and all edges in the model
- Check if detected volumes are identical to valid volumes in the model
- Add detected volumes not yet present in the model

Chapter 6

Examples and Assessment of Healing Algorithms and Routines

6.1 Assessment of Algorithms and Routines

Objective of this thesis was not only to implement a collection of healing routines and algorithms extending the existing TUM.GeoFrame framework, but also to test and assess them on a set of examples. The examples consist of self-constructed geometric models to check specific behaviour of the routines towards predefined model errors. On the other hand also "external" models and geometries were added to the testing set. These external models originate mainly from other research projects located at the chair CiE or from different research projects. During mesh generation inconsistencies or errors became apparent motivating to pass these models to the healer. Furthermore many of the external models originally motivated the development of a healer framework and several specific algorithms.
The infinite sources of model defects cannot be identified or overlooked via a set of self-defined geometric models. Especially the errors introduced by the user itself during modelling are difficult to foresee. Therefore testing healing routines against a wide range of practical examples contributes significantly to the robustness of a healing code. The scope of this thesis was to form a basis for further development of the code rather than to provide a finished healer project. For that purpose the examples which will be presented in the following are a start to evaluate the performance of the implemented routines at their current state. In a next step it will be necessary to test the routines against a far wider range of examples.
This chapter is divided in three sections with reference to the three main groups of entities selectable for healing. The first section will illustrate healing routines applied on regions (surfaces). The second section will show the relation of healing routines for regions contributing to extrusion or $2\frac{1}{2}$ D volume models, which are well-suited for shell-like structures. The last section will demonstrate the performance of the code operating on classical BREP volume models.
As mentioned in section 5.2 the implementation is organized recursively and makes use of the hierarchical decomposition of spatial objects in the BREP modelling scheme. Therefore during volume healing routines regions forming shells bounding themselves the volume are passed to the region healing routines. The section on BREP volume healing will therefore concentrate on the volume specific functionalities of the routines.

6.2 Region Healing Examples

This section covers healing routines applied on regions. At this point the notation "region" is to be clarified. A surface patch can consist of several regions. A region is the smallest complex

bounded by exactly one boundary wire and possibly consisting of internal wires for holes or lines.
The region healing routines mainly address the following errors and inconsistencies in a model:

- Merging non-disjoint nodes
- Merging double edges (non-disjoint edges) which are identical or partially overlapping
- Sorting Edges in the wires, which are bounding regions holes or form inner lines
- Deleting spurious edges
- Merging short edge segments to longer continuous edges

Most of the defects listed above are to be healed since they violate the conditions of validity for a consistent BREP model with a range of undesired consequences as illustrated in the upcoming sections. Short edge segments in contrast to that do not violate validity conditions and healing routines therefore address aspects of efficiency and further unwanted effects of the "short edge problem".

6.2.1 Ex_01 Region Healing: Simple Surface Patch

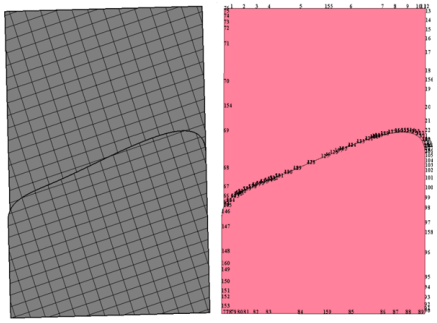


Figure 6.1: Rhino and IGES representation of the example's geometric model

This geometric model is a self-constructed example to test the specific performance of the code. The model consists of two regions forming a surface patch sharing one common continuous boundary edge. The geometry was modelled in Rhinoceros (Rhino) by means of trimmed NURBS surfaces using standard modelling operations in the software. Picture 6.1 sketches the Rhino model illustrating also the surface parameterizations. The proposed standard parameterization was applied. The user influence is limited. Only the number of knot intervals can be modified.
Picture 6.1 also shows the same structure after exporting it from Rhino into the neutral file format IGES and importing it onto the TUM.GeoFrame framework. All edges of the model are labelled to visualize that the region boundary wires consist of many short edge segments. In Rhino this boundary decomposition is not detectable. On the first sight visually only one common edge is shared by both regions with the following properties:

- Degree = 3
- Non-rational (all weight = 1)
- 5 control points
- Non-uniformly distributed knot intervals
- Open curve

The IGES file format is a simple ASCII file (if not compressed) which can be manually analysed and interpreted. The common curve can be detected in the file as well with the following attributes:

- Degree = 3
- Non-rational
- 12 control points
- Non-uniformly distributed knot intervals
- Open curve
- Multiplicity at endpoint = 4
- Multiplicity at every inner knot = 2

After importing the model into the TUM.GeoFrame framework we can still find this curve in the model with the following properties:

- Degree = 3
- Non-rational
- 5 control points
- Non-uniformly distributed knot intervals
- Open curve
- Multiplicity at endpoint = 4
- Multiplicity at every inner knot = 1

The problem is that this edge is actually not used as boundary of the trimmed surfaces and is also incapable do be used like that, what becomes obvious after a more detailed look on the geometry. The left starting point is not part of the upper surface domain as shown in figure 6.2. So even within Rhino a different trimming curve must be used. The real trimming curve nevertheless is only visualized after enabling a special option in Rhino. Picture 6.2 also illustrates the actual trimming curve. This curve is present in the IGES file with the following properties:

- Degree = 3
- Non-rational
- 97 control points
- Uniformly distributed knot intervals
- Multiplicity at endpoint = 4
- Multiplicity at every inner knot = 2

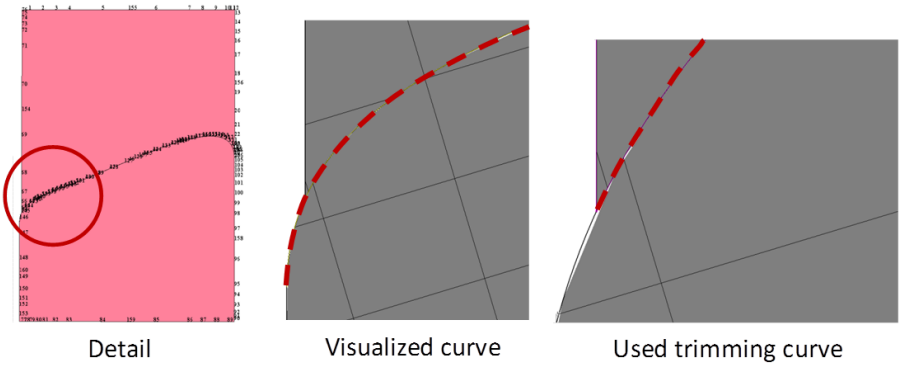


Figure 6.2: Rhino Modeller: detail on trimming curve

Within the TUM.GeoFrame framework however even that curve is not used to trim the two regions at the common intersection. Both faces are trimmed by a set of different curves, in particular by many short edge segments.
The comparison of the model description within the different standards and systems emphasizes again the influence of data exchange on the model quality. Depending on the quality of translators, half-translators and exchange standards the geometry and topology of a model might be changed slightly. The consequences for applications and routines in the downstream workflow nevertheless can be significant and will become obvious in the following. In general one can realize that modelling software like Rhino is rather influenced by rendering aspects than by the mathematical consequences of shape representations.
For the model given in this example the problem seems to arise within the half-translator provided by Open CASCADE when reading the IGES file. The consequences are dramatic since the model loses its consistency after importing it into TUM.GeoFrame. The neighbouring regions do not share common boundary edges anymore and therefore also a mesh for example will not be able to represent the connectivity of those two surfaces in means of common global degrees of freedom.
Two different quadrilateral meshes (Quad meshes) resulting from the surface meshing algorithm DoMesh are shown in picture 6.3. The difference in the mesh settings is the defined mesh density h. The mesh density controls the size of the resulting Quad elements. As we see even a mesh density of double magnitude does not influence the mesh appearance significantly. The reason lies in the meshing algorithm which enforces all boundary edges to be part of the boundary mesh elements. Since the edge length is much smaller than the mesh density the mesh is completely ruled by the short edges on the boundary, resulting in a mesh refinement according to the edge length. In areas of high curvature this mesh refinement is desirable. Nevertheless the problem is that the user cannot control the mesh refinement on such a model. Furthermore there are areas like the corner of the regions where the mesh is refined without any geometrical necessity.

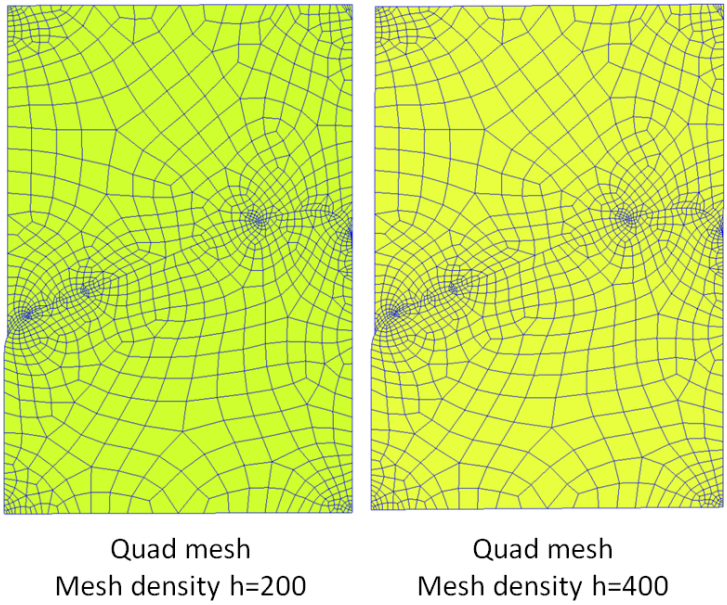


Figure 6.3: Surface Quad meshes on the original model

The enforced mesh refinement has an influence on efficiency of downstream applications. More critical is the fact that the inconsistency of the model results in an inconsistent mesh. Picture 6.4 depicts a detailed look on the boundary between the two surfaces. The mesh looks visually consistent (meaning neighbouring elements have no distinct nodes or edges which are actually non-disjoint). After labelling the nodes it becomes obvious that on the common boundary the mesh is not consistent and the structure might fall apart in a FEM simulation.

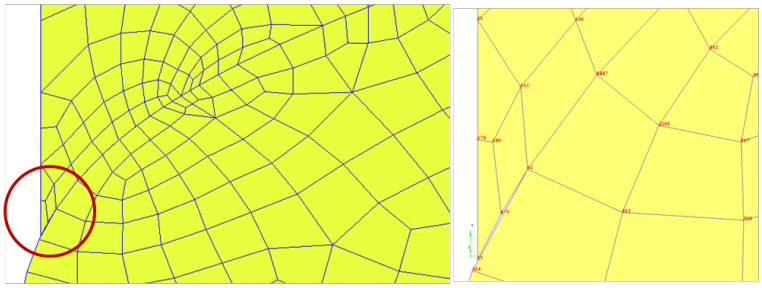


Figure 6.4: Inconsistencies in the surface mesh

Picture 6.5 shows the structure after application of healing routines. All functionalities were enabled and no further user intersection was required. The healer merges the short edges and furthermore produces a consistent model where both surfaces share one common edge. The common edge has the following properties:

- Degree = 10
- Number of knots = 65
- Number of poles 580

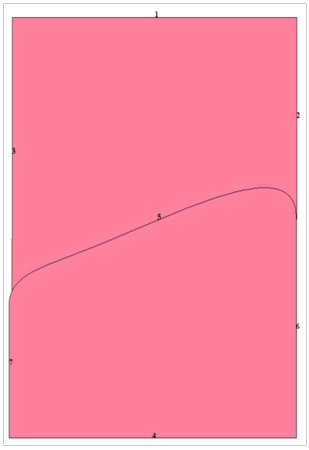


Figure 6.5: Geometric model after application of healing routines

We recognize that the degree of the curve increased significantly. There might be undesired side effects due to that representation. The mesh however benefits from the properties of the healed structure. Picture 6.6 visualizes different Quad meshes on the healed structure, among them a linear mesh with the same mesh density as assigned on the unhealed structure. In case of a linear element shape the mesh is clearly not able to follow the curved edge for the chosen mesh density. A remedy is to reduce the mesh size in total or allow a non-linear mesh shape, also shown in figure 6.6. On the healed model the mesh density can be modified locally at locations where needed according to the geometric properties of the structure. Picture 6.6 sketches different settings for local mesh refinements to enable a better approximation of the curved edge.

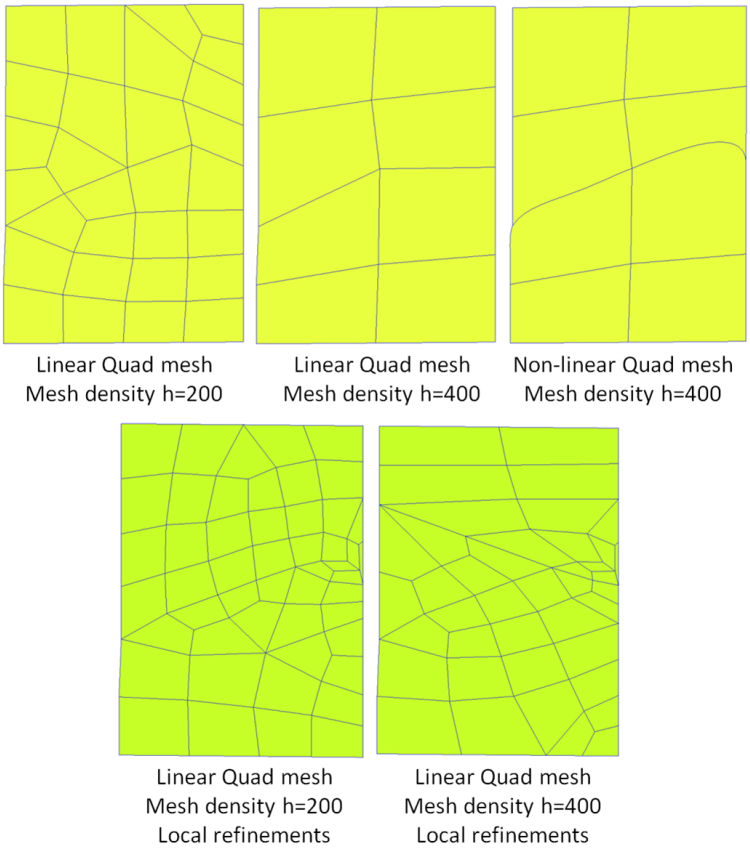


Figure 6.6: Surface Quad meshes on the healed model

6.2.2 Ex_02 Region Healing: Simple Surface Patch

The second example consists of two plane regions and is self-constructed in Rhino and exchanged via the IGES file format. In contrast to example Ex_01 there are two intersection segments along their boundary. As in the example before we can detect a highly decomposed boundary. A closer look on the common boundary reveals that the model is not consistent.

