Technology
Arts Sciences
TH Köln

Deutsches Zentrum
für Luft- und Raumfahrt
German Aerospace Center

Institute for Software
Technology

Masterthesis

Maschinenbau
Smart Systems (M.Sc.)

Towards high-order, hybrid adaptive mesh refinement: Implementation and evaluation of curved unstructured mesh elements

Vorgelegt von:	Jakob Fußbroich	Erstprüferin:	Prof. Dr. Claudia Ziller
	Im Bachfeld 6	Zweitprüfer:	Sandro Elsweijer
	51063 Köln	Eingereicht am:	13.11.2023
	jakob.fussbroich@gmx.de		
	Matr.-Nr. 11125395		

Contents

List of Figures

List of Tables

Acronyms

AMR	adaptive mesh refinement
B-rep	boundary representation
CAD	computer aided design
CFD	computational fluid dynamics
CFL	Courant-Friedrichs-Lewy
CPU	central processing unit
DG	discontinous Galerkin method
FEM	finite element method
FVM	finite volume method
HPC	high performance computing
MPI	message passing interface
MSFC	mesh size from curvature
OCCT	OpenCASCADE Technology
PDE	partial differential equation
SFC	space filling curve

1 Introduction

Understanding complex physical processes and the ability to generate solutions out of the gained knowledge is a critical task in modern research fields and the industry. Creating a simplified model that can be mathematically analysed is often necessary to gain insight into these processes. The boundaries of the model have to be set so that the effort of solving its equations is feasible but still produces meaningful results, helping to understand the physical process behind it. Examples of complex physical processes in mechanical engineering would be the occurrence of stresses in structural components or the development of a flow field around a moving vehicle.

Physical processes can generally be described by partial differential equations (PDEs). In most cases, finding an analytical solution to these is not possible. Therefore, a numerical approach is often chosen. The advantage is that numerical problems can be solved using computational resources. Whilst generating results quicker and with fewer errors, compared to human calculations, numerical solutions always approximate the analytically correct answer and are therefore always carrying a deviation [1]. How slight this deviation is depends on the simulation model's modelling and resolution. Numerical solvers for partial differential equations have been researched since the 19th century and are still developed [2, 3].

With the development of more efficient hardware, solving complex systems on a personal workstation is possible in reasonable time. Larger scale research simulations, like a computational fluid dynamics (CFD) simulation of a fully equipped aircraft [4] or the exploration of future power systems of a whole country [5], still require more capable hardware, like a high performance computing (HPC) cluster. Running calculations on a HPC framework can be expensive, and runtime is often restricted. Therefore, it is crucial to create simulation models which require the least amount of computational resources possible for the desired accuracy of the results.

1.1 Mesh generation

The general idea behind a numerical approach is to break down a continuous problem into several finite discrete issues. Discretising a domain means creating a mesh out of basic geometrical shapes, so-called elements, to approximate the present geometry. The mesh generation is crucial to the modelling process because its resolution significantly determines the geometrical representation's accuracy and the computational effort needed to solve the PDEs.

Two major approaches in generating meshes are divided into structured and unstructured. Structured meshes usually consist of quadrilaterals in 2D and hexahedra in 3D. The cell boundaries are parallel, and the spacing is uniform, making the mesh generation fast and uncomplicated. This approach regards geometry globally and focuses on the solvability of the mathematical system. However, structured meshes have limited flexibility, which makes them less accurate with complex geometry. The approach of unstructured mesh generation regards the geometry locally and independently of the solving algorithm.

Unstructured meshes provide a higher flexibility and are, therefore, broadly used in the simulation of complex geometry. They usually consist of triangles in 2D and tetrahedra in 3D. Whilst proving advantageous in the geometrical accuracy, unstructured meshes are more complex because of their irregularity. In addition, it is necessary to check the validity of the elements because it is possible to create distorted elements, which can negatively affect the solution. [6]

A hybrid mesh can achieve the most efficient use of the lower computational demand of a structured mesh and the higher geometrical approximation of an unstructured mesh. A hybrid mesh consists out of different element types. Therefore, it can combine structured and unstructured mesh zones, commonly adding a transitional layer of triangular elements to a structured quadrilateral mesh, which usually expands to a tetrahedral and hexahedral mesh in three dimensions [7]. Other elements with a triangular face like prisms or pyramids are also possible. In the case of the simulation of a flow field around a vehicle using a hybrid mesh, the discretisation of the flow field on the surface of the vehicle would happen via an unstructured mesh. In contrast, a structured mesh would be generated in the far area to lower the computational demand.

1.2 Limitations of linear meshing

As mentioned before, meshes consist of simple geometric elements. There are points , which are non-dimensional and lines in 1D, triangles and quadrilaterals in 2D and tetrahedra, hexahedra, prisms and pyramids in 3D. In a standard mesh, the corner nodes of each element are connected with straight lines forming the edges of the elements. The element nodes are, therefore, coupled with a linear function from a mathematical point of view. In the case of a geometry with no curvature, it is possible to gain a perfect geometrical representation using linear mesh elements. In reality, most complex geometries, especially in CFD, have a certain amount of curved surfaces. Lowering the element size in places with higher curvature is necessary to gain a feasible geometrical representation using linear meshes. A smaller element size also means a higher number of elements overall, which increases the computational demand for solving the simulation.

It is also possible to create high-order meshes to keep the deviation of the approximated geometry by the mesh and the physical geometry low. Elements in high-order meshes are no longer linear but curvilinear, representing the geometry more accurately. The typical way of producing high-order meshes is to create a p-version mesh [8]. Instead of a linear function describing the connection of the element nodes, they are represented with a higher polynomial degree. High-order p-version meshes are characterised by superior convergence properties, with the capability of the exponential rate of convergence [9].

However, a near-perfect geometrical approximation of the geometry can in most cases only be achieved by a polynomial of a high degree, which increases computational demand. Additionally, a flawed initial geometrical approximation is carried along even when elements are refined during the simulation process. Another way of gaining a nearly perfect geometrical representation is to curve the mesh directly to the geometry domain.

1.3 Meshes with geometrical information

A common procedure of generating a mesh for a 3D domain is: 1D meshing, 2D meshing and 3D meshing, in this order. First, the generator places mesh nodes along the nodes and edges of the present geometry concerning the preset resolution. After that, it places mesh nodes on the surfaces of the domain, connecting them to the previously placed nodes on the 1D geometries.

After the generation of the 2D mesh, the generator sets mesh nodes in the volume of the domain, finally constructing a 3D mesh. There are specific methods for each meshing process, e.g. Delaunay triangulation for 2D mesh generation and advancing front approach for 3D mesh generation [10]. Regarding this, it becomes apparent that mesh generators only use geometrical information to place mesh nodes and not to generate the connections between them.

It is possible to equip the mesh nodes with additional geometric information. Besides the dimension of the node and the exact geometry it was generated on, it is possible to provide the geometry parameters at the node's location. In a parametric geometry representation (e.g. boundary representation [11]), edges have one parameter u and faces have two parameters u, v.

Each mesh node can be equipped with the parameters of the geometry they are generated on. With this additional geometry information, it is possible to create a high-order mesh without approximating polynomials but with exact geometric details, which can be used to curve the mesh elements to the present geometry directly.

1.4 Research question

The present thesis aims to research a mapping algorithm for triangular and tetrahedral elements to curve them to an arbitrary geometry, achieving a near-perfect geometrical representation. Furthermore, it is desired to validate these curved elements by comparing them to linear elements in a numerical application scenario.

The research results will be implemented in `t8code` [12, 13]. The software `t8code` is a library for dynamic tree-based adaptive mesh refinement (AMR) on hybrid meshes with the ability of parallel mesh management. The functionality for reading-in meshes with additional geometrical information and the capability of `t8code` to curve quadrilateral and hexahedral elements to arbitrary geometries have been implemented in previous research projects [14, 15]. The future goal is to provide these functionalities for all element shapes. The foundation for the missing elements (prisms, pyramids) will be created in this thesis by researching an algorithm for triangular shapes.

The software `t8code` comes with its own advection solver, which will be used to examine the curved elements. The advection solver is already capable of handling the element geometry description used to curve the elements and is expected to provide sufficient results to validate the implementations of this thesis.

Following the information and restrictions given in the introduction, the research question can be defined as follows:

How can the implementation of curved triangular elements in tree-based AMR be realised, and does it prove advantageous over a linear approach?

1.5 Methodology

In Chapter 2, a short overlook of the theoretical foundation needed to follow and reproduce this thesis's results will be given to answer the research question above. Afterwards, in Chapter 3, the research and development of the algorithm will be described in detail, as well as its implementation in `t8code`. First, the objective of the algorithm is formulated. Different possible algorithms will be reviewed to find the most fitting. After an algorithm is determined, the implementation into the software will be described.

In Chapter 4, the implementation will be evaluated, aiming to validate it. Therefore, a simulation of the scenario described in Section 4.1 will be conducted. The simulation results of curved elements will be compared to those of linear elements to gain an overview of the implementations' capabilities and answer the research question. Chapter 5 will conclude the thesis results. In Chapter 6, a perspective on possible future research and developments related to the results of this thesis will be given.

2 Fundamentals

The following sections will provide the theoretical fundamental concepts to understand and follow in the upcoming chapters. First, there will be a brief introduction into AMR. Furthermore, the computer aided design (CAD) library OpenCASCADE Technology (OCCT) will be introduced shortly.

The geometry representation will be focused on because it is essential for developing the mapping algorithm for triangular element shapes. The advection solver, which comes with `t8code`, will be presented and explained.

2.1 Adaptive mesh refinement

As described in Chapter 1, numerical modelling discretizes a continuous physical problem into minor discrete problems by approximating the geometry with a mesh. Depending on the issue and the numerical method used to solve it, the underlying equations can then be solved for every element of the mesh to generate meaningful insights into the physical process. Standard methods for solving PDEs are finite differences, the finite volume method (FVM), the finite element method (FEM) and the discontinous Galerkin method (DG). An overview can be found in [16, 17, 18].

In most cases, the relevant events, like a stress concentration in structural components or turbulent flows at the trailing edge of a wing, are somewhat local. Hence, generating a finer mesh in these regions can be necessary, while other areas in the mesh, where fewer events occur, can be coarsened. Other refinement criteria, like the proximity to a specific geometry (e.g., the trailing edge of a wing), are also used frequently. Iteratively finding mesh regions to be refined or coarsened is inaccurate and time-consuming if done by hand.

AMR automates the process by identifying interesting or less interesting areas in the mesh according to user criteria. Mesh elements are no longer fixed in size but can vary their size to gain higher resolution where needed and vice versa. This process increases the model's resolution while keeping the computational demand equal or decreasing. Running it in parallel to manage AMR for extensive simulations is possible. Parallel, in this case, means distributing the computational effort over multiple message passing interface (MPI) ranks. The first AMR

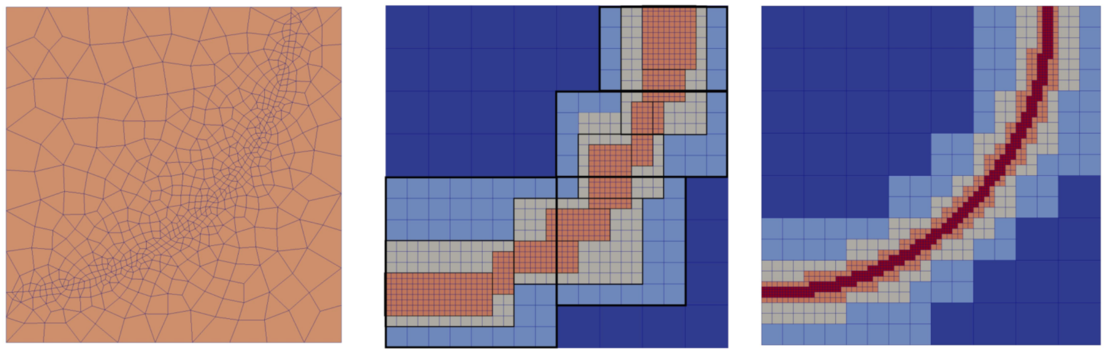


Figure 2.1: Three different AMR methods for refining a quadrilateral mesh along a circular arc. **Left**: unstructured AMR, **middle**: block-structured AMR and **right**: tree-based AMR.
The colouring for block-structured and tree-based AMR refers to the refinement level of the elements. By courtesy of Johannes Holke taken from [13].

applications originated in the mid-80s [19]. The idea of running AMR in parallel came up around 15 years later in the late 90s [20].