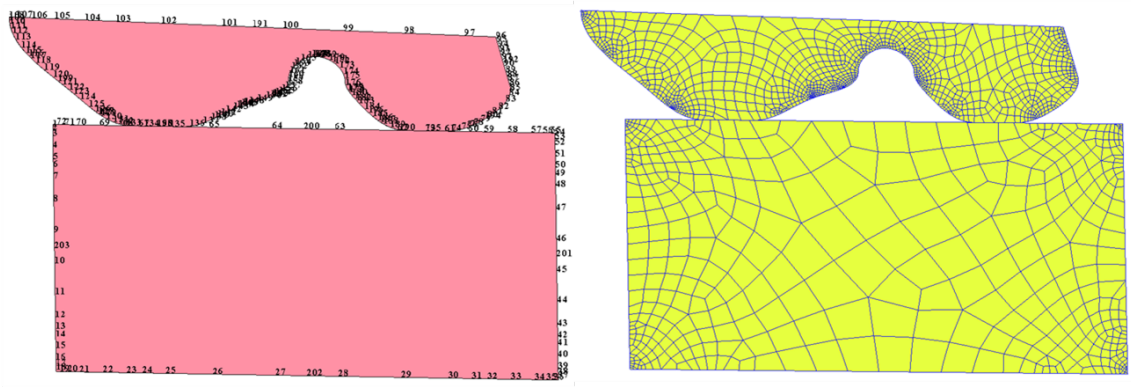


Figure 6.7: Original geometric model and surface Quad mesh on model

By observation we can recognize that the edges along the upper boundary of the lower region as well as the edges along the lower boundary of the upper region, where the intersection is present,

can be merged due to reasons of continuity. The example was constructed to demonstrate the performance of the code detecting and healing two (and in general various) overlapping segments along one edge by running in a loop until all locations of inconsistent intersections are healed. Picture 6.7 shows a surface Quad mesh on the original model and we can detect again the problematic mesh refinements without geometrical background. The inconsistency of the mesh along the common boundaries is far more obvious in this example. Picture 6.8 illustrates in contrast to that the healed model and its non-linear Quad mesh.

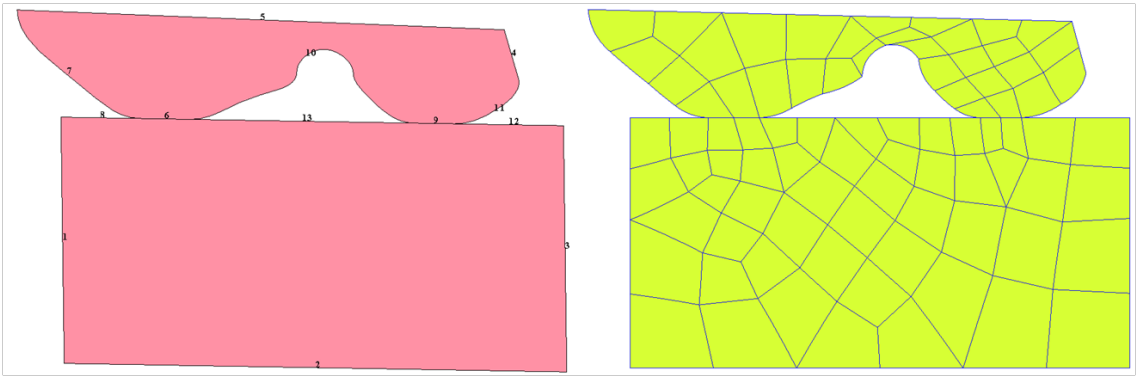


Figure 6.8: Healed geometric model and surface Quad mesh on healed model

The healing routine succeeded detecting and healing two (and in general several) inconsistent intersections along the boundary. The consistency of the model and resulting mesh is guaranteed after applying the healing routines.

6.2.3 Ex_03 Region Healing: Complex Trimmed Surface Shape - Violin

This example demonstrates the performance of the code with respect to a far more complicated surface shape illustrated in picture 6.9. The model represents the top plate of a violin corpus. The NURBS surface is trimmed on its boundary and internally to model the so called f-holes. The shape is complex by means of the boundary shape and the surface curvature itself. The model was created in the scope of a student's project to investigate the acoustical behaviour of a violin with help of numerical methods [21]. In this sense it can be seen as a realistic model used for engineering applications.

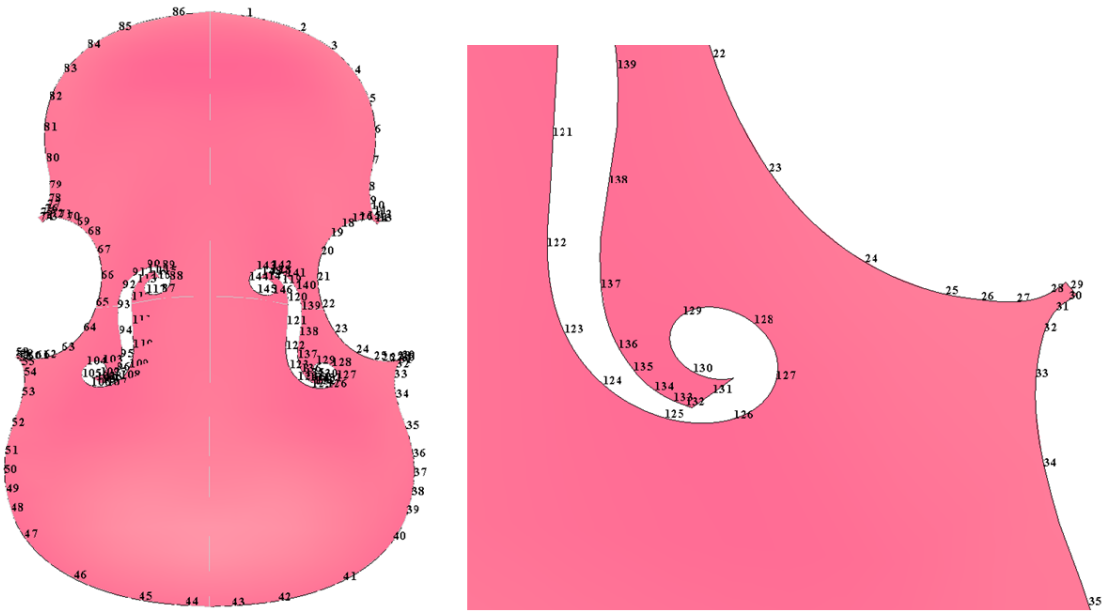


Figure 6.9: Original geometric model of violin's top plate and detail on f-holes

The model was created in Rhino and IGES was chosen as exchange format. The drawbacks of this configuration were already evident in the two previous examples. The model suffers from short edge segments along boundary wires as well as internal wires. Picture 6.9 also shows a detailed look on the edges forming the internal wire to describe the f-holes.
This geometric model is only a reduced version of the whole corpus as it was modelled and meshed within the student's project as seen in [21]. The model was created by the author of this thesis therefore insight on consequences of model inconsistencies on the meshing algorithm were present during the modelling procedure. Nevertheless it turned out that it was impossible to use the model directly for meshing and analysis routines without prior manipulations on the geometric model. This fact emphasizes again the necessity of robust and powerful healing routines, since the absence of errors cannot be guaranteed for complex models.
In the following the internal f-holes are not regarded in the model and therefore deleted from the shape representation to emphasize a specific aspect of healing routines. The top plate of the violin is meshed with a high order surface mesh depicted in figure 6.10. A mesh density of $h = 8$ was chosen. A detail on the small ears at the so-called C-Bout in picture 6.10 reveals the presence of enforced mesh refinement within the mesh. In general nevertheless the mesh quality is acceptable and not significantly affected by refinement problems.

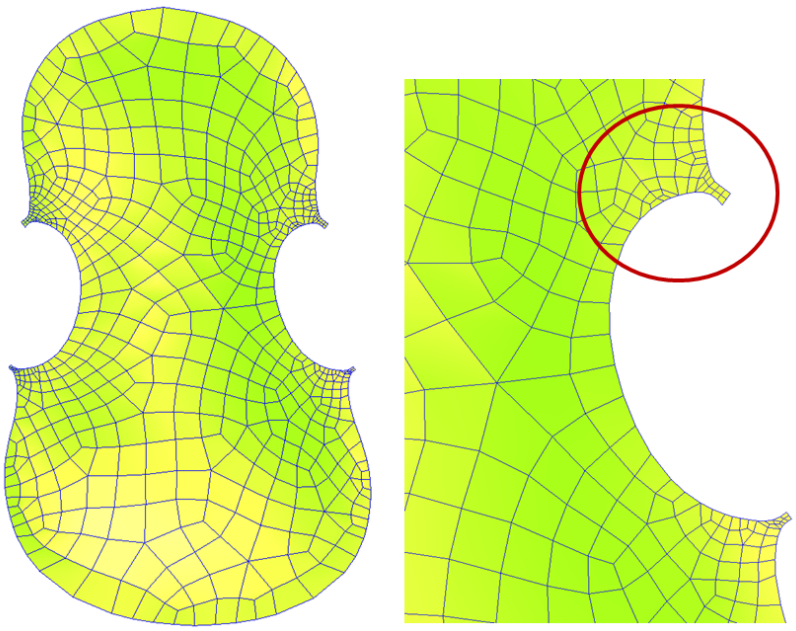


Figure 6.10: Surface Quad mesh on original model and detail of mesh at the ears

The mesh with the same settings but constructed on basis of the healed model is shown in picture 6.11. Visually the difference of the two meshes is not significant. Only a focus on the ears and C-Bout reveals little deviations due to merging of short edges.

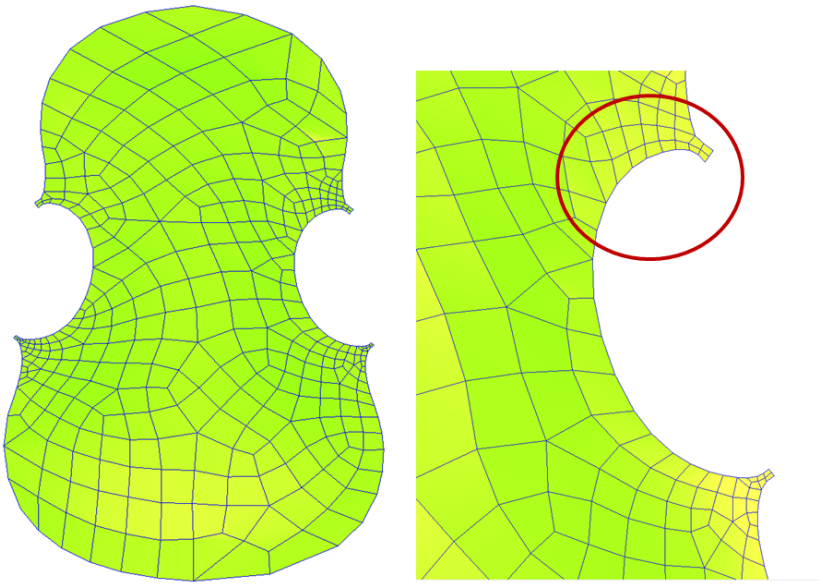


Figure 6.11: Surface Quad mesh on healed model and detail of mesh at the ears

This model is characterised by its strong curvatures along the boundary as well as on the surface itself, resulting in the necessity of relatively small element sizes to describe the geometry with the desired and acceptable accuracy. We can therefore conclude that here the short edge problematic is of far less influence on the mesh quality. There are nevertheless more reasons to prefer boundaries without edge decomposition. The model for the whole corpus for example requires connectivity of the side walls to the top plate as described in [45] to guarantee a

consistent mesh. The boundaries of the sidewalls have to be included in the top plate region in form of internal lines. This step has to be done within TUM.GeoFrame manually. Handling this modelling step is much easier on the healed model since significantly fewer edges are present. Therefore the short edge problem is also a question if storage and handling of model data and not only a pure problem for the applied meshing algorithms. Picture 6.12 shows a Quad mesh on the whole corpus.

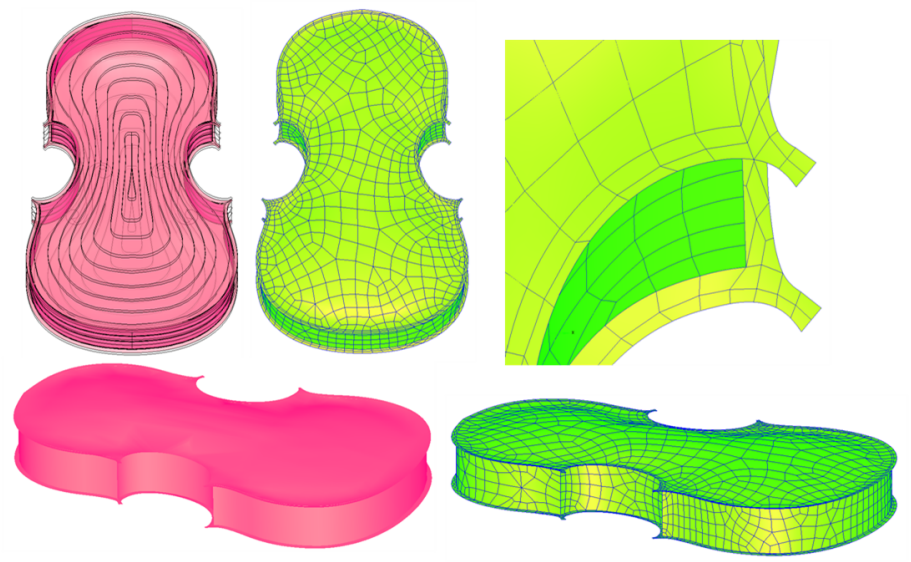


Figure 6.12: Surface Quad mesh on the entire violin corpus

6.2.4 Ex_04 Region Healing: Complex Surface Shape - wind turbine

The model analysed in this example as illustrated in figure 6.13 represents a wind turbine with three blades and was modelled in scope of a bachelor's thesis at the chair CiE [13]. Purpose of the model was to test interface meshing operations of the TUM.GeoFrame mesh generator (TUM.Meshing Engine). The results are published in [45].

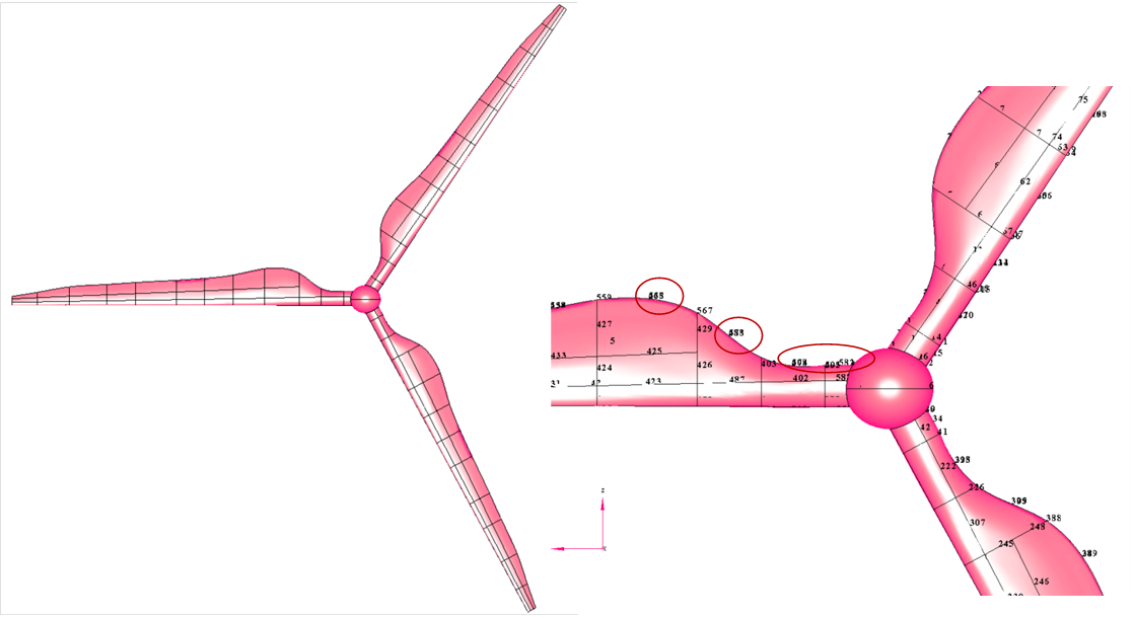


Figure 6.13: Original geometric model of a wind turbine

The model was created in Rhino and exchanged via the neutral file formats IGES and STEP. Geometric inconsistencies similar to the ones detected in the first two examples could be encountered. The STEP exchange format turned out to suffer far less from the edge decomposition problem. The problem with IGES files is arising during model import. The half-translator for STEP files provided by the external library Open CASCADE seems to work better when reading STEP files. In contrast to IGES however the STEP format turned out to include spurious edges in the model under certain circumstances. These edges do not have a physical length and might not even be topologically connected to the region's wire although included in the wire description.
The healing routines succeeded erasing double and spurious edges and were able to provide a consistent model description. The accuracy of the original model is quite good and the healer had no problems to detect non-disjoint but distinct edges within only one healing run. Picture 6.14 shows the final structure after the healing routines were applied.

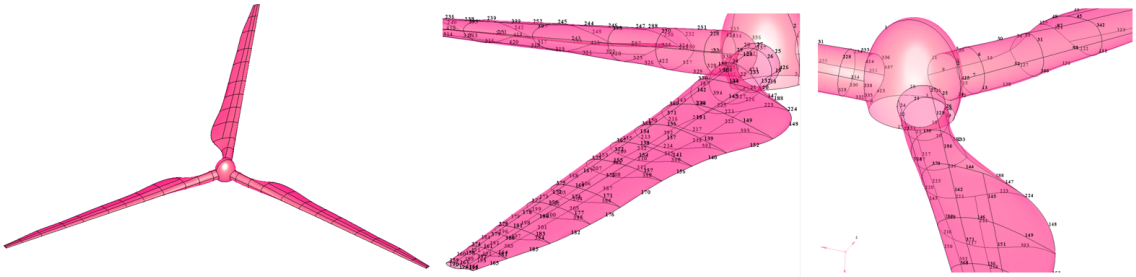


Figure 6.14: Healed geometric model of the wind turbine after region healing routines

6.2.5 Ex_05 Region Healing: Complex Surface Shape - ship model

The geometric model in this example was created in the scope of research projects at Technische Universität Hamburg-Harburg (TUHH) and represents a container ship corpus. Rhino was used as modelling software. After exchange via IGES standard the original model of the ship looks as

depicted in figure 6.15. The TUM.GeoFrame mesh generator (TUM.Meshing Engine) was used to discretize the model using high order elements for further analysis routines. As the picture indicates the model suffers again from heavy edge decomposition along the seams of the surface patches describing the sidewalls of the sip. Very problematic is also the inconsistency of the model between the seams and the stiffeners arranged perpendicular to the walls. The accuracy is rather poor within the model. The reason might be a certain geometric operations applied in Rhino or introduced by the user. In so far the model is a good example for the healer to be tested on its robustness towards poor accuracy in geometric models.

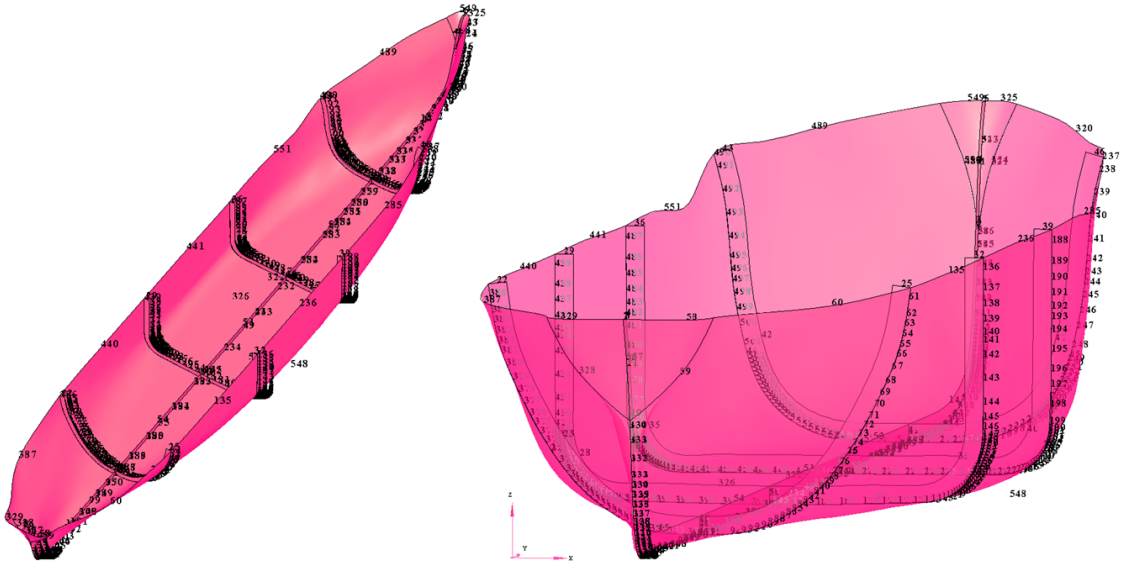


Figure 6.15: Original geometric model of a container ship

More problematic than a global poor accuracy is the fact that the accuracy varies significantly throughout the model. The walls meeting at the seams have a high accuracy between each other. Along the common boundaries consistency is even partially given although the wire is highly decomposed. Within a wire along this intersection often one or two short edges are shared by both regions challenging the healing routines and motivated finally the iterative healing approach implemented in the code.
The very poor accuracy between the walls and the stiffeners and at the front tip of the ship's bow was by far the biggest problem of the model. At the tip triangular regions meet and the accuracy is significantly low at this point as it can be seen in picture 6.16.

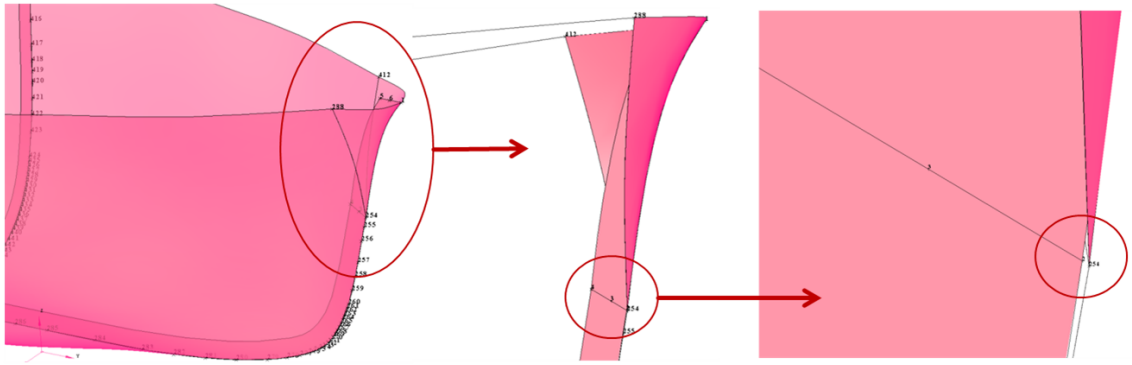


Figure 6.16: Tip of the original geometric ship model with poor accuracy

The meshing routines on the model without prior healing failed for the triangular regions at the tip due to the presence of spurious edges not belonging at all to those regions neither topologically nor geometrically.
Picture 6.17 visualizes the mesh on the rest of the structure with different mesh sizes. We can visually recognize the problem of enforced mesh refinements along the seams due to the short boundary edges.

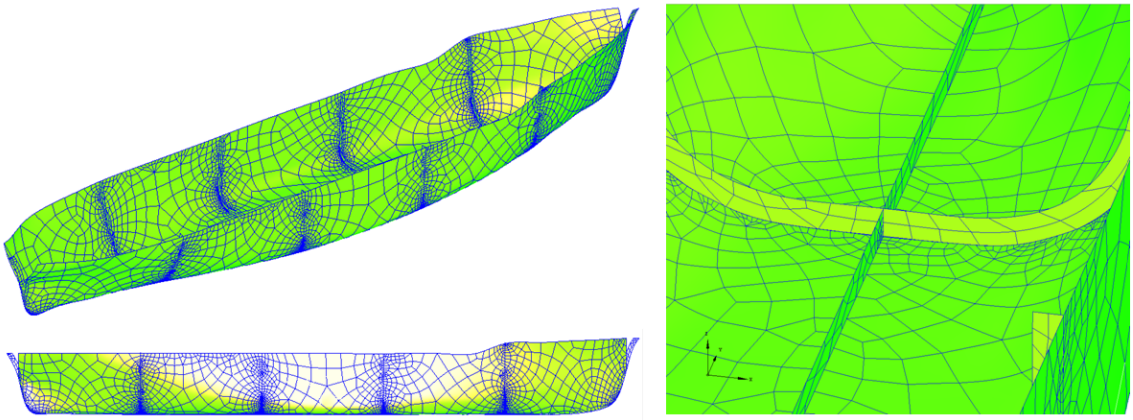


Figure 6.17: Surface Quad mesh on original geometric ship model

The model was passed to the healer to remove short and spurious edges and produce a consistent model. This model is well-suited to make use of adaptive tolerances and the optimized workflow functionality since the accuracy varies significantly in the model and therefore the optimal tolerance values in a local sense have to be determined iteratively. The healing routines contribute to this iterative search by adapting the tolerances as described in section 5.2.
Even with tolerance adaption in an optimized workflow the healer had to be called repeatedly on the model due to the very poor accuracy at the ship's nose. Either the tolerance had to be manually reduced locally at the tip or another approach was to correct this local error manually. However these kind of local errors are easy to detect since the healer performed robust on most of the other problematic points and can be treated manually.
In picture 6.18 the final healed geometric model with a minimal number of edges is shown. The resulting high order meshes on the healed structure are illustrated in picture 6.19. The healed model can be discretized with high order element shapes by a small number of elements. The number of elements significantly influences the computation time within techniques like pFEM. A higher order for the element shape increases the degrees of freedom. Therefore few elementes are desirable. The enforced mesh refinements unnecessarily blow up the problem size.