Different AMR methods are displayed in Figure 2.1. All three meshes in Figure 2.1 are refined along a circular arc. On the left side, an unstructured mesh, consisting of quadrilaterals and triangles, is refined via unstructured AMR. The middle mesh consists only of quadrilaterals and is refined via block-structured AMR. The different patches are highlighted with thick lines.

Lastly, the mesh on the right side, consisting only of quadrilaterals, is refined via tree-based AMR. The colouring for block-structured and tree-based AMR shows the refinement level of the elements. While all nodes within the unstructured mesh on the left are connected to the neighbouring elements, the structured meshes have so-called hanging nodes. These do not align with the nodes of the adjacent elements. Meshes with hanging nodes are called non-conforming. When using AMR with non-conforming meshes, the numerical solver has to be able to handle them properly, e.g. [21, 22, 23].

In the following, unstructured AMR and tree-based AMR will be described in more detail because of their relevance to this thesis. Block-structured AMR does not play a further role and will therefore not be regarded moving on.

2.1.1 Unstructured adaptive mesh refinement

Unstructured AMR is characterized by arbitrary connection relations between the mesh elements. That means no underlying structural rule prescribes how elements are connected. Meshes are usually conforming in unstructured AMR. The degree of adaptability to any geometry is high with unstructured AMR meshes, which is why the method is often used when the domain is complex [24].

Besides the high adaptability of unstructured AMR meshes, the requirement of memory to store all elements, especially when the mesh is rather big, is higher than of, e.g. tree-based AMR. This increased memory demand is because it is necessary to store the coordinates of each node of the refined mesh separately and the neighbour of every element. Even though it is possible to partition these meshes on multiple processes, their runtime is usually orders of magnitude slower when compared to the partition of structured meshes [24, 25].

Generally, triangle and tetrahedral elements are used in unstructured AMR. With the Delaunay triangulation method, it is possible to gain a high degree of conformation between geometry and the generated mesh [26, 27]. However, there are also unstructured meshing applications using quadrilateral or hexahedral elements [28].

2.1.2 Tree-based adaptive mesh refinement

Tree-based AMR is based on so-called refinement trees. These data structures contain a parent element and all children elements up to the highest refinement level. The root of a refinement tree is the coarsest element. Generally, these are the elements of the initial mes, which is also referred to as a forest of trees. Therefore, the initial mesh is often referred to as the coarse mesh. Assuming a coarse element is a quadrilateral, it can be refined to four sub-quadrilaterals on the first refinement level. Figure 2.2 shows a refined quadrilateral on the left-hand side. On the right-hand side, the according refinement tree is shown.

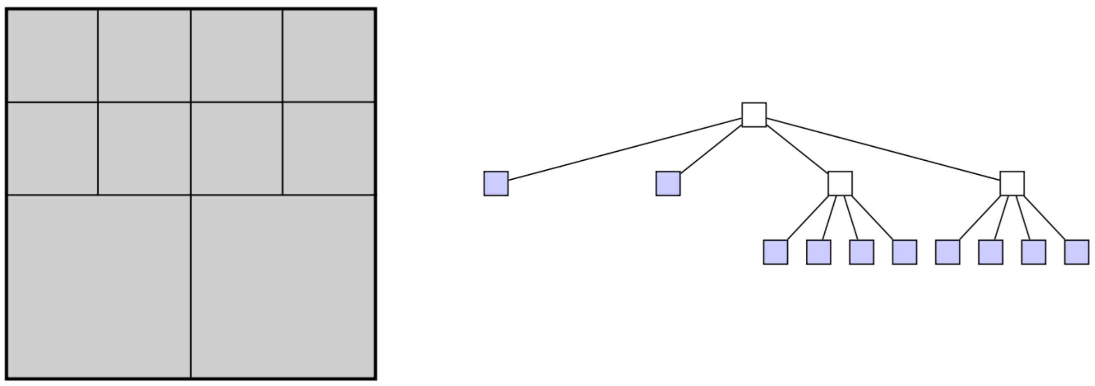


Figure 2.2: A refined quadrilateral and the conforming refinement tree. The coarse element is first refined into four sub-quadrilaterals. Then, the top two elements are refined once more. By courtesy of Johannes Holke taken from [13].

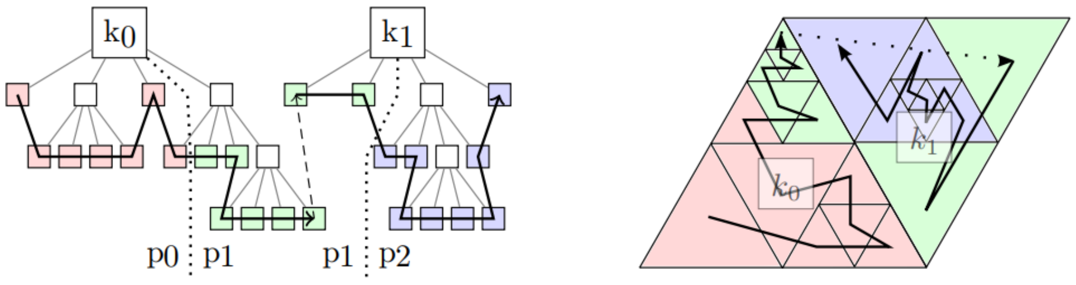


Figure 2.3: Two triangle elements *k0* and *k1* with a tree-based adaptive refinement. The space filling curve (SFC) stores all elements of both trees one dimensional. Additionally, the mesh is spread over three different processes *p0*, *p1* and *p2*, which are represented by different colours. By courtesy of Johannes Holke taken from [13].

The blue elements in Figure 2.2, which are the finest elements of their branch, are then stored in a one-dimensional array. One-dimensional arrays are very effective in memory usage, and elements and their neighbouring elements can be found relatively quickly compared to unstructured AMR.

The order in which the elements are stored in the one-dimensional array is dictated by a so-called SFC, implemented for every element shape. Figure 2.3 shows how the SFC works. The minor elements on their respective branch of the refinement tree are stored linearly over the borders of different refinement trees and processes. More on the advantages of this form of memory usage can be found in [13, 29]. Tree-based AMR is the adaptive refinement method used in `t8code`.

2.2 Boundary representation - Geometry representation in `OpenCASCADE`

Geometries can be generated with several software systems or geometry libraries. Besides big commercial CAD software like AutoCAD, Inventor or SolidWorks, many open-source geometry libraries are free to use and often provide more low-level functionalities for the users. That makes them less intuitive in usage but just as powerful as a commercial CAD software because of the ability to use the functions in personal projects or development tasks. There are Turf, Cgal or OCCT, to name a few.

The following focuses on OCCT because `t8code` currently wraps the geometry management functions of the software. Hence, the geometry representation of OCCT will be presented in more detail. Due to the size and complexity of the geometry library, only the relevant aspects of this thesis will be discussed.

Because of the overlapping nomenclature, when discussing geometries and meshes (faces, surfaces, edges, curves, vertices, points), the geometry topologies will have OCCT as a prefix. The new prefix applies outside of Section 2.2 because of the high usage of geometry topology names in this section.

Shapes are represented via their boundary in OCCT. A boundary representation (B-rep) is contrary to a cell decomposition, where the shapes are not described by their outer limitations but by cells filled with the material the solid is composed of [30]. A shape contains topological as well as geometrical components. Topological components are vertices Λ, edges E and faces K. They hold their geometrical components like point λ, curves ε and surfaces κ [30].

The topological components describe the relations between geometrical components. In B-rep, a volume is defined by a closed set of faces K. Every point inside the closed group can be distinguished by every point outside of it, forming a volume or a solid. Accordingly, a face K is bounded by edges E, which then are determined by vertices Λ.

Geometrical components are the spatial and visible members of the geometry representation. They are parametric, so the respective parameters can describe every location on the geometrical component. A point λ is of dimension zero and has no parameters because its coordinates can fully define it. A curve ε is of dimension one and is represented by a map

$$\varepsilon\colon \Omega_\varepsilon := [u_{\min},\, u_{\max}] \to \mathbb{R}^3, \qquad u \mapsto \varepsilon(u) \tag{2.1}$$

where the curve domain Ω_ε is described by a range of the curve parameter u. A surface is of dimension two and is also represented by a map

$$\kappa\colon \Omega_\kappa := [u_{\min},\, u_{\max}] \times [v_{\min},\, v_{\max}] \to \mathbb{R}^3, \qquad (u,v) \mapsto \kappa(u,v) \tag{2.2}$$

where the surface domain Ω_κ is described by a range of the surface parameters u and v. Volumes or solids are of dimension three, but because of the B-rep, they are not a map and do not have a geometry. The range of parameters u for curves ε and u and v for surfaces κ is limitless and depends on the geometry. For example, the parameters for a quadrilateral surface κ represent the two axes of the surface κ, so combining both parameters can form a description of every point on the surface κ. For a cylindrical surface κ, the parameters depend on a cylindrical coordinate system with the u-parameter having a period of 2π and the v-parameter defining the axial position.

The OCCT library has a broad range of different and more complex geometries. Besides curves and surfaces, there are also parabolas, circles, planes, spheres, B-spline curves and B-spline surfaces, to name a few. To gain an overview of the many more algorithms OCCT provides for handling geometries, looking into the documentation [31] is helpful. Note that this section is based on [15].

2.3 Advection solver of `t8code`

To evaluate the algorithms researched in this thesis, using a solver capable of using `t8code` managed meshes is necessary. Therefore, `t8code` comes with its own solver for advection equations. This solver will evaluate the curved triangular and tetrahedral AMR. The solver already exists and was developed by Dr. Johannes Holke. Therefore, the section is based on [13, 15].

2.3.1 Advection equation and implementation in the finite volume method

The general advection equation describes how a quantity ϕ is advected with a given flow u over time t. The advection PDE is

$$\frac{\partial \phi}{\partial t} + \nabla \cdot (\phi \vec{u}) = 0 \tag{2.3}$$

which simplifies to equation (2.4) with the assumption that the flow u is divergence-free

$$\frac{\partial \phi}{\partial t} + \vec{u} \cdot \nabla \phi = 0. \tag{2.4}$$

To solve the PDE shown in equation (2.4) on `t8code` managed meshes, a FVM of polynomial degree zero is used, providing first-order convergence rates. Although the flow u could be the discrete solution of a fluid solver, the `t8code` advection solver assumes that u is given analytically. The advection equation is to be integrated over every volume of the mesh. Additionally, the Gauss divergence theorem with volume V, a continuously differentiable vector field defined on a neighbourhood of volume V, the outward pointing normal vector $\vec{n}$ and a piecewise smooth boundary S

$$\iiint_V (\nabla \cdot F) \mathrm{d}V = \iint_A (F \cdot \vec{n}) \mathrm{d}A \tag{2.5}$$

is applied. The resulting equation (2.6) reads as follows:

$$\frac{\partial}{\partial t} \iiint_V \phi(\vec{x},\, t) \mathrm{d}V + \iint_A (\phi(s,\, t)\vec{u}(s,\, t) \cdot \vec{n}(s)) \mathrm{d}A = 0. \tag{2.6}$$

The domain $\Omega \in \mathbb{R}^3$ can now be discretized into elements E with i faces ξ. Furthermore, the constant value $\phi_{E,\, t}$ holds true for each element E and can therefore approximate $\phi(\vec{x},\, t)$. The flux $\psi(E,\, E'_j;\, \xi_j)$ going through face ξ_j into the neighbouring element E'_j, with j being the index of the neighbouring element, is also introduced. The magnitude of the flux is determined by the flow $\vec{u}_{\xi_j,\, t}$ at the midpoint of the face ξ_j and the area A of the face. The flux is always orthogonal to the face ξ_j and points outwards of element E towards element E'_j

$$\psi(E,\, E'_j;\, \xi_j) = \begin{cases} \phi_{E,\, t}\vec{u}_{\xi_j,\, t} \cdot \vec{n}(\xi_j)\mathrm{A}(\xi_j) & \text{for } \vec{u}_{\xi_j,\, t} \cdot \vec{n}(\xi_j) \geq 0 \\ \phi_{E'_j,\, t}\vec{u}_{\xi_j,\, t} \cdot \vec{n}(\xi_j)\mathrm{A}(\xi_j) & \text{for } \vec{u}_{\xi_j,\, t} \cdot \vec{n}(\xi_j) < 0. \end{cases} \tag{2.7}$$

In most numerical applications, conforming meshes are used. They do not have hanging nodes on edges or surfaces, so the flux from one element to another does not split up. In `t8code`, the tree-based refinement of elements leads to hanging nodes at element boundaries where the refinement level differs by at least one.