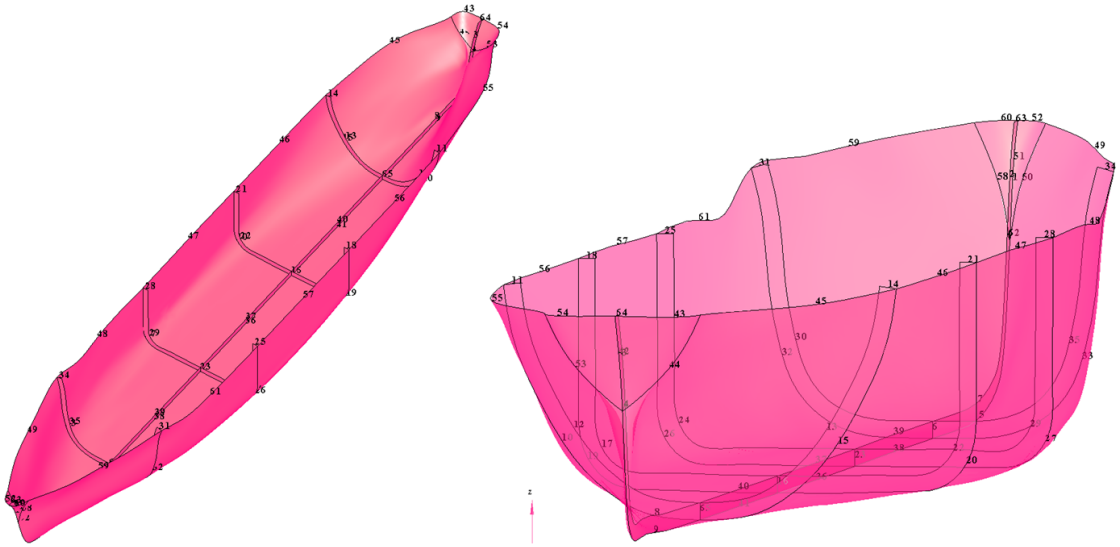


Figure 6.18: Healed geometric ship model

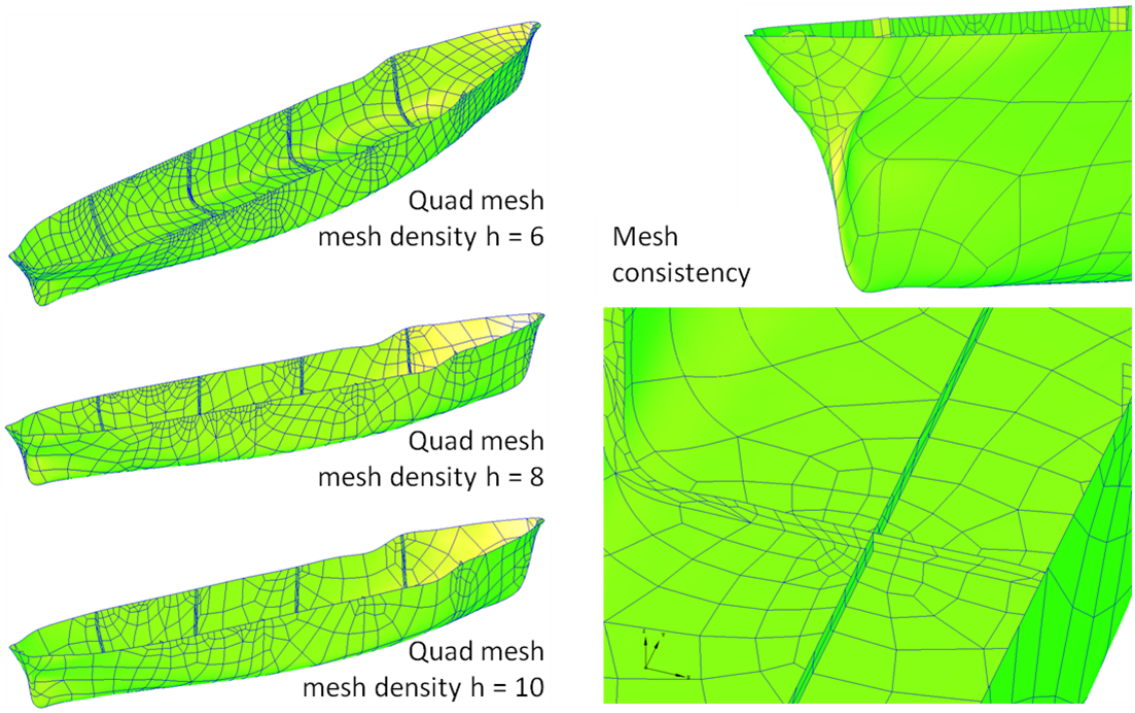


Figure 6.19: Surface Quad meshes on healed geometric ship model

6.2.6 Ex_06 Region Healing: Wire sorting

This example wants to demonstrate the performance of the edge sorting algorithm in the presence of dangling edges. Picture 6.20 sketches a region and its bounding wire. Edge number 4 is a so-called dangling edge. As it was mentioned in section 5.3.2 that simpler sorting algorithms looping over the edge vector and checking the endpoint connectivity will fail since an edge with two valid neighbours is present. This bifurcation point is not considered in the simpler sorting algorithm. Therefore a more sophisticated algorithm is requested. This algorithm is based on

the kernel of the Node-Edge incidence matrix. The theoretical background is given in section 3.2.2.

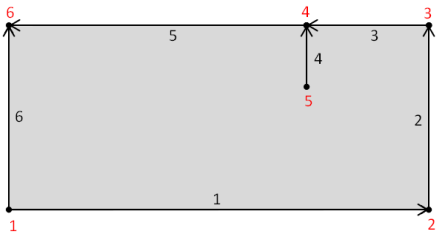


Figure 6.20: Region model with dangling edge

First an edge vector for the bounding wire is given in the following form:

{edge_1, edge_2, edge_3, edge_4, edge_5, edge_6 }

The Node-Edge incidence matrix reads:

$$B(G) = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \tag{6.1}$$

The kernel of this matrix is given as follows:

$$Null(B(G)) = c * \begin{bmatrix} -1 \\ -1 \\ -1 \\ 0 \\ -1 \\ 1 \end{bmatrix} \tag{6.2}$$

Clearly the only possible fundamental cycle { edge_1, edge_2, edge_3, edge_5, edge_6 } is detected. The dangling edge is simply deleted from the wire. The solution of the transpose of this matrix reads:

$$Null(\hat{B}(G)) = c * \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{6.3}$$

This solution indicates that actually all edges are connected with each other forming a connected graph. The information obtained from both solutions in combination can be used to delete edge 4 from the boundary wire and include it as an internal line into the description of the region. This approach is guaranteeing a valid representation without losing edge 4 in the model.
Let us consider a modified model sketched in picture 6.21 where edge 4 is simply cloned and added as edge number 7 to the wire description in the following form:

{edge_1, edge_2, edge_3, edge_4, edge_5, edge_6, edge_7 }

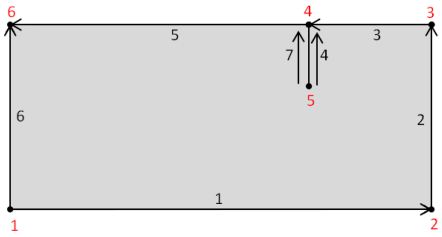


Figure 6.21: Modified region model with dangling edge

The edge sequence is not ordered as we can recognize. The Node-Edge incidence matrix reads:

$$B(G) = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{6.4}$$

The Nullspace is spanned by two bases and given in the following form:

$$Null(B(G)) = c_1 * \begin{bmatrix} -1 \\ -1 \\ -1 \\ 0 \\ -1 \\ 1 \\ 0 \end{bmatrix} + c_2 * \begin{bmatrix} 0 \\ 0 \\ 0 \\ -1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{6.5}$$

Two fundamental cycles are detected in the graph. The solution of the transpose of this matrix reads:

$$Null(\hat{B}(G)) = c * \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \tag{6.6}$$

From the Nullspace of $B\hat{(}G)$ we can conclude that all edges are connected. From the union of both Nullspace vectors on B, N_1 and N_2: $-1 * (N_1 + N_2)$ we obtain one valid bounding wire including all edges. This representation would however include two non-disjoint edges and therefore violate the validity conditions. Moreover this representation is incompatible with the downstream meshing algorithm for example.
To conclude this example it was illustrated that the kernel of the graph determined by edges and nodes can be used to reveal information about cycles in the structure. In this example it can be used to find a valid bounding wire and detect dangling edges. Dangling edges can be deleted from the bounding wire and treated as internal lines. A mesh on such a representation of the model is given in picture 6.22.

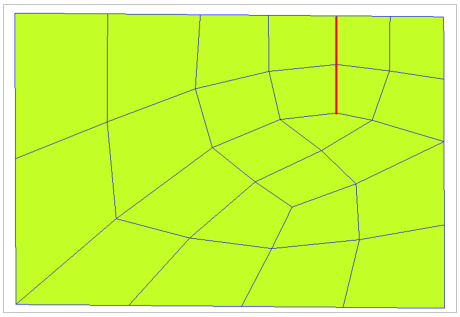


Figure 6.22: Mesh on a model with a dangling edge

6.3 Extrusion Volume Healing Examples

Due to the recursive implementation concept described in section 5.2 healing routines for extrusion volumes or $2\frac{1}{2}$ D volumes include also region healing routines. The region healing routines apply to all regions within the shells forming this type of volume. $2\frac{1}{2}$ D volumes are dimensionally reduced volumetric models suitable for shell-like structures. The volumes are described by means of their mid-surface shells or in case of extrusion volumes by means of a top and bottom surface shells. The extrusion volume routines at the current state do not provide any specific algorithms or operations addressing errors or inconsistencies related directly to the volumetric information. So far only the regions in the shells are passed to the region healing routines. Routines related directly to the volumetric information of extrusion volumes are part of further development covered in section 5.4 and 7.2.
This section demonstrates the influence of errors or inconsistencies related to regions within the shells on the $2\frac{1}{2}$ D volumetric description. Since the volumetric information of $2\frac{1}{2}$ volumes is only provided by shells, region healing plays an overproportionally important role for the volumetric model.

6.3.1 Ex_07 Extrusion Volume Healing

The example in this section is based on a geometric model illustrated in picture 6.23 which consists of three shells. It is meant to form a $2\frac{1}{2}$ D extrusion volume. The shell in the middle of the three shells illustrated is used as a reference shell with no specific geometric meaning. This shell is rather used within the meshing algorithm. The volume itself is described by an upper surface shell and a lower surface shell bounding the solid. Sidewalls are not necessary to model in this case. They are given by the connection of the boundary wires of the upper and lower shell.

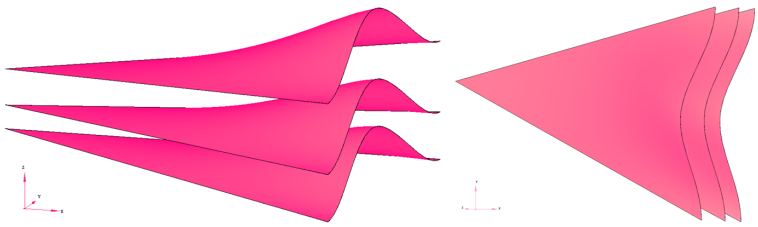


Figure 6.23: Geometric model describing an extrusion volume

The dimensional reduction of the model is a big modelling advantage in particular when describing thin shell-like structures. This advantage is also exploited in meshing algorithms for

example. The volume can be discretized with Hexahedron elements. In a first step the mid-surface or reference surface (since it does not necessarily have to be in the middle of the volume or even within the solid at all) is discretized with a surface Quad mesh. In a second step the surface mesh is extruded until intersecting with the upper and lower shell.
In the model in this example every shell contains exactly one triangular region. The boundaries of the regions are decomposed into short edges shown in picture 6.24. This problem was already covered in the previous section about region healing routines. On the healed model the boundary of every region can sufficiently be described by only 3 boundary edges demonstrated in picture 6.25.

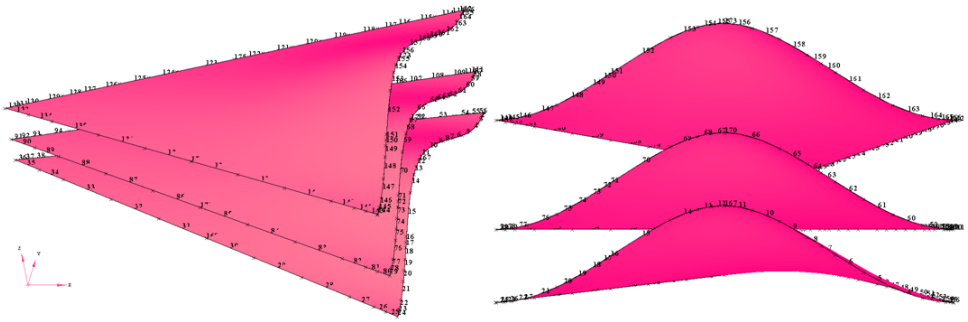


Figure 6.24: Labelled geometric model with boundary edge decomposition

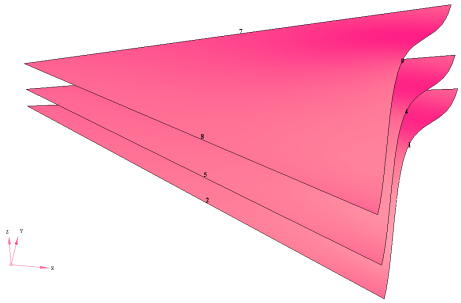


Figure 6.25: Healed geometric model of an extrusion volume

This example wants to stress out the negative consequences of the edge decomposition and enforced mesh refinement on the volumetric mesh elements. Picture 6.26 illustrates different surface Quad meshes on the reference shell, varying in their meshing settings. The mesh clearly is completely determined by the boundary edges and different settings concerning the mesh density have only little influence. Picture 6.26 also shows two meshes with a higher propagation factor. The propagation factor determines the gradient in the element size and can be used to enforce larger elements in the interior of the regions. As a result elements in vicinity of the boundary might have a critical aspect ratio illustrated in the picture. On such a mesh the advantages of high order meshing cannot be exploited sketched also in picture 6.26 since the problem of critical aspect ratios is not solved by high order shapes.

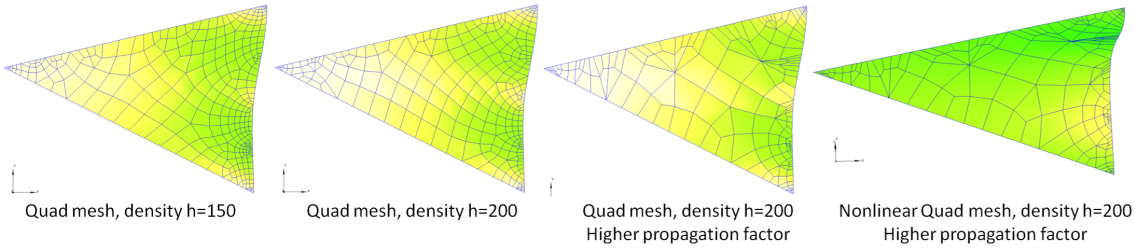


Figure 6.26: Surface Quad mesh on reference shell of the original model

The resulting volume mesh with Hexahedra elements can be seen in picture 6.27. In particular

the view from behind on the cross section reveals the occurring problems due to the element sizes. The enforced mesh refinement results in small element extensions compared to the model's thickness. The extrusion operation results in critical element ratios also in normal direction to the surface. The lower side of several elements degenerate to a point or line. These elements are not to be used in a FE analysis.

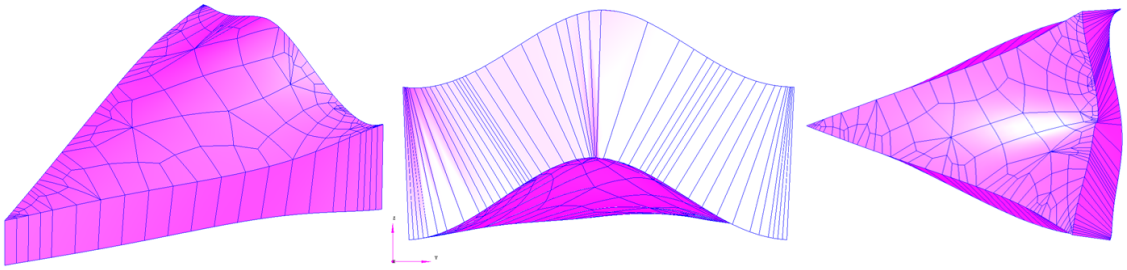


Figure 6.27: Volumetric Hex mesh on original model

The difference in the mesh quality between the original model and the healed structure becomes apparent as illustrated in picture 6.28. Linear as well as non-linear surface and volumetric meshes are sketched. The advantage of high order elements can be exploited in an optimal way on the healed model also demonstrated in picture 6.28.

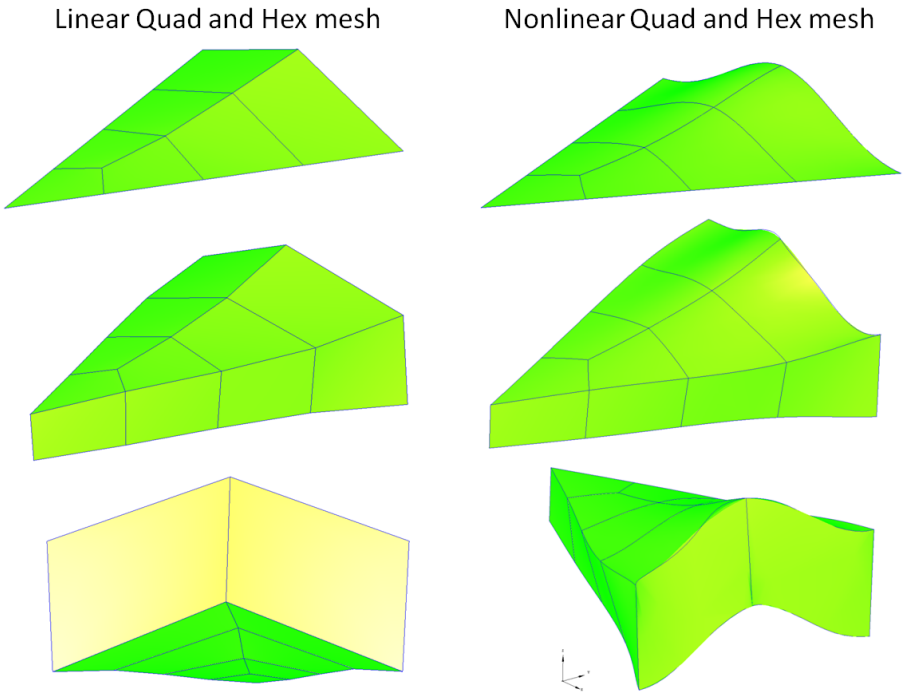


Figure 6.28: Surface and volumetric meshes on healed model

The situation is more critical after a modification of the geometric model as shown in picture 6.29. The shells of the structure are composed in this case of two regions. Picture 6.30 visualizes the resulting surface Quad mesh on the mid-shell. The enforced mesh refinement is now also present at the common interface of the surface patch. Moreover the mesh losses its consistency which becomes visible having a detailed look on the mesh presented in picture 6.30.

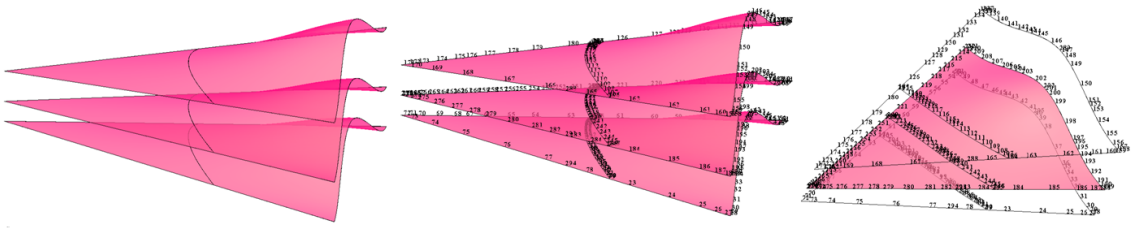


Figure 6.29: Modified model with two regions per shell in its original form

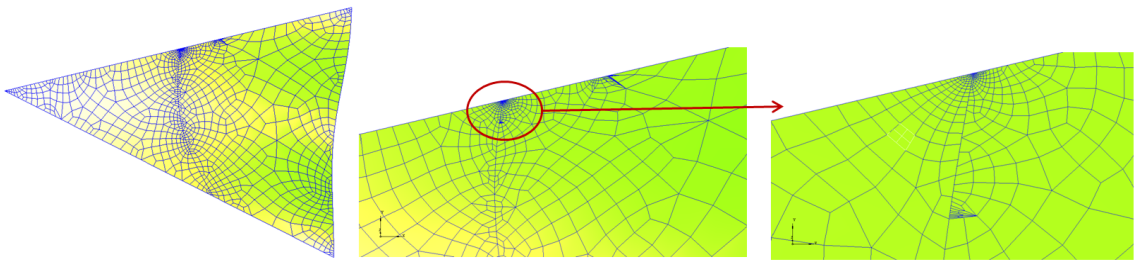


Figure 6.30: Surface Quad mesh on original modified extr. volume model

Picture 6.31 shows the resulting volumetric Hex mesh on the original model. As before elements collapse into lines or points due to their enforced small size in transversal direction compared to the thickness.

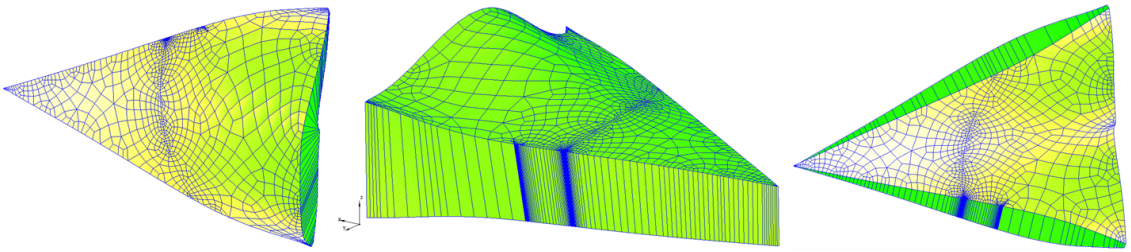


Figure 6.31: Volumetric Hex mesh on original modified extr. volume model

In contrast to the original model and mesh picture 6.32 illustrates the healed structure and the corresponding surface Quad mesh and volumetric Hex mesh. Only the healed model might be used for further analysis steps due to the defective and critical ratios of elements in the mesh on the unhealed structure.

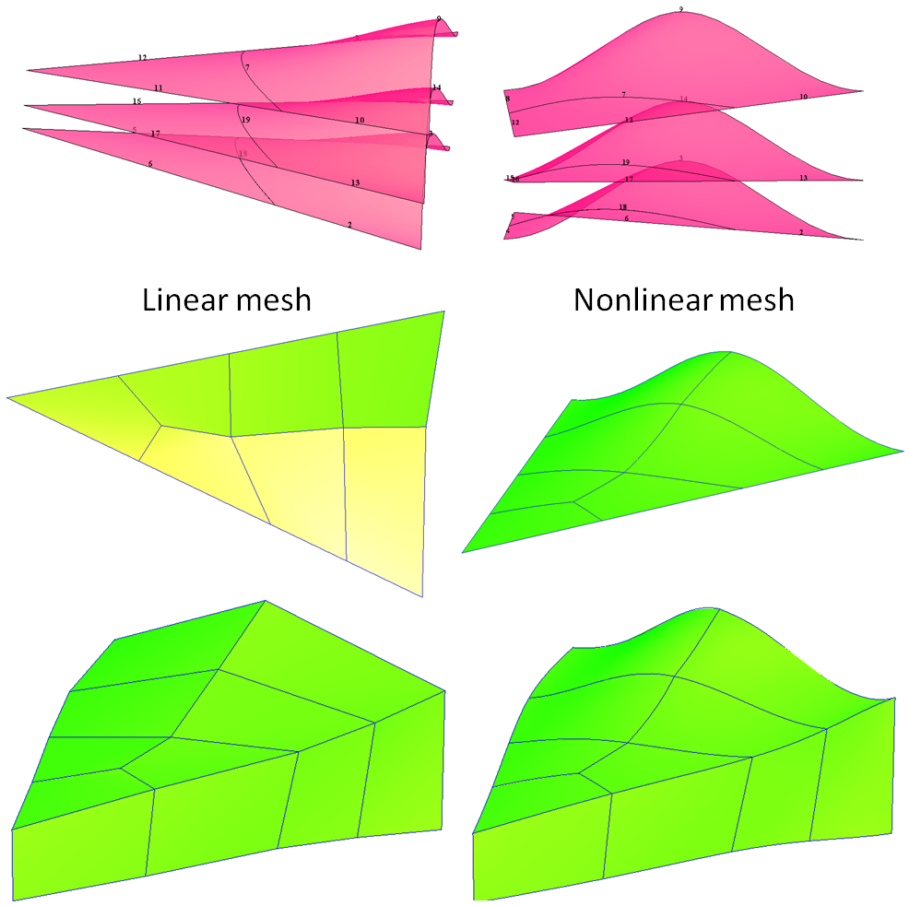


Figure 6.32: Healed geometric model of modified extr. volume model and meshes on the model

6.4 BREP Volume Healing Examples

This section demonstrates the performance of the currently implemented BREP volume healing routines. As for extrusion volumes the recursive implementation concepts includes region healing routines within the BREP volume healing workflow. The focus of this section is however only on the operations related to the volumetric character of the models itself.
A BREP volume is bounded by shells, which themselves consist of regions. A BREP volume must have exactly one boundary shell and can have several internal shells or holes. The volume healing routines have the purpose to validate volumes in the model according to the conditions of validity described in section 2.2.6. The conditions can be summarized by stating that the volumes must form manifold complexes. This is guaranteed by disjointedness of vertices, edges and faces. Non-disjoint edges must meet at one vertex. Non-disjoint faces have to share a common edge at the intersection. As illustrated in section 2.2.6 these conditions are one the one hand to be checked on a topological level meaning that the connectivity of edges and regions forming a volume must result in a closed volume. On the other hand a validity check would also include the geometrical information since primitives might have intersections or even self-intersections due to the geometrical shape assigned to the edges or faces.
The implemented healing algorithms to analyse the validity of BREP volumes in the model include mainly topological investigations at the current state. The main purpose of the algorithm is to analyse if the set of regions and edges included in the description of a single volume is topologically able to form a valid BREP. If this is not the case the algorithm tries to extract a

valid BREP volume by considering the neighbourhood of it in the geometric model. Geometric relations are only used to check if inner shells describing holes for example are inside the boundary shell of the volume. Furthermore the available routines try to find further valid BREP volumes in the geometric model if this functionality is enabled. The performance of the code is illustrated with the following examples.

6.4.1 Ex_08 BREP Volume Healing: Simple BREP example

The first example is a geometric model consisting of three BREP volumes arranged in form of an L-shape sketched in picture 6.33. The volumes have a simple linear shape and no internal holes or shells. The boundary shells consist simply of 6 regions. The volumes have one respectively two regions neighbouring another volume.

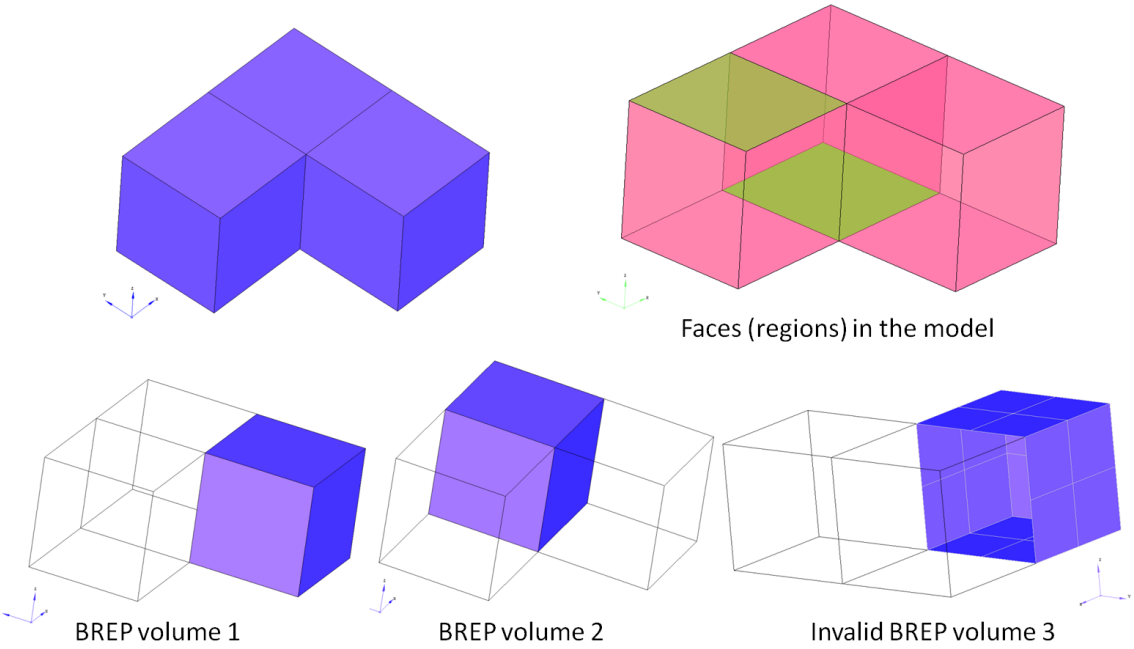


Figure 6.33: Geometric model with three BREP volumes; visualization of regions in the model

Picture 6.33 also visualizes the shape of every single volume. The volumes 1 and 2 form valid BREPs in contrast to volume 3 where one face is missing, resulting in an invalid non-closed solid. This volume violates the conditions of validity since a point cannot be tested to be inside or outside the volume.
Picture 6.33 shows also all the modelled regions (faces) in the geometric model. So we can see that all the regions bordering two adjacent volumes are present in the model even the region missing in the boundary of volume 3. Two regions are coloured green to indicate that actually several regions are overlapping in the model at this points. The presence of these double or non-disjoint regions violates the conditions of validity and will also cause problems in the BREP check algorithm described in the following. This model is clearly a self-constructed test example to check the performance of the routine. Nevertheless invalid BREP volume description due to missing regions is a problem of practical relevance in geometric modelling.
The volume healing routine loops over all volumes of the model and analyses their topological validity. Section 3.2.2 describes the theoretical background of the algorithm. In short, the algorithm computes the kernel of the Edge-Region incidence matrix. The resulting Nullspace vectors (if any vector is found) hold topological information on a valid BREP volume description.