Therefore, the flux on a face ξ_j with hanging nodes must be calculated with the area and flow of the sub-faces ξ'_j. For simplicity reasons, splitting the flux over two or more sub-faces ξ'_j will not be represented in the following equations. Nevertheless, the `t8code` advection solver can deal with hanging nodes.

We can now apply the flux $\psi(E,\, E'_j;\, \xi_j)$ to equation (2.6) and get

$$\frac{\partial}{\partial t}\iiint_E \phi_{E,\, t}\mathrm{d}V + \sum_{j=0}^{i-1}\iint_{\xi_j}(\phi_{E,\, t}\vec{u}_{\xi_j,\, t} \cdot \vec{n}(\xi_j))\mathrm{d}A = 0. \tag{2.8}$$

Finally, the discretization in time has to be respected by using the difference quotient and the volume V of an element

$$0 = \mathrm{V}(E)\frac{\phi_{E,\, t+\Delta t} - \phi_{E,\, t}}{\Delta t} + \sum_{j=0}^{i-1}\psi(E,\, E'_j;\, \xi_j). \tag{2.9}$$

We can transform equation (2.9) and yield

$$\phi_{E,\, t+\Delta t} = \phi_{E,\, t} - \frac{\Delta t}{\mathrm{V}(E)}\sum_{j=0}^{i-1}\psi(E,\, E'_j;\, \xi_j). \tag{2.10}$$

Equation (2.10) is the final advection equation formulated to be solved by a FVM. For every time step Δt the equation is solved for every element E and every face ξ of that element.

2.3.2 Courant-Friedrichs-Lewy number

The Courant-Friedrichs-Lewy (CFL) number is a parameter of the `t8code` advection solver, and it can influence the convergence behaviour of the simulation. The inventors and eponyms of the CFL number state that a finite difference equation is unable to converge with CFL > 1 [32]. Originally, the CFL number indicates how many identical rectangular elements E an event with the velocity v can pass in a given time step Δt:

$$\mathrm{CFL} = \Delta t \frac{v}{\mathrm{len}(E)}. \tag{2.11}$$

We have to adapt this original version of the CFL number because our elements E are not always identical, they are not all rectangular, and the flow $\vec{u}_{E,\,t}$ differs magnitude depending on t and Ω. The adapted version of equation (2.11) is defined for every element at each time step separately, with $\mathrm{V}(E)$ being the volume of element E and $\mathrm{V}(E)^{1/d}$ resembling the mean diameter of the element E of dimension d.

$$\mathrm{CFL}_{E,\,t} = \Delta t \frac{|\vec{v}_{E,\,t}|}{\mathrm{V}(E)^{1/d}}. \tag{2.12}$$

As mentioned before, the CFL number has to be smaller than one for the simulation with the `t8code` advection solver to converge. To ensure stability, a CFL number way lower than one should be used because of the varying size of elements during the AMR process.

2.3.3 Refinement criterion of the `t8code` advection solver

The advection equation given by equation (2.10) is solved for a domain Ω by the `t8code` advection solver. There are two domains in an advection application: the advected concentration $\Omega_{1,\,t=0}$ and the remaining space $\Omega_{2,\,t=0}$. The whole domain therefore consists out of both sub domains $\Omega_{1,\,t} \cup \Omega_{2,\,t} = \Omega$. An element inside or outside the advected concentration can be detected by the element's $\phi_{E,\,t}$ value. The initial condition of the advected concentration, therefore, is

$$\phi_{E,\,0} = \begin{cases} \mathrm{dist}(\overline{\Omega_{1,\,0}} \cap \overline{\Omega_{2,\,0}}) & \text{for } E \in \Omega_{1,\,0} \\ -\mathrm{dist}(\overline{\Omega_{1,0}} \cap \overline{\Omega_{2,\,0}}) & \text{for } E \in \Omega_{2,\,0}, \end{cases} \tag{2.13}$$

with $\mathrm{dist}(\overline{\Omega_{1,\,0}} \cap \overline{\Omega_{2,\,0}})$ being the distance of the element E from the interface between both sub domains. Hence, both subdomains $\Omega_{1,\,0}$ and $\Omega_{2,\,0}$ can be distinguished by their sign during the simulation.

Following equation (2.13), the values of $\phi_{E,\,t}$ inside the advected concentration get increasingly smaller the closer an element is to the centre of domain $\Omega_{1,\,t}$ and increasingly larger the further an element is away of the boundary of both domains inside domain $\Omega_{2,\,t}$. Based on this behaviour of $\phi_{E,\,t}$ the refinement criterion can be defined by

$$|\phi_{E,\,t}| < hb, \tag{2.14}$$

where the parameter $h = \mathrm{V}(E)^{1/d}$ defines the average diameter of an element of dimension d and the parameter b controls the distance of the refinement around the interface. The parameter b is referred to as the thickness of the refinement band.

3 Algorithm development

This chapter will focus on the research and development of an algorithm for high-order triangular or tetrahedral elements. First, the objective and specifications of the algorithm will be described. Different possible algorithms will be presented and assessed to find the most fitting, according to criteria stated in the following section. Additionally, the algorithm has to fit the current structure of meshes with geometrical information in `t8code`. More details on meshes with geometrical data can be found in Section 2.1. After an algorithm is selected, the implementation of that algorithm into `t8code` will be described for triangular and tetrahedral elements.

3.1 Objective of the algorithm

`t8code` can already handle high-order curved meshes. The ability is currently limited to quadrilateral and hexahedral elements. Regarding all other two-dimensional (triangles) and three-dimensional (tetrahedra, prisms, pyramids) element shapes, it becomes apparent that they all have at least one triangular face. Therefore, researching an algorithm covering triangular shapes is the first step in developing high-order mesh support for all element shapes, with the possibility of using curved hybrid meshes.

To understand the algorithm's requirements for triangular element shapes, looking into the existing algorithm for quadrilaterals is reasonable. The current algorithm was researched and developed by Sandro Elsweijer in 2022 [15].

As an external library for AMR mesh management, `t8code` imports a linear parametric mesh in *msh* format 4.x [33] and the *brep* file containing the geometrical information. Every mesh element (vertex, edge, face) which lies on a geometry has information about the parameters of that geometry. With the additional information about each geometry component from the *brep* file, `t8code` has all the information to curve the linear mesh elements to the geometry domain.

A schematic visualisation of the mapping and curving of linear mesh elements can be seen in Figure 3.1. The figure shows how a reference point $[0, 1]^2$ is mapped linearly to the global space $\mathbb{R}^2$. Subsequently, the displacement between the linear element edges and the OCCT curves ε of the quadrilateral element are evaluated at the points where the reference coordinate lies. The displacement gets scaled so that reference points, which lay directly on the linear element edge

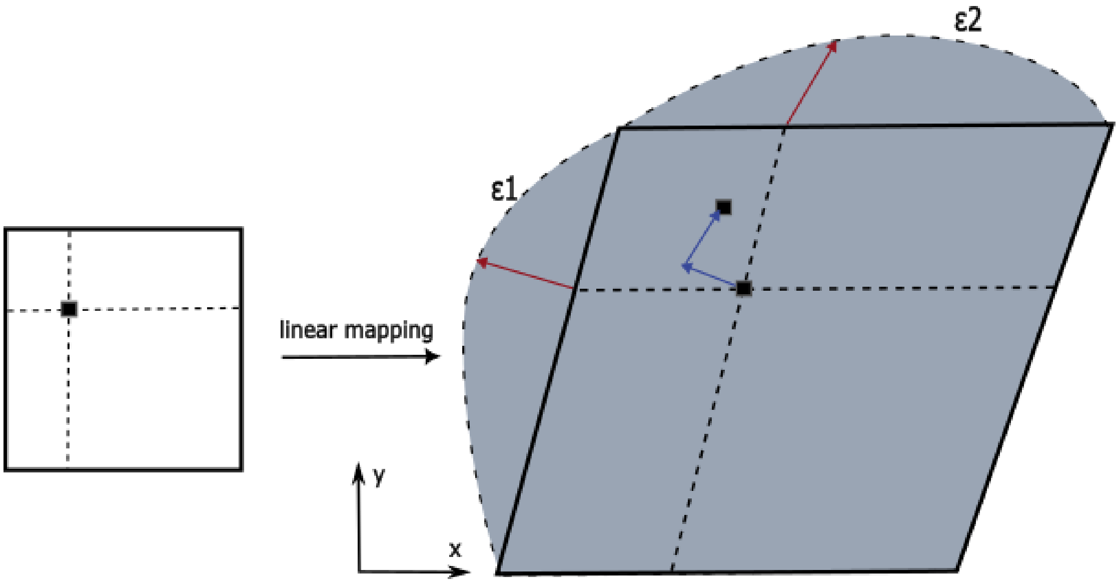


Figure 3.1: Mapping of a reference coordinate from the unit square (left) to the global space (right) with subsequent evaluation of the displacement from linear element edges to OCCT curves ε. After the displacement is computed, it is added onto the mapped reference coordinate with appropriate scaling. The figure is inspired by [15].

linked to a OCCT curve ε are shifted with the total displacement. Points, which lay on an edge without linkage against an OCCT curve ε are not displaced at all. The reference point shown in Figure 3.1 is therefore shifted proportionately with the displacement from the vertical and the horizontal edges to the linked OCCT curves ε.

For simplicity reasons, Figure 3.1 only shows the displacement evaluation of the quadrilateral edges. If the face of the element is also linked against a OCCT surface κ, the same evaluation of the displacement from the linear face to the OCCT surface κ with the appropriate scaling across the face applies.

The scaling on quadrilateral faces is relatively straightforward because it is only necessary to check the orthogonal coordinates from the OCCT curve ε or surface κ and to a linear interpolation of the displacement with the coordinates value. The same applies to hexahedral elements because the orthogonality upholds from two to three dimensions. This orthogonality is missing when evaluating triangular shapes. Therefore, finding an algorithm capable of accurately scaling displacement over the whole element without using orthogonal coordinates is crucial.

Generally, the desired algorithm should create a mesh representing the geometry near perfectly. That is, because under refinement of an element new nodes are created on the curved edges and faces. Hence, a potential error in the geometry representation grows with the refinement level.

Arbitrary reference points will be generated by `t8code`, so the algorithm should produce exact results, no matter the position inside the element's volume.

Lastly, the algorithm should fit inside the software structure of `t8code`. Currently, the algorithm for curved quadrilaterals and hexahedra is embedded inside a function, which evaluates the element based on the element class (quadrilateral or hexahedron). The same function should be expanded to cover triangles and tetrahedra. This way, `t8code` can deal with curved hybrid meshes in two dimensions because the evaluation and curving of an element happens for each element separately concerning the element class. Curved hybrid meshes are not yet implemented for three dimensional meshes.

3.2 Mapping approaches

This Section presents the research and development of the mapping algorithm for triangular mesh elements. The focus lies on triangles rather than tetrahedra because the general concept behind the algorithm for triangles also applies to tetrahedral elements.

First, different approaches will be covered before the actual algorithm is selected. After describing the objective of the algorithm we present how the algorithm is implemented in `t8code`. The additional challenges for tetrahedra will also be addressed in Section 3.4.

3.2.1 Approach 1: Polynomial basis functions

High-order mesh generation often relies on polynomial basis functions. The approximation properties of polynomials are excellent, especially for smooth functions. In terms of accuracy, high-order approximation with polynomials proves advantageous over low-order approximation in terms of degrees of freedom and even computational cost [34].

However, to gain a certain accuracy, it might be necessary to use polynomials of high order, which increases the computational demand. Hence, polynomial high-order approximation is powerful with sufficient initial resolution but proves less robust with coarse initial meshes.

One possibility to make polynomial high-order approximation more robust is to tailor finite element basis functions to the problem. This idea has been covered in numerous previous works for different solving methods [35, 36, 37, 38]. After reviewing the literature on high-order curved meshes with a polynomial approach, the paper of Sanjaya and Fidkowski [39] stood out. The authors present a method using tailored basis functions and reference-to-global mapping to achieve high-quality, high-order curved meshes. As shown in Figure 3.1, `t8code` uses a similar kind of reference-to-global mapping.

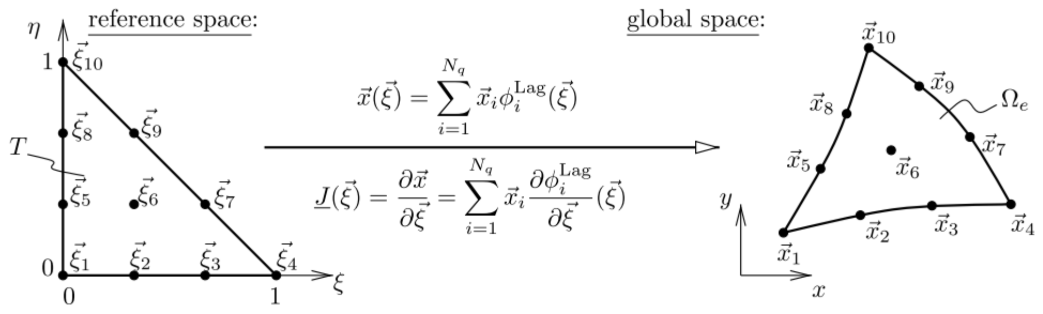

reference space:
global space:
$\vec{x}(\vec{\xi}) = \sum_{i=1}^{N_q} \vec{x}_i \phi_i^{\mathrm{Lag}}(\vec{\xi})$
$\underline{J}(\vec{\xi}) = \frac{\partial \vec{x}}{\partial \vec{\xi}} = \sum_{i=1}^{N_q} \vec{x}_i \frac{\partial \phi_i^{\mathrm{Lag}}}{\partial \vec{\xi}}(\vec{\xi})$
Ω_e
T
η
ξ
x
y

Figure 3.2: Example of a $q = 3$ mapping from a reference triangle to a curved element in global space. Figure taken from [39].

Figure 3.2 shows how points from the reference space $[0,1]^2$ with the coordinates η and ξ are mapped to a curved triangle element in global space $\mathbb{R}^2$ with the coordinates x and y). The mapping function is

$$\vec{x}(\vec{\xi}) = \sum_{i=1}^{N_q} \vec{x}_i \phi_i^{Lag}(\vec{\xi}) \tag{3.1}$$

where $\vec{x}(\vec{\xi})$ is the coordinate in global space in dependence of the coordinate in reference space $\vec{\xi}$. N_q is the total number of degrees of freedom in the mapping, with q being the polynomial order. $\phi_i^{Lag}(\vec{\xi})$ is the Lagrange basis function.

Even though the mapping presented by Sanjaya and Fidkowski would be implementable in `t8code`, the decision falls against the approach. As stated in Section 3.1, one requirement of the desired algorithm is for the curved mesh to be as true to the actual geometry as possible.

Although polynomials with a high order can achieve high accuracy, they will, in most cases, still approximate the exact geometry and, therefore, have a slight deviation. By the approach of equipping a coarse, linear mesh with geometrical information, as described in Section 1.3 and 2.2, mesh elements can be curved precisely to the geometry without having to use high order polynomials, which increase the computational demand.

3.2.2 Approach 2.1: Displacement scaling along the angle bisector of the triangle

The searched-for approach should be able to curve triangular mesh elements to an arbitrary geometry with near-perfect geometrical accuracy. As mentioned in the previous Section, this is only possible if the algorithm has access to the information about the geometry the element was generated on. The mechanism to read external meshes into `t8code` can equip every mesh component (vertex, edge, face) with the parameters of the geometry it lies on. An example of how

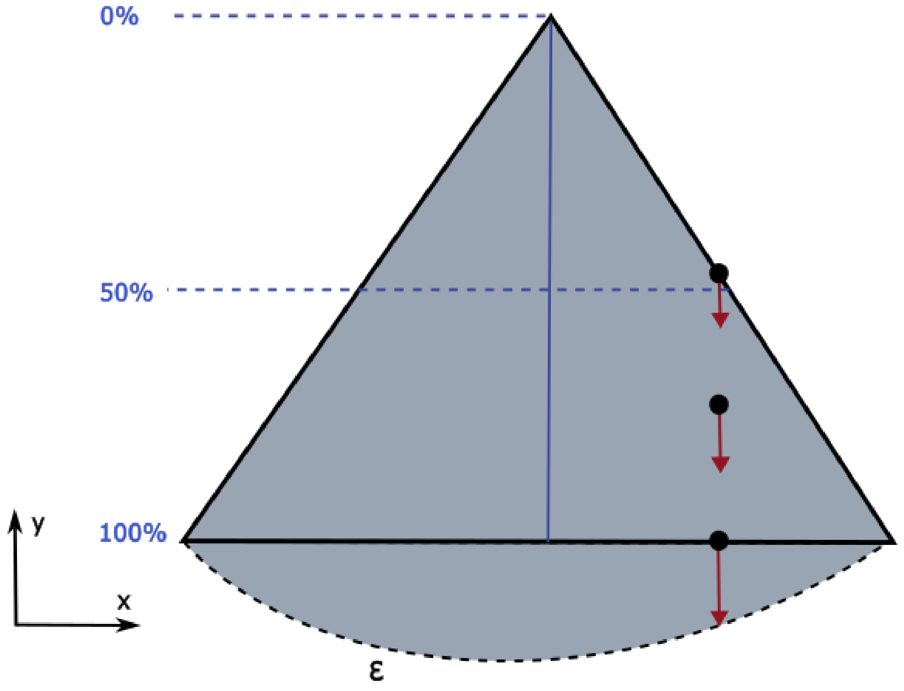


Figure 3.3: Scheme of edge displacement scaling along the angle bisector of the neighbouring edges to the edge linked against an OCCT curve ε.

this information is used to curve edges and faces of elements to the actual geometry is presented in Section 3.1.

The challenge of curving triangular mesh elements with the algorithm schematically presented in Figure 3.1 is to find a suitable scaling factor. Without a scaling factor, the displacement of an edge or a face of an element would be propagated throughout the whole element, resulting in a distorted element, which will probably have no connection to the neighbouring elements besides the vertices.

The first approach of the present thesis to finding a suitable scaling factor is to use the length of the angle bisector from the opposite vertex of an edge. This approach is shown in Figure 3.3. An OCCT curve ε is linked to a triangle element edge. The displacement between the OCCT curve ε and the linear edge of the triangle is calculated at the point of the reference coordinate inside the triangle. It is visualised by the red arrow in the figure. The computed displacement is then scaled with the length of the angle bisector of the angle between both neighbouring edges of the edge with the linked OCCT edge ε. Whilst the scaling factor is at one right on the edge, it decreases linearly to the opposite vertex until it reaches 0. This scaling method is equal to the technique for quadrilaterals, but instead of the orthogonal of the edge reaching the opposite edge in the quadrilateral, the orthogonal points to the opposite vertex in a triangle.

The presented scaling method is unsuitable for triangles because of the violated boundary conditions. The boundary condition is that the displacement on the edge, which is linked to an OCCT curve ε, is scaled with 100 percent of the displacement, and both neighbouring edges should have a scaling factor of 0.

Figure 3.3 shows the resulting problem of the boundary condition violation. The displacement, more to the right of the angle bisector, is scaled around 40 percent when it reaches the neighbouring edge. That means the point right on the adjacent edge would be shifted, as shown in Figure 3.1, resulting in a wrongly curved element.

The idea of using the angle bisector is, therefore, not suitable for the scaling within the algorithm. However, the concept works for points lying right on the angle bisector. The following Subsection will present an extension of this idea.

3.2.3 Approach 2.2: Vertical and horizontal displacement scaling

The problem with the scaling approach using the angle bisector, presented in Subsection 3.2.2, is that the scaling only takes the orthogonal direction of the edge into account. While the scaling works for points right on the angle bisector, the deviation gets increasingly more prominent the

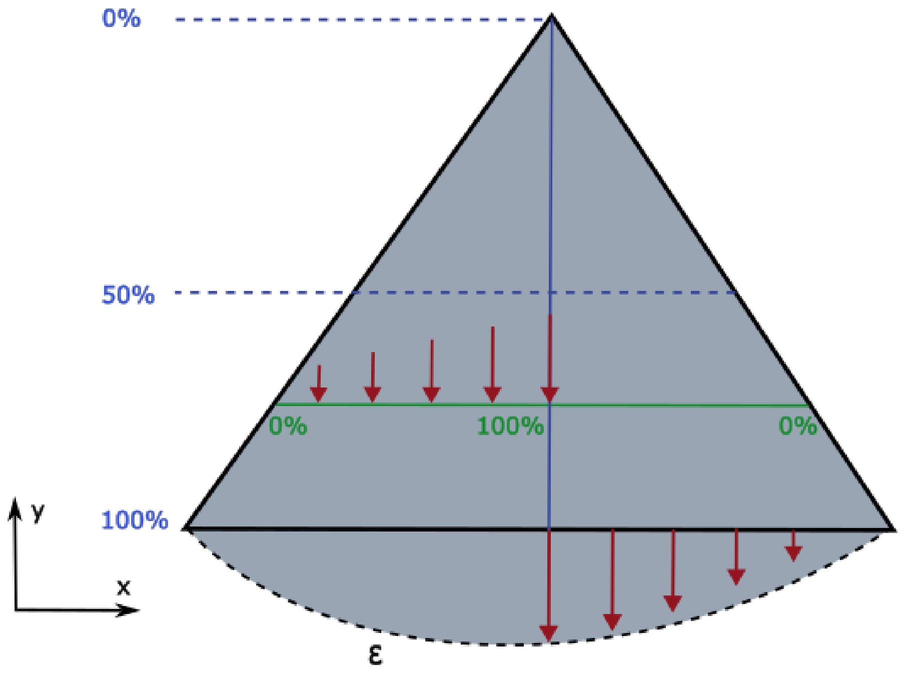


Figure 3.4: Scheme of edge displacement scaling along the angle bisector of the edges opposite vertex and the element's width at the reference coordinate's height.

further away the reference coordinate is from the angle bisector. Hence, an additional horizontal scaling would be helpful.

With the horizontal scaling, it is possible to stay true to the boundary condition of zero percent scaling on the neighbouring edges of the edge linked against the OCCT curve ε. Figure 3.4 shows how this scaling would look like. Besides the vertical scaling along the angle bisector (blue), a horizontal scaling (green) is introduced. The horizontal scaling is at 100 percent right on the angle bisector and decreases to both sides until it reaches the edges, where the scaling is at 0 percent. The horizontal scaling happens concerning the width at the height of the reference coordinate.

Even though the additional horizontal scaling improves the scaling introduced in Subsection 3.2.2, it is also unsuitable for a robust triangular curving algorithm. While the boundary conditions on the neighbouring edges with 0 percent displacement are correct now, the boundary condition on the edge linked against the OCCT curve ε is not valid any more. Every displacement of a point on the edge should be scaled 100 percent to curve the edge exactly to the OCCT curve ε. That is true for the point on the edge right where the angle bisector intersects the edge.

Again, the deviation becomes more prominent when moving towards the edge's vertices. That is shown in Figure 3.4 with the displacement arrows (red) on the bottom edge. The displacement should be scaled with 100 percent, which means that the arrows point right on the OCCT curve ε. Because of the horizontal mapping, the displacement gets scaled down when moving away from the angle bisector.

The approach with vertical and horizontal scaling proves advantageous over the method only regarding the angle bisector. Still, it must be more suitable to reach a near-perfect geometry representation. Therefore, a fourth approach has to be developed.

3.2.4 Approach 2.3: Non-orthogonal displacement scaling

Both approaches described in Subsections 3.2.2 and 3.2.3 relied on the orthogonal direction of the linked edge. That leads to violations of the boundary conditions set to the curving algorithm for triangles.

Because of the missing orthogonality in a triangle, it is necessary to take a step back from these approaches and find a way to scale the displacement throughout the element without relying on orthogonality and the angle bisector of the triangle.