If a volume is detected to be invalid the algorithm tries to validate it as it will be demonstrated with this model. Volume 1 forms a valid BREP what should also be confirmed by the algorithm. Picture 6.34 sketches the corresponding Edge-Region incidence matrix of this volume.

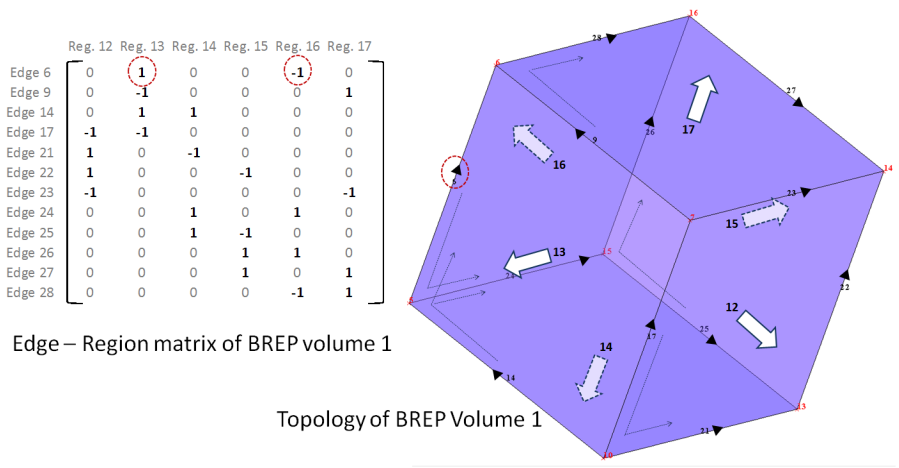


Figure 6.34: Edge-Region incidence matrix of BREP volume 1

The Nullspace or kernel is computed to check the topological validity. Exactly one Nullspace vector is found for volume 1 sketched in figure 6.35. The existence of a Nullspace vector is a necessary condition for topological volume validity. If exactly one Nullspace vector is found, only one BREP volume can be constructed from the analysed set of edges and regions (cycles). In the case of volume 1 all regions are included in the description of a valid BREP volume indicated by a non-zero entry in the Nullspace vector. The magnitude is irrelevant since the vector can be arbitrarily scaled. The sign nevertheless indicates a valid orientation of every single boundary wire of the regions to form a valid BREP volume in total. The convention is set by the Möbius rule. Every edge is shared by two faces and following the orientation of the wires in the different faces this edge is traversed in opposite direction. Details can be found in section 3.2.2.

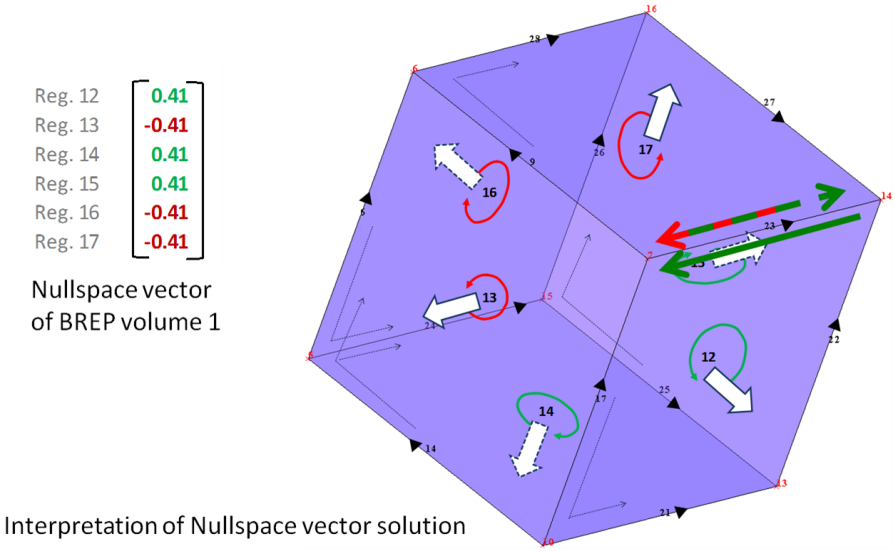


Figure 6.35: Interpretation of Nullspace vector of BREP volume 1

Picture 6.35 illustrates the interpretation of the signs in the computed Nullspace vector for this

particular BREP volume. Since the vector can be scaled arbitrarily the sign only guarantees a consistent orientation without telling anything about clock- or counter-clockwise directions. Volume 2 is validated in a similar manner. Volume 3 is detected to be invalid since no Nullspace vector is found at all for the regions and edges assigned to the volume. This volume is marked to be invalid by the routine and passed to another algorithm which tries to create a valid BREP volume on basis of the defected description. Therefore the map of edges and regions which is used to detect a valid BREP volume is extended by all regions sharing topologically an edge with the volume of investigation. The specific regions are shown in picture 6.36 and coloured magenta. Since non-disjoint regions (double regions) are present they will also be added to the extended search map indicated by the green colour in the picture. At this point it is important to mention, that prior region healing routines are important since they have the purpose to validate the topological relations on level of nodes, edges and faces, guaranteeing that no double nodes, edges or non-consistent face-edge relations are present in the model anymore. If this is not the case the volume healing routines will fail.

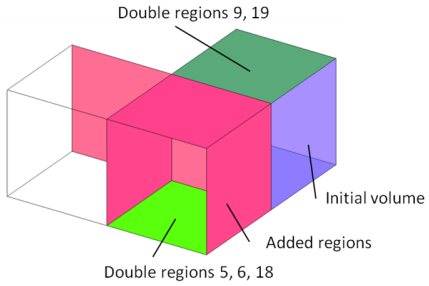


Figure 6.36: Extended Region matrix to validate BREP volume 3

In a next step the kernel of the extended Edge-Region matrix is computed. Since all neighbouring regions are added to the map without considering any further conditions we first have to extract the information we need from this matrix. In this case four different Nullspace vectors are found. The reasons to find more than one vector might be that an entire neighbouring volume was added to the map or like it is the case here that double regions are present. Therefore the algorithm first performs basic column operations to detect double regions in the map. To be more specific the algorithm extracts the four linear independent Nullspace vectors. Topologically a double region is a valid BREP volume consisting of only two regions. Only a geometric check reveals that those two regions actually are identical and non-disjoint. The algorithm here nevertheless is not dependent on a geometric test since a volume formed by two double regions is clearly not of interest at this point. Therefore the algorithm searches the vector with the largest intersection of boundary regions with the original defective volume and deletes double regions only contributing to a two region volume. Iteratively all double regions are detected and deleted from the map such that we finally end up with one single Nullspace vector sketched in figure 6.37. Several regions in the map do not contribute to a valid BREP indicated by the zero entry. The missing region number 8 is detected and the volume can be validated as shown in picture 6.37.

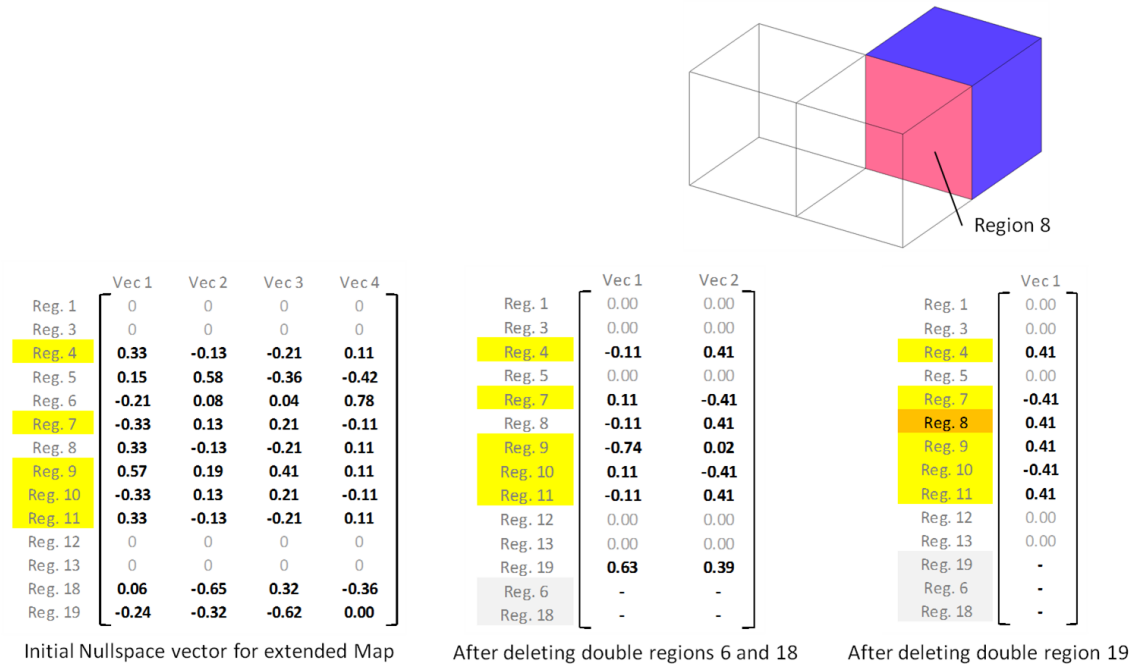


Figure 6.37: Intermediate and final Nullspace solution validating BREP volume 3

After all BREPs are validated the model can be used in downstream applications like meshing for example. The volume mesh on a volumetric BREP model is constructed using the external Netgen library. The resulting mesh is shown in figure 6.38. This meshing step for BREP volumes is not yet automated and scope of further code development.

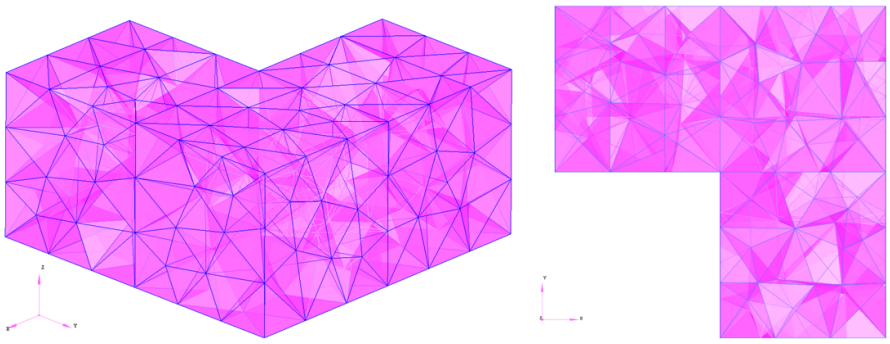


Figure 6.38: Tetrahedra mesh on healed geometric volume model

6.4.2 Ex_09 BREP Volume Healing: Simple BREP example

The model in this example is of the same type than in the example before. The purpose is to illustrate again the steps included in the algorithm to validate and find valid BREP volumes. The model is visualized in figure 6.39. Four cubes are arranged in a T-shape. A detailed look on all volumes reveals that every BREP volume is invalid due to the fact that the inside faces neighbouring the volumes are not added to the volume description.

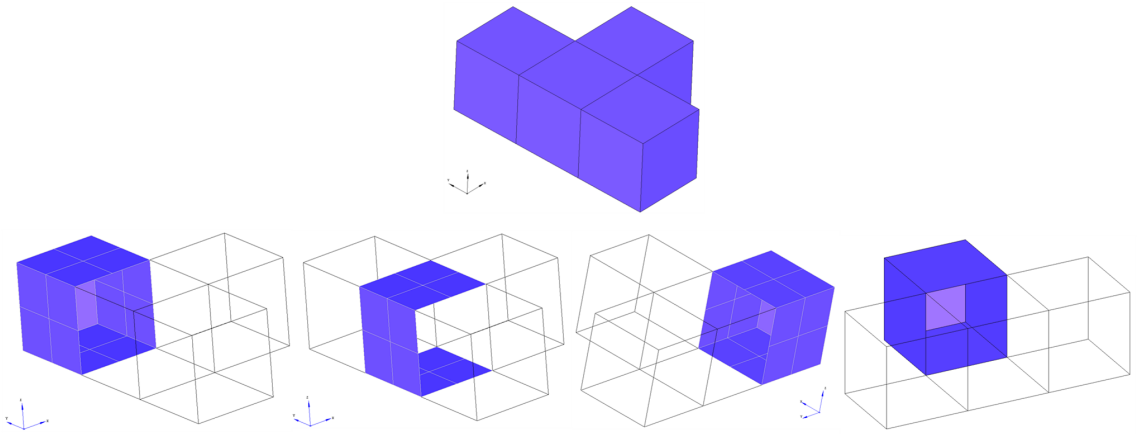


Figure 6.39: Original geometric model consisting of four BREP volumes

Picture 6.40 sketches for every single volume the extended region map amongst which the missing regions are found and a valid BREP volume is extracted. The algorithm detects all missing regions without a problem no matter if double regions are present in the model or not. Important is to mention that this is only the case because the faces which are missing are present in the model but are simply not assigned to the volumes.

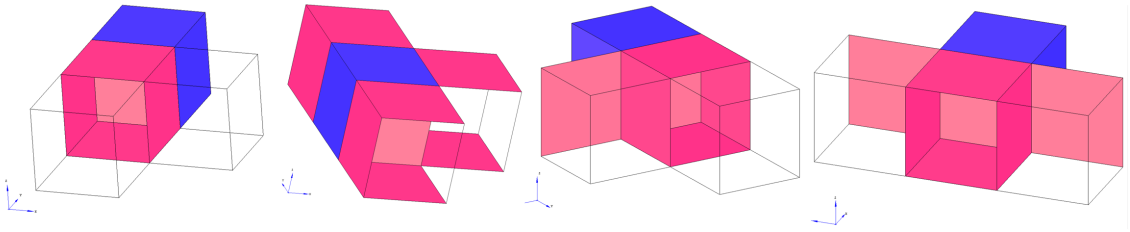


Figure 6.40: Single BREP volumes with their extended region maps

The situation is significantly different if those regions are totally missing in the model as sketched by picture 6.41. First of all none of the volumes can be validated since the algorithm only tries to add regions present in the model. Therefore in a next step all four volumes are deleted from the model since they are still invalid after the routine terminates.

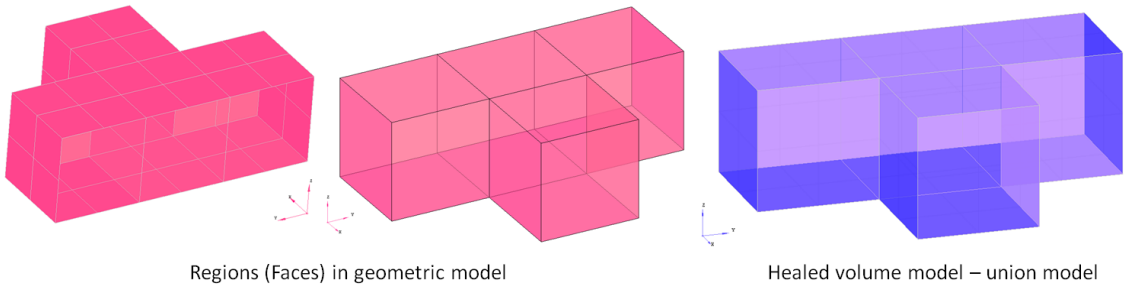


Figure 6.41: Modified geometric volume model and its healed union volume representation

After deleting invalid volumes the routine tries to find all other possible BREP volumes if this functionality is enabled. In this case the algorithm will find the only possible valid BREP volume

including all regions in the model. The resulting BREP volume is the union of all the original single volumes. Picture 6.41 sketches the final volumetric model.

6.4.3 Ex_10 BREP Volume Healing: BREP model with inner holes

The geometric model presented in this example is also self-created but more complex than the examples before. Picture 6.42 sketches all the faces present in the model. We can visually detect 5 volumes which are able to be formed out of those regions. The outer two volumes would then have two internal holes described by a little cube.

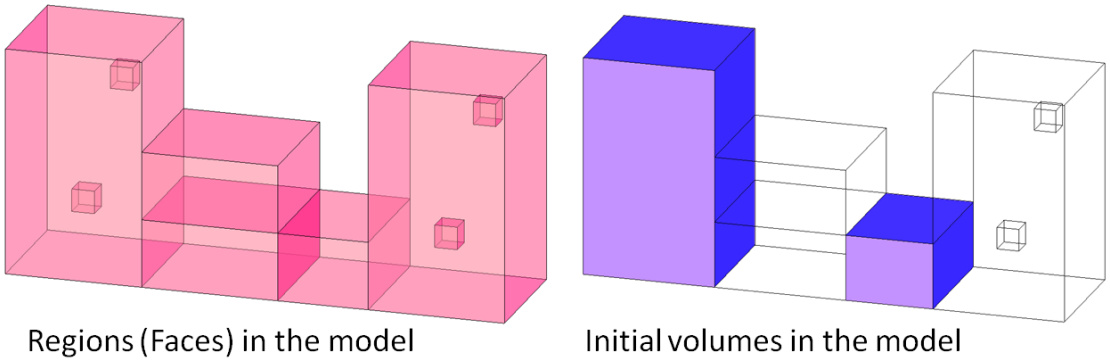


Figure 6.42: Faces and initial BREP volumes of geometric BREP volume model

On the right side of picture 6.42 the volumes actually modelled are visualized revealing that only two of the five possible volumes are included in the model. The example was created to check the performance of the code towards more complex examples and test the behaviour in case of internal holes. Double regions can also be found in the model although not visualized here. Picture 6.43 shows the resulting model after application of the BREP volume healing routines. We can see that all five volumes are detected correctly. First of all the code validates the existing volumes in the model. After that the other three possible volumes are detected correctly. The algorithm is described in detail in section 5.5. On creation time of new volumes every volume is tested to be geometrically inside another one and therefore must form a internal hole. Even in that simple example it becomes evident that the mutual checks have to be implemented efficiently to decrease the processing time.

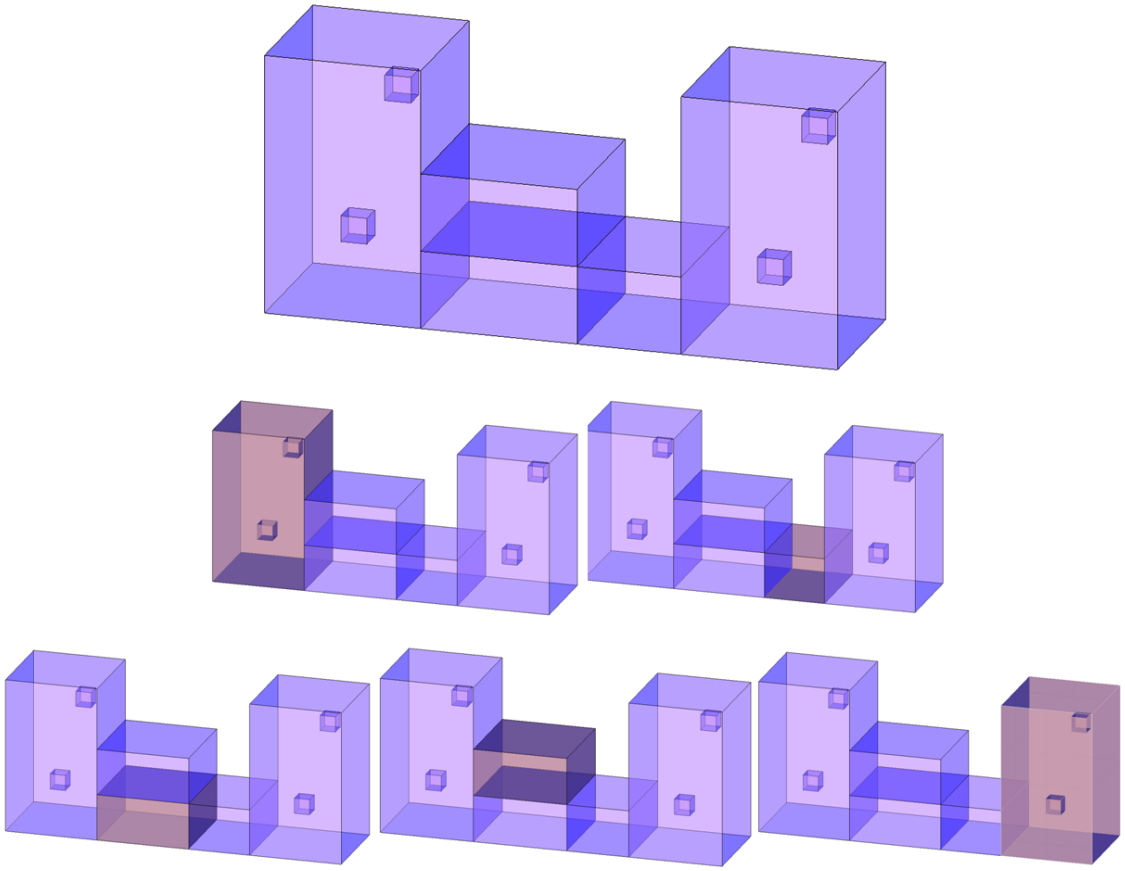


Figure 6.43: Healed geometric model with all possible BREP volumes

6.4.4 Ex_11 BREP Volume Healing: non-linear BREP model

This example differs from the simpler ones at the beginning of this section only in so far as the geometry of the model is more complex as it can be seen in picture 6.44.
It was mentioned that the algorithm at its current state to check the BREP validity is of pure topological character and not affected by the geometric shape. Therefore for the algorithm the cube does not differ in any way from a simpler linear cube. As a consequence the algorithm has no problem to validate the BREP volume. As mentioned further geometric algorithms would be necessary to analyse if the faces of the model intersect and therefore violate the conditions of validity. The Netgen library is used to generate a Tetrahedral mesh on the model as shown in picture 6.44.

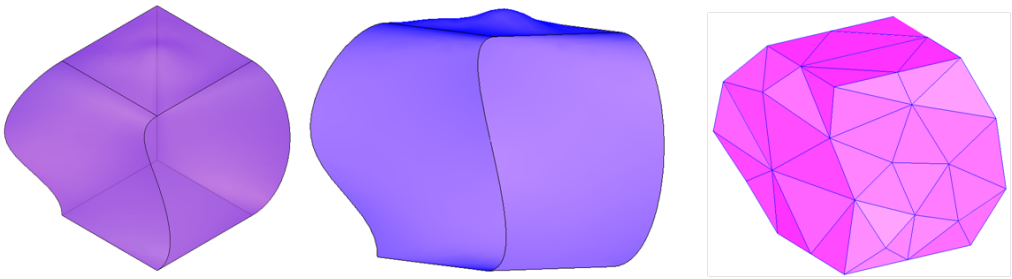


Figure 6.44: Geometric model of BREP volume and Tetrahedra mesh

Chapter 7

Summary, Conclusions and Outlook

7.1 Summary and Conclusions

At the beginning of this thesis problems or gaps in the engineering analysis workflow due to errors and inconsistencies in the geometric model were described. The problems mainly arise during the modelling procedures itself or while exchanging geometric information between different systems due to translation or round-off errors. The impact of errors and defects on the workflow can be significant and might reach from inefficiencies to a total failure of routines in downstream applications. This thesis illustrates the gaps and errors in the workflow with the help of several examples created in commercial modelling software and imported into the TUM.GeoFrame framework. The consequences of defects are mainly described with respect to meshing algorithms. The purpose was to motivate the implementation of efficient and robust healing routines on geometric models. Considering that according to Michael Hardwick and Robert Clay (Sandia National Laboratories) modelling and meshing can cover up to 70 percent of the analysis time for modern engineering problems additional time consuming healing operations on the model are critical for the efficiency of the entire workflow. Since healing errors in geometric models by hand is often impossible or too costly for complex models automated routines with high efficiency and robustness are required.
Within this thesis model errors and inconsistencies, their sources, consequences and possible remedies are analysed and described in a theoretical manner. In a next step aspects of algorithmic implementation are covered. The implementation of own healing routines in an existing framework is the major objective of this thesis. The goal was to extend the existing TUM.GeoFrame framework by a healing sub-project. Therefore a concept for the implementation of healing routines in form of a half-standalone sub-project was developed. The implementation concept and code design is described and illustrated in this work.
The interaction of the user with the healer project is one aspect of the implementation concept. A basic dialog was created to communicate with the healer out of the GUI provided by the existing TUM.GeoFrame framework. The core framework was extended by several routines to realize the user interaction and storage of user defined healing settings.
Furthermore the healer implementation concept covers a data structure concept within the healer sub-project as well as the necessary interfaces with the core framework. Beside general project information a local topology kernel within the healer is realized. This local topology kernel holds all the topological information of the healer during processing time and differs slightly from the global topology kernel within TUM.GeoFrame. The local kernel is optimized for the healer routines to guarantee fast access to topological relations exploited in the healing algorithms.
The core of the healer project is a collection of healing algorithms and operations implemented within the scope of this thesis. By far the most algorithms are related to edges, nodes and wires and bundled to region healing routines. The second largest group of algorithms is related

to BREP volumes and exploited in routines checking their topological and geometric validity. However the algorithms themselves are only one part of the implementation concept. The central aspect of implementation is rather related to the routines bundling the specific operations to an entire workflow. For each of the three major entity groups - regions, extrusion volumes and BREP volumes - exactly one healing routine is provided. The routines are defined by a given sequence of healing operations. Efficiency and robustness of the entire workflow motivates the order in which the operations are called. The implementation concept for the routines is related to the hierarchical character of BREP models. With a range of different geometric models, self-constructed as well as practical external examples, the performance, efficiency and stability of the code was tested and improved. As sketched in this thesis the healer code at its current state performs stable and efficient on various models, also including complex geometric shapes and larger model sizes.
It turned out that the order of healing operations might have a significant influence on the efficiency and robustness of the entire healing workflow. Therefore the optimization of the workflow was one central aspect of the implementation concept. This concept is extended by and iterative approach including dynamic adaption of healing tolerances. The concept is described in detail within this thesis.
The performance of the implemented code is demonstrated with the help of a range of different examples illustrated in the last part of the thesis. The healed models were used in downstream applications, in particular meshing algorithms, to illustrate the positive contribution of the implemented routines to an entire analysis workflow.

7.2 Outlook and further development

It was not the primary goal of this thesis to provide the TUM.GeoFrame framework with a finished “all-purpose” healer sub-project. The focus was rather on setting up a structure and concept for further development of the healer code. One reason why further development is necessary clearly lies in the restricted realization time. Healing routines and algorithms are time consuming to develop and implement due to the complexity of geometric shape information and the fact that the performance has to be checked carefully on a wide range of examples to guarantee stability and robustness. On the other hand the existence of such a finished “all-purpose” healer code can be doubted. The ambiguous character of many geometric and topologic operations sets limits for the automation of healing routines without any additional user interaction. Often the desired result of healing operations is not evident for a computational algorithm since the code has only little inside on the model’s characteristics. The success of many operations depends significantly on suitable tolerance values which are not known beforehand in general. A classical example is the disability of a code to judge if a gap between two surfaces or edges is a desired hole or an inconsistency of the model. The decision mainly depends on the applied tolerance in this case and therefore on suitable user settings. Therefore classical algorithms might never be able to provide a robust algorithm with no need for any kind of user interaction.
Implementation of smart user interaction can be one powerful remedy to overcome this ambiguity. The goal of a robust healer must be to perform stable and efficiently on a wide range of detected errors and inconsistencies and include smart user interactions in case of ambiguous situations in the model.
All of these aspects inspire the further development of the healer code. The development will have to address three major aspects. On the one hand the set of algorithms has to be extended by additional geometric and topological operations. For region healing routines the code has to be equipped with robust surface intersection detection algorithms. At its current state the code only considers intersection problems between surfaces or regions on the boundaries or in the

vicinity of the boundaries. Within a tolerance sphere intersections are detected and healed on level of nodes, edges and wires. Intersection problems inside the surfaces are not considered and healed by the code at its current state. For trimmed NURBS surfaces these types of intersection algorithms are not trivial and will be part of future code improvement. Surface intersection problems also play an important role to check the validity of BREP volumes which are at the current state validated mainly on a topological level. For extrusion healing routines the code has to be extended by volume specific operations. Algorithms checking if the defined set of shells in combination with the extrusion paths will result in a meaningful volume might be a necessary and useful healing operation for example, which is not yet implemented in the code at its current state. This problem is also in a way related to surface intersections.
In a second step all these additional algorithms would have to be included into the existing healing routines in an efficient manner. Modification and extension of healing routines will have to go along with intensive testing and debugging against a wide range of geometric models. In general debugging and testing of the healer code is crucial to optimize the healing workflow and improve stability and performance. Within the testing process code bugs are eliminated and tolerance settings and factors for tolerance adaption might be optimized. Furthermore almost every complex model has its own intrinsic characteristics challenging healing routines and therefore contributes to a stable and robust workflow within the healer.
The improvement of user interaction in the healing process is the third major aspect of future development. At its current state the user can only interact with the healer at the start of healing routines and initiate further healing steps after checking the results of the applied routines with the defined tolerance settings. As illustrated in the thesis an interactive approach within the healing process itself would be beneficial for robustness, efficiency and stability. The user's decision can be regarded or even necessary in cases where the algorithm is confronted with ambiguous situations. Therefore the problematic situations must be visualized in a way such that the user can make fast decisions and therefore the overall healing workflow will be an efficient and robust compromise between minimized user interaction and overall healing routine efficiency and robustness.
Healing routines are confronted with an almost infinite number of geometric characteristics. Therefore the code has a high potential for further development and improvement. The structure of the code as realized within the scope of this thesis is supposed to form a basis for these future improvements.