The furthest point from a linked edge is the opposite vertex of the triangle element. Therefore, the scaling has to reach this opposite vertex to have a scaling throughout the whole element. The other end of the scaling straight was the projection along the coordinates from the reference point onto the linked edge. A reference point of $(0.6, 0)$ would mean that the edge displacement is calculated at $x = 0.6$ if the edge is parallel to the x-axis. By using the projection along the x- and

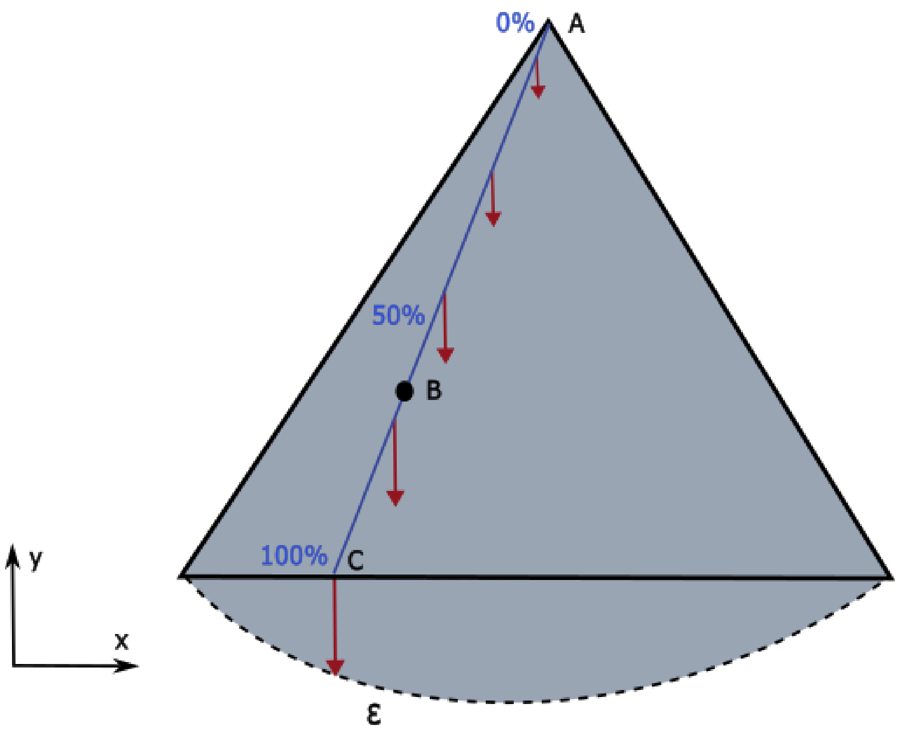


Figure 3.5: Scheme of edge displacement scaling along the projection of the reference point from the opposite vertex onto the linked edge.

y-axis, the problems with boundary violation mentioned above occur. Hence, the new approach, presented in Figure 3.5, does not use the value of the reference point coordinates to calculate the displacement of the edge but the intersection point of a straight line from the opposite vertex, through the reference coordinate and onto the linked edge.

The figure shows an arbitrarily chosen reference point at approximately $(0.3, 0.4)$. A straight line is drawn from the opposite vertex of the linked edge onto the edge through the reference point. Afterwards, the displacement between the linear edge and the linked OCCT curve ε is calculated at that intersection point. The scaling factor γ for the displacement at the reference point can then be calculated as follows

$$\gamma = \overrightarrow{AB}/\overrightarrow{AC}, \tag{3.2}$$

with the opposite vertex A of the linked edge, the reference point B and the intersection point C of the line and the linked edge. With equation (3.2), every reference point on the linked edge will get scaled with a hundred percent of the displacement at that point because the reference point is equal to the intersection point in that case. The distance between the two is zero if the reference point lies on the opposite vertex. Therefore, the scaling factor is also zero. In conclusion, the displacement at reference points close to the linked edge will get scaled with values near hundred percent, and the displacement at reference points close to the opposite vertex will get scaled with values near zero percent.

Now, the boundary conditions have to be checked. It was already shown that all reference points on the linked edge receive a scaling factor of 1. Hence, this boundary condition is fulfilled. The other boundary condition is that reference points on both neighbouring edges will not be displaced. This boundary condition is fulfilled, even though the scaling factor for reference points on the adjacent edges is not zero, because the intersection point on the linked edge is, in all cases, one of the vertices. The displacement at the vertices between the linear edge and the OCCT curve ε is always zero, because initial mesh nodes are generated exactly on the geometry.

The scaling algorithm for triangle elements is suitable and fulfils the requirements stated in Section 3.1 and the previous Subsections. This algorithm will be the basis for developing the curving algorithm for triangle and tetrahedra elements. The implementation in `t8code` will be presented in the following Section.

3.3 Implementation for triangular elements

The algorithm approach, researched and developed in Section 3.2, will now be implemented in `t8code`. The implementation is presented in this Section. The focus is less on how the code is written precisely than on the general structure and flow. The developed code can be observed in [40].

To follow the implementation structure, understanding how `t8code` calls the functions which contain the curving algorithms is necessary. The first step is to read a file containing the initial mesh, also known as coarse mesh, into `t8code` and create its own data structure, called a cmesh. The file has to be a *msh* file, which mesh generators like *Gmsh* can export. If mesh elements should be curved the reader takes a *brep* file with the same name as the *msh* file into account. Reading a *msh* file containing parametric mesh elements and a *brep* file with the parametric geometry description, `t8code` has all the information needed to evaluate each element geometry.

Evaluating an element in `t8code` means creating strategically placed reference coordinates in the reference space of the current element. The reference coordinates are then mapped linearly to the element domain $\mathbb{R}^3$. This procedure happens for linear meshes as well as curved meshes. The appropriate curving algorithm is then called if the mesh should be managed as a curved mesh in `t8code`. The curving algorithm uses the information of the parametric mesh and the parametric geometry to shift the linearly mapped reference coordinates Λ_in according to the element's curvature and returns the shifted reference coordinates Λ_out.

Triangle elements can be linked differently to an OCCT geometry. It has to be concerned, that edges E, faces K or both can be linked to an OCCT geometry. The curving algorithm must cover all cases to ensure the highest possible geometrical accuracy. Mesh vertices are not linked to OCCT geometries because they are created on the geometry by the mesh generator and do, therefore, not have to be shifted by the curving algorithm. Mesh edges E can be linked to OCCT curves ε or to OCCT surfaces κ. If an edge is already linked to an OCCT surface κ, the mesh reader of `t8code` locks the linking of OCCT curves ε to the edge E. An edge E can, therefore, never be linked to an OCCT curve ε and an OCCT surface κ at the same time. Lastly, the face K of a triangle element can only be linked to OCCT surfaces κ.

```
Input: Reference point $\Lambda\_in$ in reference space $[0,1]^2$
       Information about the element tree
Result: Shifted reference point $\Lambda\_out$ in domain $\mathbb{R}^3$
1  for each face K do
2      if face K is linked against OCCT surface $\kappa$ then
3          Get the surface parameters of the current face K;
4          Calculate the surface parameter in the domain $\mathbb{R}^3$ by interpolation with the
             reference point $\Lambda\_in$;
5          if edge $E\_face$ of face K is linked against OCCT curve $\varepsilon$ then
6              Calculate the intersection point (see Subsection 3.2.4) in reference space;
7              Map the intersection point linearly from reference space to the domain $\mathbb{R}^3$;
8              Get the curve parameter at the intersection point by linear interpolation;
9              Convert curve parameters to surface parameters;
10             Get the surface parameter at the intersection point by bi-linear interpolation;
11             Calculate the scaling factor for the surface parameter displacement (see
                 equation (3.2));
12             Calculate the actual surface parameter displacement and multiply with the
                 scaling factor;
13         end
14         Calculate the point on the OCCT surface $\kappa$ with the scaled surface parameter
             displacement;
15         Set that point on the OCCT surface $\kappa$ to be the shifted reference point $\Lambda\_out$
16     end
17 end
```

Algorithm 1: t8_geom_evaluate_occ_triangle: The algorithm evaluates triangular element faces to curve them to a linked OCCT surface κ.

Algorithm 1 shows the pseudocode for the evaluation of triangle element faces by the function `t8_geom_evaluate_occ_triangle`, which contains the curving algorithm for triangles. The algorithm shows how the parameters of the OCCT geometries are used to calculate points on the curved geometries, which are then used to calculate the displacement between the linear and the curved element.

One caveat of the face evaluation is that the face displacement can not be calculated right at the reference point Λ_in if the boundary edges of the face are also curved. That is, because the curvature of the edge already shifts the reference point. In this case, the surface parameters must be shifted before the face displacement can be calculated. This happens in line 5 in algorithm 1. In line 11 the scaling factor for the edge displacement gets calculated according to equation (3.2), with the intersection point calculated in line 7. After the right point on the OCCT surface κ, concerning the curvature of the edges, is found, the face displacement can be calculated. Afterwards, the linear reference point Λ_in gets shifted with the face displacement to produce the output reference point Λ_out.

After evaluating the face K of curved triangle elements, the evaluation of the edges E follows.

Algorithm 2 shows the pseudocode for evaluating edges E of curved triangle elements. The evaluation of the edges E was already described in the if statement in line 5 of algorithm 1. As mentioned earlier, edges E can be linked to OCCT curves ε as well as OCCT surfaces κ. It is therefore necessary to either calculate the displacement of the linear edge E to the OCCT curve ε or to the OCCT surface κ. After this distinction is made, the displacement can be calculated accordingly. The displacement has to be scaled again with the scaling algorithm described in Subsection 3.2.4 until the shifted reference point Λ_out can be returned by the function.

In combination, the two algorithms 1 and 2 cover all cases of linkage to a triangle element. After evaluating the faces K and the edges E, the element is curved precisely to the geometry. From this point on, `t8code` can manage the mesh just as a linear mesh, with all the advantages of the AMR library.

3.4 Implementation for tetrahedral elements

This Section will present the implementation of the curving algorithm for tetrahedral elements. Because of the additional dimension from triangles to tetrahedra, it is necessary to address some differences to the triangles presented in Sections 3.2 and 3.3.

In the following, the scaling of displacements generated by OCCT curves ε and OCCT surfaces κ will be addressed separately because the algorithm handles them individually. The evaluation and scaling of tetrahedral edges will be presented firstly. While the displacement from the linear edge to the OCCT curve ε is mapped over the face of a triangle, a tetrahedron has a volume and each edge has two neighbouring faces. Hence, generating two scaling factors, one for each neighbouring face, is necessary to scale the edge displacement properly.

```
   Input: Reference point $\Lambda\_in$ in reference space $[0,1]^2$
          Information about the element tree
   Result: Shifted reference point $\Lambda\_out$ in domain $\mathbb{R}^3$
 1 for each edge E do
 2     if edge E is linked against OCCT curve $\varepsilon$ or against OCCT surface $\kappa$ then
 3         Get the parameters (curve or surface) of the current edge E;
 4         Calculate the intersection point (see Subsection 3.2.4) in reference space;
 5         Map the intersection point linearly from reference space to domain $\mathbb{R}^3$;
 6         if edge E is linked against OCCT curve $\varepsilon$ then
 7             Get the curve parameter at the intersection point by linear interpolation;
 8             alculate the point on the OCCT curve $\varepsilon$ with the interpolated curve
                 parameter;
 9         end
10         if edge E is linked against OCCT surface $\kappa$ then
11             Get the surface parameters at the intersection point by bi-linear interpolation;
12             Calculate the point on the OCCT surface $\kappa$ with the interpolated surface
                 parameters.
13         end
14         Calculate the scaling factor for the displacement (see equation (3.2));
15         Calculate the displacement of the global intersection point to the point on the
             OCCT curve $\varepsilon$ or the OCCT surface $\kappa$;
16         Scale the calculated displacement with the scaling factor;
17         Add the scaled displacement onto the reference point $\Lambda\_in$ in domain $\mathbb{R}^3$ and set
             the result to be the shifted reference point $\Lambda\_out$;
18     end
19 end
```

Algorithm 2: t8_geom_evaluate_occ_triangle: The algorithm evaluates triangular element edges to curve them to a linked OCCT curve ε or anOCCT surface κ.