List of Figures

2.1 Entire production and design cycle in the engineering workflow . . . 3
2.2 Idealization example in tunneling and building construction . . . 4
2.3 Pre-Processing, Processing and Post-Processing workflow . . . 5
2.4 Sub-tasks in the processing and analysis workflow . . . 5
2.5 Simple graph enxample . . . 6
2.6 UML structure for description of spatial objects . . . 8
2.7 Invalid intersection operations in the CSG scheme . . . 8
2.8 Complex CSG example constructed in AutoCAD . . . 9
2.9 Non-unique CSG-tree for a geometric model . . . 9
2.10 Simple standard enumeration scheme example . . . 10
2.11 Simple Quadtree example . . . 11
2.12 Quadtree example in computer graphics [28] . . . 12
2.13 BREP model with same topology but different geometric shapes . . . 12
2.14 Vef-relations in the BREP representation scheme . . . 14
2.15 Invalid BREP model: double entities . . . 15
2.16 Invalid BREP model: non-manifold points and surface intersections . . . 15
2.17 Rational Bézier description of a circle quadrant . . . 20
2.18 Non-rational curve projection in 4D space . . . 21
2.19 Piecewise parametric curve description . . . 22
2.20 B-Spline basis function triangle . . . 23
2.21 Trimmed NURBS surface model describing a violin top plate . . . 26
2.22 Trimming and non-watertight intersections . . . 27
2.23 Example for a T-Mesh . . . 27
2.24 Information exchange via common systems and standards . . . 29
2.25 Information exchange via direct translators . . . 29
2.26 Information exchange via neutral file formats . . . 30
2.27 Typical structure of an IGES file . . . 32
2.28 Boundary decomposition due to exchange via IGES . . . 33
2.29 STEP specification . . . 33
2.30 FEM formulation and element shapes . . . 35
2.31 FEM elements and meshes for shell-like structures . . . 36
2.32 Location of the healer in the discretization process . . . 37
2.33 Healing routines in the workflow . . . 37

3.1 Intersection curve components from S/S intersection pairs . . . 53
3.2 Marching methods: component jumping and subdivision remedy . . . 54
3.3 Non-orientable manifold: Möbius strip . . . 56
3.4 General graph example . . . 57
3.5 Linear graph example . . . 57
3.6 Infinite linear graph example . . . 57

3.7 Complete graph example . . . 58
3.8 Spanning tree and fundamental cycles . . . 63
3.9 Simple graph example . . . 64
3.10 Simple graph example with two disconnected substructures . . . 64
3.11 Simple graph example with one disconnected end . . . 65
3.12 Fundamental cycles in the graph . . . 66
3.13 Spanning tree on the graph . . . 66
3.14 BREP model with three regions . . . 67
3.15 BREP model of regions with periodic boundaries . . . 68
3.16 Simple BREP volume model . . . 69
3.17 Node-Edge graph of a BREP volume . . . 69
3.18 Cycles and faces in a BREP model . . . 70
3.19 BREP model and bounding cycles . . . 71
3.20 Alternative Cycle-Edge graph representation of volume (cycles as vertices) . . . 71
3.21 Hypergraph of BREP volume . . . 72

4.1 Structure of the TUM.GeoFrame framework . . . 76
4.2 Overview over topological entities in the TUM.GeoFrame framework . . . 77
4.3 Relation between Topo_Objects, Geo_Objetcs and Attributes . . . 78
4.4 Interfaces of the healer sub-project with the core of TUM.GeoFrame . . . 79
4.5 Contributions of this thesis to the framework . . . 80
4.6 TUM.GeoFrame workflow without healer . . . 81
4.7 TUM.GeoFrame workflow with healer . . . 81
4.8 User interaction via Healing Dialog within the framework . . . 82
4.9 TUM.GeoFrame workflow with healer and user interaction . . . 82

5.1 Hierarchy of tolerance values . . . 86
5.2 Hierarchical tolerance adaption workflow . . . 87
5.3 Hierarchical recursive implementation concept of healing routines . . . 88
5.4 Adaptive tolerance concept in the region healing workflow . . . 89
5.5 Optimized healing workflow concept . . . 90
5.6 Region model with a dangling edge . . . 94
5.7 Edge merging procedure . . . 95
5.8 Linear edge merging procedure . . . 96
5.9 Composed edge construction . . . 96

6.1 Rhino and IGES representation of the example's geometric model . . . 107
6.2 Rhino Modeller: detail on trimming curve . . . 109
6.3 Surface Quad meshes on the original model . . . 110
6.4 Inconsistencies in the surface mesh . . . 110
6.5 Geometric model after application of healing routines . . . 111
6.6 Surface Quad meshes on the healed model . . . 112
6.7 Original geometric model and surface Quad mesh on model . . . 112
6.8 Healed geometric model and surface Quad mesh on healed model . . . 113
6.9 Original geometric model of violin's top plate and detail on f-holes . . . 114
6.10 Surface Quad mesh on original model and detail of mesh at the ears . . . 115
6.11 Surface Quad mesh on healed model and detail of mesh at the ears . . . 115
6.12 Surface Quad mesh on the entire violin corpus . . . 116
6.13 Original geometric model of a wind turbine . . . 117
6.14 Healed geometric model of the wind turbine after region healing routines . . . 117

6.15 Original geometric model of a container ship . . . 118
6.16 Tip of the original geometric ship model with poor accuracy . . . 118
6.17 Surface Quad mesh on original geometric ship model . . . 119
6.18 Healed geometric ship model . . . 120
6.19 Surface Quad meshes on healed geometric ship model . . . 120
6.20 Region model with dangling edge . . . 121
6.21 Modified region model with dangling edge . . . 122
6.22 Mesh on a model with a dangling edge . . . 123
6.23 Geometric model describing an extrusion volume . . . 123
6.24 Labelled geometric model with boundary edge decomposition . . . 124
6.25 Healed geometric model of an extrusion volume . . . 124
6.26 Surface Quad mesh on reference shell of the original model . . . 124
6.27 Volumetric Hex mesh on original model . . . 125
6.28 Surface and volumetric meshes on healed model . . . 125
6.29 Modified model with two regions per shell in its original form . . . 126
6.30 Surface Quad mesh on original modified extr. volume model . . . 126
6.31 Volumetric Hex mesh on original modified extr. volume model . . . 126
6.32 Healed geometric model of modified extr. volume model and meshes on the model 127
6.33 Geometric model with three BREP volumes; visualization of regions in the model 128
6.34 Edge-Region incidence matrix of BREP volume 1 . . . 129
6.35 Interpretation of Nullspace vector of BREP volume 1 . . . 129
6.36 Extended Region matrix to validate BREP volume 3 . . . 130
6.37 Intermediate and final Nullspace solution validating BREP volume 3 . . . 131
6.38 Tetrahedra mesh on healed geometric volume model . . . 131
6.39 Original geometric model consisting of four BREP volumes . . . 132
6.40 Single BREP volumes with their extended region maps . . . 132
6.41 Modified geometric volume model and its healed union volume representation . . . 132
6.42 Faces and initial BREP volumes of geometric BREP volume model . . . 133
6.43 Healed geometric model with all possible BREP volumes . . . 134
6.44 Geometric model of BREP volume and Tetrahedra mesh . . . 134

List of Tables

2.1 vef-relations in the BREP modelling scheme 13

3.2 S/S Intersection approaches 53

5.1 Tolerance values 85

Bibliography

[1] *IGES - File Format Specification.* http://ts.nist.gov/standards/iges/.

[2] *ISO - International Standarization Organization.* www.iso.org/.

[3] *LAPACK - Linear Algebra PACKage.* www.netlib.org/lapack/.

[4] *NETGEN - automatic mesh generator.* www.hpfem.jku.at/netgen/.

[5] *Open CASCADE.* www.opencascade.org.

[6] *QT Cross-platform application and UI-framework.* http://qt.nokia.com.

[7] *STEP - STEP on one page.* http://www.mel.nist.gov/sc5/soap/.

[8] *STEP standarization: ISO_10303.* http://en.wikipedia.org/wiki/ISO_10303-21.

[9] *VTK - Visualization Toolkit.* www.vtk.org.

[10] Lars-Erik Andersson, Neil F. Stewart, and Malika Zidani. Error analysis for operations in solid modeling in the presence of uncertainty. *SIAM Journal on Scientific Computing*, 29:811–826, 2007.

[11] Bruce G. Baumgart. Winged-edge polyhedron representation for computer vision. Technical report, 1975.

[12] John Adrian Bondy and U.S.R. Murty. *Graph Theory With Applications.* Elsevier Science Ltd, 1976.

[13] Martin Breitenbach. Modellierung von dünnwandigen Strukturen für die Vernetzung mit gekrümmten Hexaeder Elementen. Technical report, 2011.

[14] Dominique Chapelle and K.J. Bathe. *The Finite Element Analysis of Shells: Fundamentals.* Springer, 2003.

[15] C.S. Chong, A. Senthil Kumar, and H.P. Lee. Automatic mesh-healing technique for model repair and finite element model generation. *Finite Elements in Analysis and Design*, 43:1109–1119, 2007.

[16] J. Austin Cottrell, Thomas J. R Hughes, and Yuri Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA.* John Wiley and Sons, 2009.

[17] Fred. H. Croom. *Principles of Topology.* The Saunders Series. Holt, Rinehart and Winston, Inc., 1989.

[18] Reinhard Diestel. *Graph Theory.* Graduate Texts in Mathematics. Springer-Verlag, 2006.

[19] Rida T. Farouki. Closing the gap between cad model and downstream application. *SIAM News*, 32:303–319, 1999.

[20] Rida T. Farouki. Workshop on Mathematical Foundations of CAD, 1999.

[21] Felix Frischmann, Martin Buchschmid, Martin Ruess, Johannes Kreutz, and Gerhard Müller. Numerical models for violins. In *DAGA - Jahrestagung für Akustik*, 37, 2011.

[22] Christoph M. Hoffmann. *Geometric and solid modeling: an introduction*. Morgan Kaufmann, 1989.

[23] Christoph M. Hoffmann. Robustness in geometric computations. *J. Computing and Information Science in Engineering*, 1:143–156, 2001.

[24] Michael E. Hohmeyer. A surface intersection algorithm based on loop detection. Technical report, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, California, 1991.

[25] Thomas J.R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Prentice-Hall, 1987.

[26] David J. Kasik, William Buxton, and David R. Ferguson. Ten cad challenges. Technical report, 2005.

[27] Shankar Krishnan and Dinesh Manocha. An efficient surface intersection algorithm based on lower dimensional formulation. Technical report, Department of Computer Science, University of North Carolina, 1997.

[28] Sylvain Lefebvre. *Modèles d'habillage de surface*. PhD thesis, Université Joseph Fourier de Grenoble, 1992.

[29] Martii Mäntylä. *An introduction to solid modeling*. Computer Science Press, 1988.

[30] William S. Massey. *A Basic Course in Algebraic Topology*. Graduate Texts in Mathematics. Springer-Verlag, 1991.

[31] Stéphane M. Morvan and Georges M. Fadel. Ivecs: an interactive virtual environment for the correction of .stl files. Technical report, College of Engineering and Science, Clemson University, 1996.

[32] Nicholas M. Patrikalakis and Takashi Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer-Verlag, 2010.

[33] N. Anders Petersson and Kyle K. Chand. Detecting translation errors in cad surfaces and preparing geometries for mesh generation. Technical report, Center for Applied Scienttific Computing, Lawrence Livermore NAtional Lab, 2001.

[34] Les Piegl and Wayne Tiller. *The NURBS Book*. Monographs in visual communication. Springer-Verlag, 2 edition, 1997.

[35] Ernst Rank, Stefan Kollmannsberger, Christian Sorger, and Alexander Düster. Shell finite cell method: A high order fictitious domain approach for thin-walled structures. *Computer Methods in Applied Mechanics and Engineering*, 200:3200–3209, 2011.

[36] Artistides A. G. Requicha. Representations for rigid solids: Theory, methods, and systems. *ACM Computing Surveys*, 12:437–464, 1980.

[37] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, 16(2):187–260, 1984.

[38] Abhijit V. Sawat and John W. Nazemetz. Impediments to data exchange and use of step for data exchange. Technical report, Oklahoma State University.

[39] Anupam Saxena and Birendra Sahay. *Computer Aided Engineering Design*. Springer Netherlands, 2005.

[40] Joachim Schöberl. Netgen - an advanced front 2d/3d-mesh generator based on abstract rules. *Computing and Visualization in Science*, pages 41–52, 1997.

[41] Manfred Schweingruber-Straten. *Generierung von Oberflächennetzen nach der Gebietsteilungstechnik*. Shaker Verlag, 1999.

[42] Matthew T. Sederberg and Thomas W. Sederberg. T-splines: A technology for marine design with minimal control points. Technical report, T-splines inc. / Brigham Young University, 2010.

[43] Vadim Shapiro. Solid modeling: Handbook of computer aided geometric design. Technical report, 2001.

[44] Christian Sorger, A. Düster, and E. Rank. Generation of curved high-order hexahedral finite element meshes for thin-walled structures. Technical report, Computation in Engineering, Technische Universität München, 2009.

[45] Christian Sorger, Felix Frischmann, Stefan Kollmannsberger, and Ernst Rank. Visual domesh: Automated high-order hexahedral mesh generation for shell-like structures. Technical report, Computation in Engineering, Technische Universität München, 2011.

[46] Ian Stroud. *Boundary Representation Modelling Techniques*. Springer, 2006.

[47] U.S. Product Data Association (US PRO), www.uspro.org. *Initial Graphics Exchange Specification 5.3*, 2006.

[48] Ibrahim Zeid. *CAD/Cam Theory and Practice*. The McGraw-Hill Companies, 2009.