Figure 3.6 shows how the edge displacement generated by the edge linked against the OCCT curve ε, is scaled on the neighbouring faces $K1$ and $K2$. The boundary conditions for the scaling on the adjacent faces stay the same. That means the edge displacement right on the linked edge should be scaled with a hundred percent, while the scaling factor on the other two edges on each neighbour face should be zero.

Two values are saved to achieve a scaling that conforms to those boundary conditions. The first value is the orthogonal coordinate of the edge at the projected reference point coordinate on the respective face K. This value is represented with the red arrows in Figure 3.6. The second value is the maximum of the orthogonal coordinate at the reference point on the neighbouring face K.

This value is represented with the blue arrow in Figure 3.6. The scaling factor γ is determined with the equation

$$\gamma = 1 - (\Lambda\_K/\Lambda\_Kmax), \tag{3.3}$$

it is calculated by the relation of the projected orthogonal reference coordinate Λ_K on the neighbouring face K and the maximum orthogonal coordinate Λ_Kmax on the same face K.

Now that the scaling of edge displacement for tetrahedral elements is described, the algorithm for tetrahedral edge evaluation can be presented.

Algorithm 3 shows the pseudocode for how the edge displacement is evaluated and scaled to curve a tetrahedron to a given OCCT geometry. Again, just as in algorithm 1 and 2, a reference point Λ_in is given to the function, which evaluates the linkage against OCCT geometries to output a shifted reference point Λ_out. Edges E of a tetrahedron can be linked against OCCT curves ε or OCCT surfaces κ. The distinction is made via the parameters the linked edge E possesses.

As mentioned before, OCCT curves ε have a parameter u, while OCCT surfaces κ have two parameters u and v. The displacement is calculated by the difference of the linked edge E to the calculated point on the OCCT curve ε or OCCT surface κ. Following, the displacement gets scaled over the neighbouring faces K_neigh of edge E with the scaling factor presented in equation (3.3).

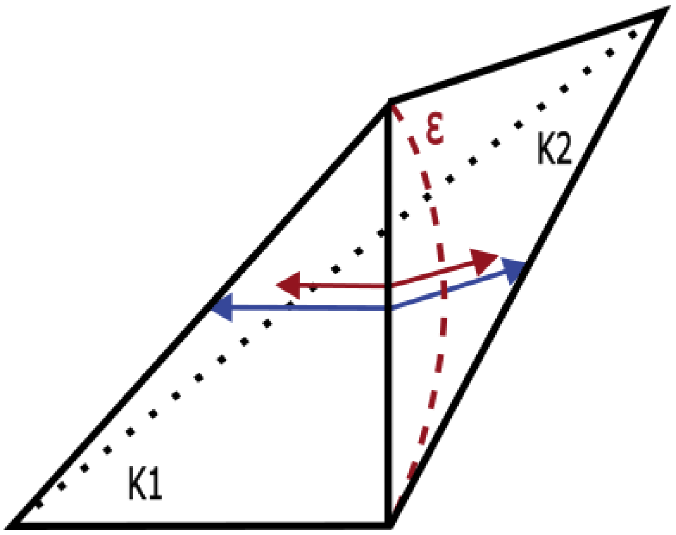


Figure 3.6: Scheme of edge displacement scaling on the neighbouring faces $K1$ and $K2$ of the edge linked against an OCCT curve ε in the reference tetrahedron. The red arrow represents the reference coordinate projected onto the face K, and the blue arrow represents the maximum orthogonal direction at that point on the face K.

```
    Input: Reference point $\Lambda\_in$ in reference space $[0,1]^3$
           Information about the element tree
    Result: Shifted reference point $\Lambda\_out$ in domain $\mathbb{R}^3$
 1  for each edge E do
 2      if edge E is linked against OCCT curve $\varepsilon$ or against OCCT surface $\kappa$ then
 3          Save the coordinates of the vertices of edge E;
 4          Interpolate linearly between the vertex coordinates to the the coordinates of the
              reference point $\Lambda\_in$ projected on the edge E;
 5          Get the parameters (curve of surface) of the current edge E;
 6          if edge E is linked against OCCT curve $\varepsilon$ then
 7              Get the curve parameter at the point of the reference point $\Lambda\_in$ projected on
                  the edge E by linear interpolation;
 8              Calculate the point on the OCCT curve $\varepsilon$ with the interpolated curve
                  parameter;
 9          end
10          if edge E is linked against OCCT surface $\kappa$ then
11              Get the surface parameters at the point of the reference point $\Lambda\_in$ projected
                  on the edge E by bi-linear interpolation;
12              Calculate the point on the OCCT surface $\kappa$ with the interpolated surface
                  parameters;
13          end
14          Calculate the displacement of the reference point $\Lambda\_in$ projected on the edge E
              to the point on the OCCT curve $\varepsilon$ or the OCCT surface $\kappa$;
15          Calculate the scaling factors for both neighbouring faces K_neigh of edge E (see
              equation (3.3));
16          Scale the calculated displacement with the scaling factors on both neighbouring
              faces K_neigh;
17          Add the scaled displacement onto the reference point $\Lambda\_in$ coordinates in the
              domain $\mathbb{R}^3$ and set the result to be the shifted reference point $\Lambda\_out$;
18      end
19  end
```

Algorithm 3: t8_geom_evaluate_occ_tet: The algorithm evaluates tetrahedral element edges to curve them to a linked OCCT curve ε or an OCCT surface κ.

Now, the evaluation of tetrahedral faces will be presented. Tetrahedra possess a third dimension, so it is now necessary to scale the calculated displacements from edges E or faces K through the element volume. That is done equally compared to triangle elements, presented in Section 3.2. Instead of a straight line drawn from the opposite vertex of an edge to the intersection point on the edge (see Figure 3.5), the straight line gets drawn from the opposite vertex of a face to an intersection point on the face. The intersection point on the face is the projection of a reference point inside the tetrahedron volume to the face, which is evaluated. For triangles that was not

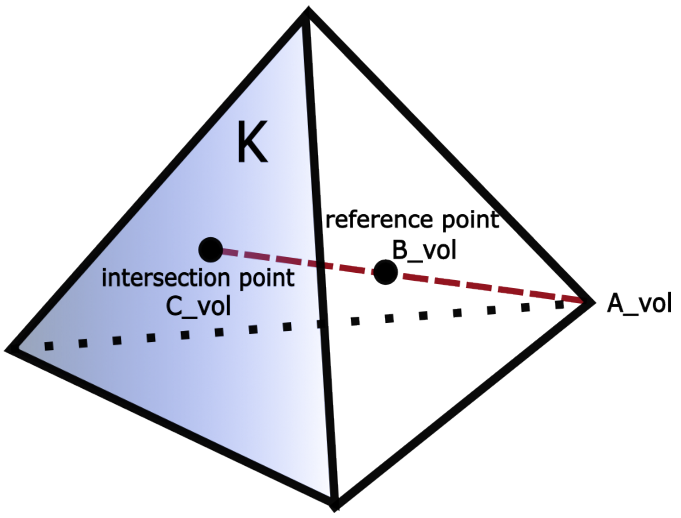


Figure 3.7: Scheme of the generation of an intersection point on the face K, by the projection of the reference point from the opposite vertex of face K inside the tetrahedron.

necessary, because every reference coordinate has to lie right on the face. For tetrahedron that is not true, so the reference point has to be projected to each face in order to evaluate it.

Figure 3.7 shows how the intersection point on face K is generated by the projection of the reference point from the opposite vertex of face K inside the tetrahedron. After the intersection point is generated, the displacement of the face can be evaluated at that point. To scale the calculated face displacement through the volume of the tetrahedron, equation 3.2 is used. Instead of A being the opposite vertex of an edge (see Figure 3.5), A_vol represents the opposite vertex of the linked face K, on which the intersection point lies. The reference point is represented by B_vol, and the intersection point, this time on the face and not on the edge, is represented by C_vol. This results in equation 3.4, which is a slight modification of equation 3.2 and calculates the scaling factor γ throughout the volume of the tetrahedron.

$$\gamma = \overrightarrow{A\_volB\_vol} / \overrightarrow{A\_volC\_vol}. \tag{3.4}$$

The algorithm for the face evaluation of tetrahedral elements can be presented with the modification described above.

```
   Input: Reference point $\Lambda\_in$ in reference space $[0,1]^3$
          Information about the element tree
   Result: Shifted reference point $\Lambda\_out$ in domain $\mathbb{R}^3$
1  for each face $K$ do
2      if face $K$ is linked against OCCT surface $\kappa$ then
3          Calculate the intersection point (see Figure 3.7) on the face $K$ in reference space;
4          Get the surface parameters of the current face $K$;
5          for each edge $E\_face$ of face $K$ do
6              if edge $E\_face$ is linked against OCCT surface $\kappa$ then
7                  Get the curve parameters of the edge $E\_face$;
8                  Get the curve parameter at the intersection point projected on the edge
                     $E\_face$ by linear interpolation;
9                  Get the vertex coordinates of the edge $E\_face$;
10                 Get the coordinates of the intersection point projected on the edge
                     $E\_face$ by linear interpolation;
11                 Calculate the scaling factors for both neighbouring faces $K\_$neigh of
                     edge $E\_face$ (see equation (3.3));
12                 Calculate the displacement of the edge scaled with the scaling factors of
                     both neighbouring faces $K\_$neigh and save the result in
                     face_displacement_from_edges;
13             end
14         end
15     end
16     Calculate the surface parameters at the intersection point in domain $\mathbb{R}^3$ by
         interpolation with the intersection point in reference space;
17     Correct the intersection point on face $K$ with displacement form the edges $E\_face$
         face_displacement_from_edges;
18     Calculate the point on the OCCT surface $\kappa$ with the surface parameters at the
         intersection point in domain $\mathbb{R}^3$;
19     Calculate the scaling factor of the face displacement through the element volume
         (see equation (3.4));
20     Calculate the face displacement between the intersection point on face $K$ and the
         point on the OCCT surface $\kappa$ at the same location and scale it with the scaling
         factor;
21     Add the scaled face displacement onto the reference point $\Lambda\_in$ in domain $\mathbb{R}^3$ and set
         the result to be the shifted reference point $\Lambda\_out$;
22 end
```

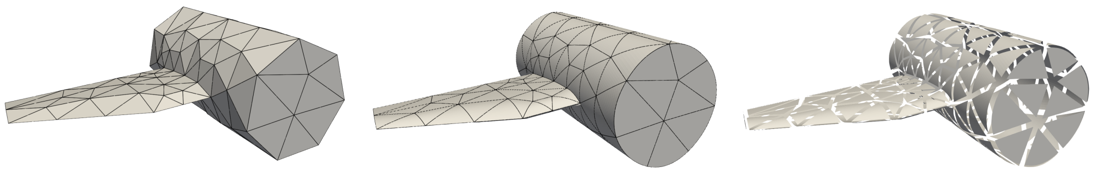
Algorithm 4: t8_geom_evaluate_occ_tet: The algorithm evaluates tetrahedral element faces to curve them to a linked OCCT surface κ.

Algorithm 4 contains the pseudocode for the face evaluation of tetrahedra. The order of the edge and face evaluation is essential. First, the edges are evaluated inside the function `t8_geom_evaluate_occ_tet`. That is important because the scaling of the edge displacement over the neighbouring faces impacts the intersection point (see Figure 3.7) on them. The curvature of the edges result in a shift of the reference point projection, before the face displacement can be evaluated.

After the intersection point on a face K is found, it is corrected with the displacement produced by linked edges E. This is described in the lines 4 to 17 in algorithm 4. After the intersection point is corrected, the face displacement is calculated by the difference of the updated intersection point on face K and the point on the OCCT surface κ at the exact location. The calculated face displacement is then scaled by the scaling factor presented in equation (3.4), and the result is added onto the reference point Λ_in shifting it according to the geometry curvature to output Λ_out.

3.5 Visualisation of the curving algorithms

After the researched and developed algorithms for curving triangle and tetrahedra elements are implemented in `t8code`, several arbitrary domains are meshed to check if any bugs or errors occur. A simplified section of an aircraft is used to test the implementation of curved triangle elements. The round fuselage and the aerodynamically shaped wing cover most of the curving algorithm's edge cases.

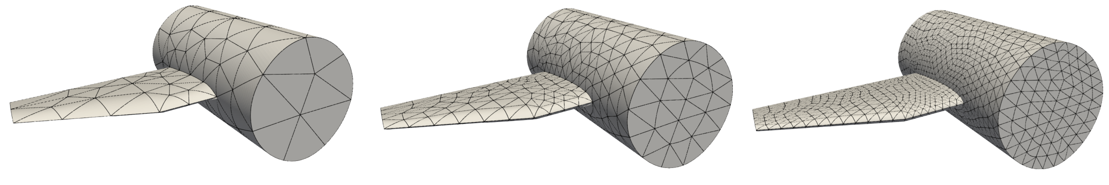


(a) Linear mesh **(b)** Curved mesh **(c)** Curved mesh (shrunk)

Figure 3.8: On the left in Figure **(a)**, the linear mesh of an aircraft section, meshed with linear triangle elements on the surface, is shown. The middle Figure **(b)** shows the same mesh, but the curving algorithm for triangle elements is applied, curving the elements exactly to the geometry. The curved mesh is visualised on the right with shrunk elements in Figure **(c)**, proving that triangle elements are used.

Figure 3.8 shows three versions of the aircraft section mentioned above. In all three cases, the aircraft surface is meshed with triangle elements. The linear mesh is visualised on the left in Figure 3.8a. The geometry representation is relatively poor, evident at the front. The front surface is meshed with only nine elements, resulting in a poor representation of the circle shape. The same inaccuracies can be found on the rest of the fuselage and the wing.

The curved mesh in Figure 3.8b has the same number of elements but provides a near perfect geometrical representation of the aircraft section. That shows that the curving algorithm fulfils its requirements. Figure 3.8c shows the same mesh as the previous figure but with shrunk elements. That indicates that the mesh only consists of triangle elements on the surface of the aircraft section.

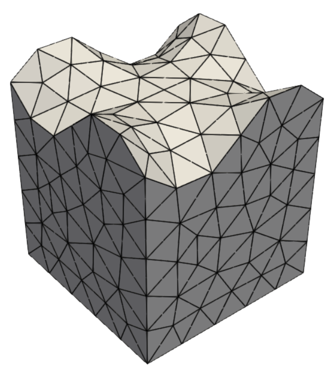


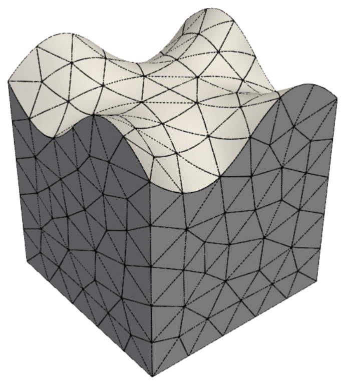
(a) Refinement level 0 **(b)** Refinement level 1 **(c)** Refinement level 2

Figure 3.9: The global refinement of the curved triangle mesh (see Figure 3.8) in two steps is shown. At each step, a refinement of 1:4 of the elements from the previous step is carried out.

It is necessary to prove that the curved elements can be adequately refined to check if the curving algorithm is viable in an AMR environment like `t8code`. Figure 3.9 shows three steps of refinement starting with the coarsest mesh on the left side in Figure 3.9a. Figure 3.9b in the middle is the first global refinement step, refining each element 1:4. Lastly, the second refinement step, with another refinement of 1:4 of each element, is shown in Figure 3.9c.

Now, the algorithm for tetrahedral elements has to be checked as well. While tetrahedral elements can also discretise the aircraft section, another geometry is chosen. As mentioned before, a face and an edge of an element can not be linked to the same OCCT geometry. For triangle meshes, edges are rarely linked to an OCCT surface because this can only happen if the triangle lies on the OCCT surface. If the triangle lies on the OCCT surface, the face is linked to it, automatically blocking the linkage of the edges to the same OCCT surface. That is different for tetrahedra. Because tetrahedral meshes not only mesh the surface of a domain but also the volume, it is possible for a tetrahedron edge to be linked against an OCCT surface without any face of the element lying on the same OCCT surface. The chosen geometry to check curved tetrahedra covers more of the described cases. The geometry is a cube with one face being a $B-spline$ surface.


(a) Linear mesh


(b) Curved mesh

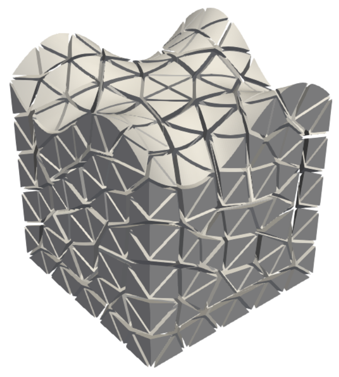
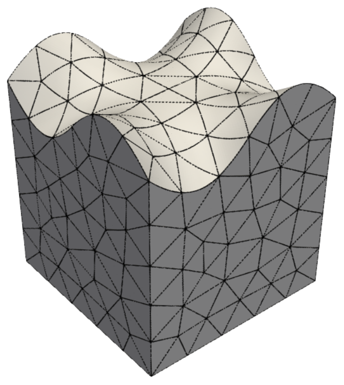
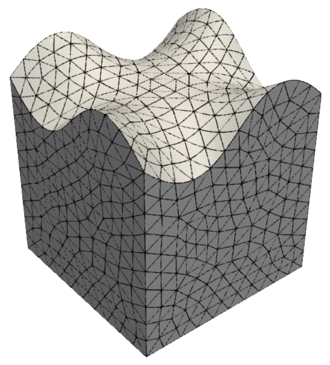

(c) Curved mesh (shrunk)

Figure 3.10: On the left in Figure **(a)**, the linear mesh of a cube with a $B-spline$ surface, meshed with linear tetrahedral elements, is shown. The middle Figure **(b)** shows the same mesh, but the curving algorithm for tetrahedral elements is applied, curving the elements exactly to the geometry. On the left, in Figure **(c)**, the curved mesh is visualised with shrunk elements.

Figure 3.10 shows three versions of the tetrahedral mesh of the cube. The left Figure 3.10a shows the linear mesh, which provides a relatively poor geometry representation on the $B-spline$ face of the cube. Figure 3.10b shows the same mesh after applying the curving algorithm. Without changing the number of elements used for the discretisation, the geometry representation is massively improved. The right Figure 3.10c shows the same mesh but with shrunk elements to prove that the mesh consists of tetrahedra and is not a triangle surface mesh.


(a) Refinement level 0


(b) Refinement level 1

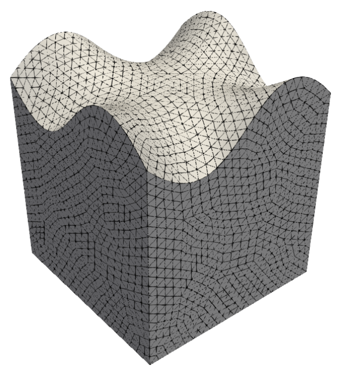

(c) Refinement level 2

Figure 3.11: The global refinement of the curved tetrahedral mesh (see figure 3.10) in two steps is shown. At each step, a refinement of 1:8 of the elements from the previous step is carried out.

Accordingly to the curved triangle mesh refinement in Figure 3.9, the curved tetrahedral elements must also be checked for proper refinement. Figure 3.11 shows three refinement stages.

At each step, a refinement of 1:8 elements happens. Tetrahedral meshes are much more intensive regarding computational resources than triangle meshes in an AMR application like `t8code`. Not only do they consist of more elements when the same geometry is meshed with the same element size because of the volume discretisation, but they also have a larger refinement ratio. The amount of elements increases with a 1:8 refinement, compared to a 1:4 refinement for triangles.

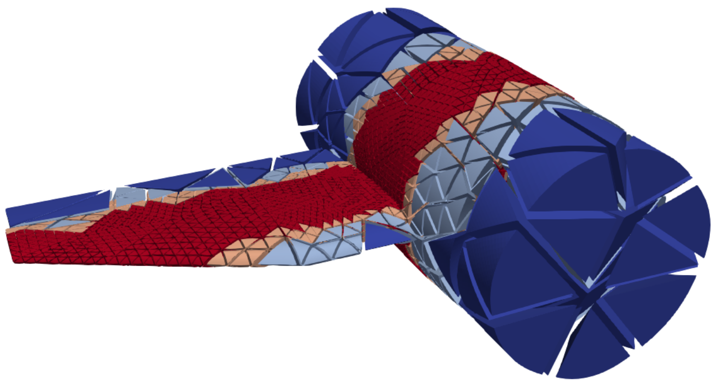


Figure 3.12: The figure shows the aircraft section, before used for the visualisation of the curved triangles meshes. The model is meshed with curved tetrahedral elements and a refinement band of refinement level 3 is situated at the depth of the wing. The different colours show the refinement levels of each element.

Figure 3.12 shows that it is also possible to mesh the aircraft section from Figures 3.8 and 3.9 with curved tetrahedral elements. The Figure is the middle time step of a dynamic simulation of a refinement band passing along the axis of the aircraft with a refinement level of three.

4 Evaluation

After the algorithms for curving triangle and tetrahedral elements to arbitrary geometries have been presented in detail and their implementation into `t8code` in Chapter 3, it is now time to evaluate the algorithms.

Hence, an application scenario will be presented in Section 4.1. The application scenario contains an airfoil generated by the Joukowsky transform, which will be presented in Section 4.2. Afterwards, the simulation setup is described in detail in Section 4.3 to ensure the reproducibility of the results, which are then presented in Section 4.4.

4.1 Application scenario

There is no limit to the geometry used to evaluate and validate the curved triangle and tetrahedral elements because, in both cases, an unstructured mesh will be created. Hence, the flow around an airfoil will be the application scenario used to assess and validate the curving algorithms for triangle and tetrahedral elements. There will be a two-dimensional geometry and mesh to evaluate triangle elements and a three-dimensional mesh to evaluate tetrahedral elements. The three-dimensional domain will extrude the two-dimensional domain, expanding the same geometry by a constant third dimension.

To conduct the evaluation and validation, the advection solver presented in 2.3 will be used. As described in the respective section, the advection equation for each element is solved for a given geometry and analytical flow field. Therefore, generating an airfoil geometry for which an analytical flow field can be calculated is necessary. More information on that can be found in Section 4.2 and Subsection 4.2.2.

Before conducting a simulation, it is critical to define a criterion which functions as a quality indicator of the requirements of the curving algorithms. The main requirement of the algorithms is to curve each element precisely to the geometry. The definition of the analytical flow field around the airfoil prescribes that there is no in- or outflow of the domain. Hence, there should be no flux across the geometry boundaries. It is possible to track the volumes of the subdomains $\Omega_{1,\,t}$ and $\Omega_{2,\,t}$ over time to check whether this assumption is valid. A near-perfect geometri-

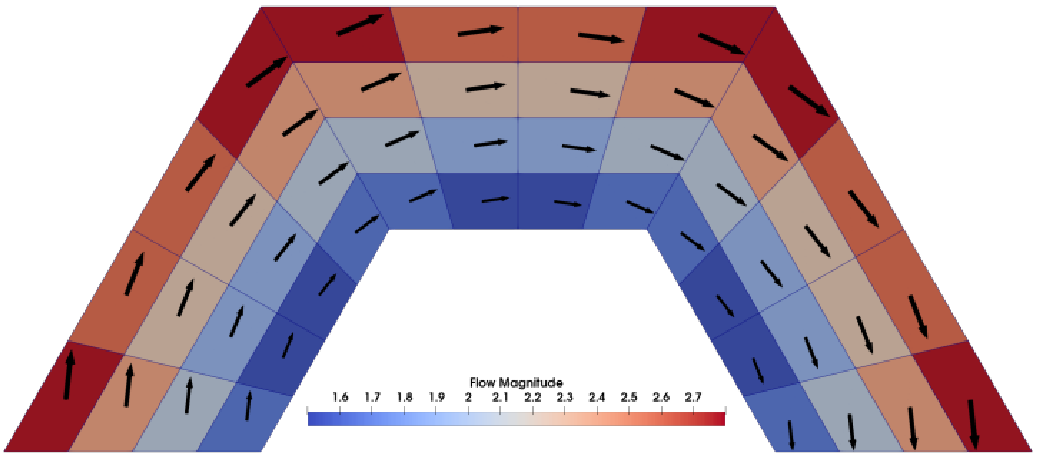


Figure 4.1: An analytically defined flow field around a circle with poor geometry resolution. In- and outflow occur on the corners of the mesh because the flow is parallel to the actual geometry and not the mesh boundary. Additionally, the magnitude of the flow is uneven due to the poor geometry resolution. By courtesy of Sandro Elsweijer taken from [15].

cal representation should result in a constant volume, while a specific volume loss should be detectable for coarse geometry resolution.

The volume loss occurs because the flow is parallel to the geometry. In case of a poor geometry resolution, the flow does not match the mesh, resulting in in- and outflow of the domain. Moreover, the flow magnitude is affected by a poor geometry resolution.

That is visible in Figure 4.1. A circle with a poor geometry resolution is shown in the figure. The flow field, represented by the black arrows, flows in a circular motion clockwise around the circle geometry. Because of the corners in the mesh, which result from the poor resolution, in- and outflow over the domain boundaries occurs. This results in an uneven flow magnitude. A volume loss would undoubtedly be trackable if a volume would be advected around the circle and the flow field, shown in Figure 4.1.

In summary, the criterion for the evaluation and validation of the curving algorithms is

$$\begin{aligned} V(\Omega_{1,\,t}) &= V(\Omega_{1,\,t+\Delta t}), \\ V(\Omega_{2,\,t}) &= V(\Omega_{2,\,t+\Delta t}), \end{aligned} \tag{4.1}$$

with the volume V of the subdomains $\Omega_{1,\,t}$ and $\Omega_{2,\,t}$. The criterion prescribes that the volume of the subdomains should stay constant between time steps.

4.2 Modelling

As mentioned in Section 4.1, the application scenario is the flow around an airfoil. How air flows around an airfoil depends on the shape, the velocity and the angle of attack. Calculating an analytical flow field is, therefore, not a simple task. With the Joukowsky transform [41], it is possible to create an airfoil geometry and calculate the analytical flow field around that specific geometry.

The theory behind the Joukowsky transform will be presented in the following Subsections 4.2.1 and 4.2.2. Furthermore, the actual specifications of the airfoil and flow field used for the simulations (see Section 4.3) will also be described.

4.2.1 Joukowsky airfoil geometry and mesh

Via the Joukowsky transform, it is possible to generate a two-dimensional airfoil geometry analytically. The Joukowsky transform is a conformal map from a circle in the complex ζ-plane to an airfoil geometry in the complex z-plane. With the equation

$$\zeta = z + \frac{1}{z}, \tag{4.2}$$

complex coordinates $\zeta = \chi + i\eta$ the target plane ζ gets transformed conformally into a complex coordinate $z = x + iy$ on the z-plane. With the mapping equation (4.2), the circle gets transformed into a broad range of aerodynamic profiles.

The shape of the profile is determined by the radius of the circle in the ζ-plane and its centre point. For radii $r > 1$ at the centre point $(0,0)$, symmetrical ellipses of different sizes result. The ellipses form into airfoil profiles with a shift of the centre point.

Figure 4.2 shows an example of an airfoil geometry produced by the transformation. The radius is chosen so that the point $(1,0)$ in the ζ-plane lies on the circle. That grants a sharp trailing edge of the airfoil geometry. One downside of the Joukowsky mapping is that it is conformal for every point ζ, except for points where the deviation of ζ equals zero. Those occur at $\zeta = \pm 1$, translating to the leading and trailing edges on the z-plane. These points constitute a singularity regarding the flow around the Joukowsky airfoil, which can be calculated analytically. The velocity of the flow is, therefore, unlimited at these points. More information can be found in Subsection 4.2.2.

It is necessary to create the geometry and the initial mesh in *Gmsh* to generate a Joukowsky airfoil geometry, which can be used by `t8code`. If the `T8_WITH_OCC` flag is passed to `t8code`, a *msh* and a *brep* file with the same name is read. While the *msh* file contains the information about the mesh points and the geometry parameters they lie on, the *brep* file contains the information

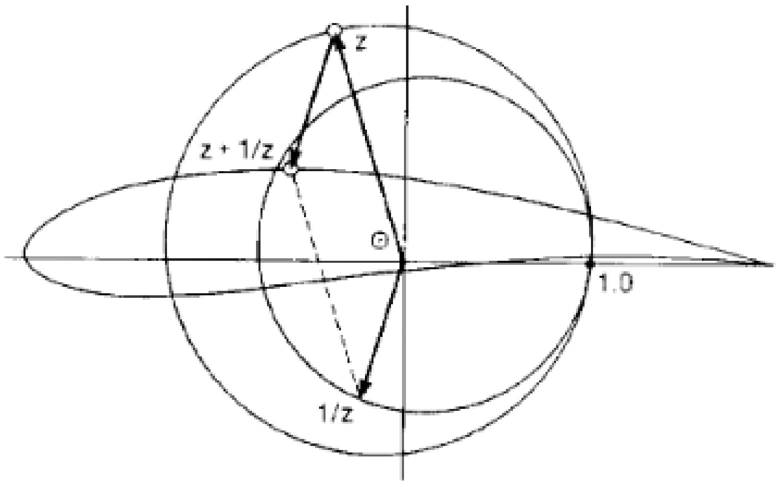


Figure 4.2: Visualisation of the Joukowsky transform from a circle to an aerodynamic profile with equation (4.2). Figure taken from [42].

about the geometry itself. To produce conforming *msh* and *brep* files, it is possible to use the *geo* scripting language of *Gmsh* to write a script, which produces both files according to user-defined criteria. As mentioned above, the airfoil shape is determined by the circle's centre point and radius in the ζ-plane. These are the only parameters necessary to construct the airfoil in two dimensions.

Before points of the airfoil in the z-plane can be generated, the circle in the ζ-plane has to be described geometrically. That is done by defining the circle as a function:

$$r = r(\phi). \tag{4.3}$$

With equation (4.3), it is now possible to generate n points, which describe the circle in two dimensions, with a delta of ϕ:

$$\Delta\phi = \frac{2\pi}{n}. \tag{4.4}$$

For the geometry used in this thesis, an amount of $x = 100$ point is used. The points are connected by B-splines later, so a higher number of points results in higher control over the B-spline shape and a more accurate geometry representation. Now, the centre point and the radius have to be determined. For the radius to pass through the point $(1,0)$ in the ζ-plane, resulting in a sharp trailing edge, the following equation can be used:

$$r = \sqrt{(1-\mu x)^2 + \mu y^2}. \tag{4.5}$$

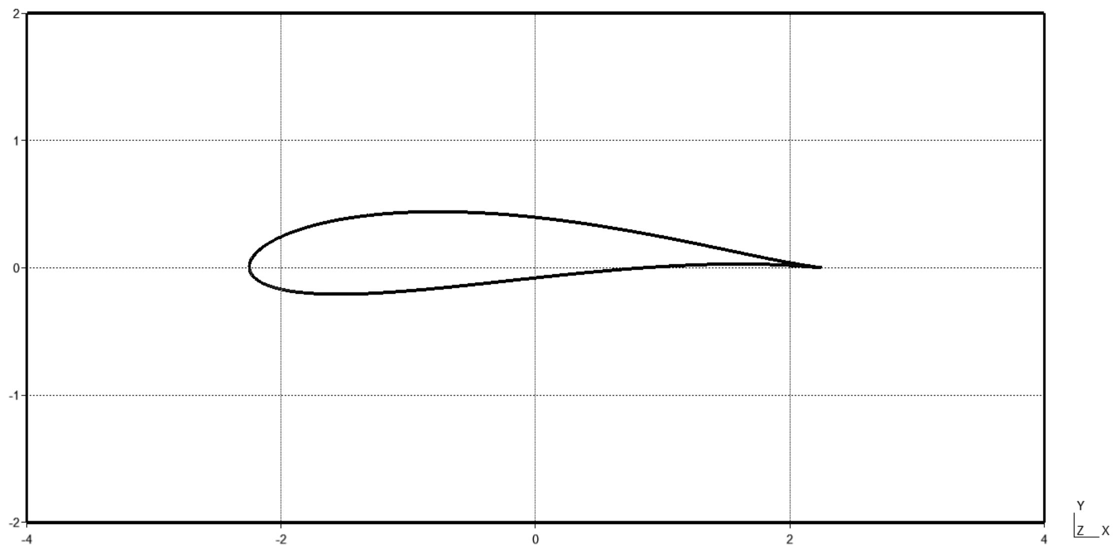


Figure 4.3: Visualisation of the two-dimensional Joukowsky airfoil used in this thesis. The figure shows the output of the *brep* file produced by the *geo* script with the parameters mentioned above.

In equation (4.5), μx and μy are the centre point coordinates. With this equation, the circle always passes through the point $(1,0)$, no matter where the centre point lies. The centre point used for this thesis lies at $\mu x = -0.12$ and $\mu y = 0.08$.

After the circle in the ζ-plane is constructed, the Joukowsky transform (equation (4.2)) is used to create the airfoil geometry. The circle always passes through the point $[2,0]$ on the z-plane at the trailing edge, and the leading edge will be at a x position lower than -2, due to equation (4.5). The leading and trailing edge locations are essential because of the flow field's singularities at $z = \pm 2R$. More information on that in Subsection 4.2.2.

The singularity at the leading edge lies outside the airfoil geometry and is, therefore, inside the flow field, resulting in a point for which the stream velocity gets infinitely large. On the other end of the airfoil geometry, the singularity lies right on the trailing edge. Therefore, the trailing edge is moved slightly to prevent any adverse effects of the singularity at that location. The airfoil geometry can be designed to be slightly larger than the airfoil used for the flow field to cancel the effects of the singularities. The singularities disappear inside the airfoil and, therefore, outside of the flow field.

To increase the size of the airfoil geometry, every coordinate on the z-plane gets multiplied by a factor of 1.1. The corner points of the domain around the airfoil are set to be $(-4,2)$, $(-4,-2)$, $(4,2)$ and $(4,-2)$. The resulting geometry is visualised in Figure 4.3.

Now that the two-dimensional geometry is defined, a second geometry in three dimensions has to be created. The evaluation should cover triangle as well as tetrahedral elements. The second

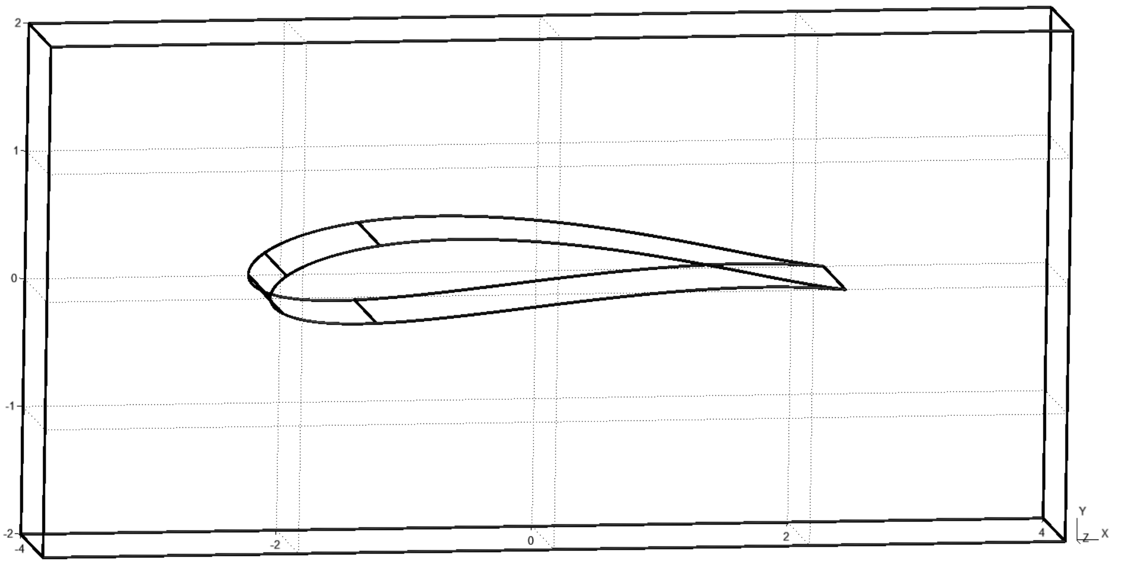


Figure 4.4: Visualisation of the three-dimensional Joukowsky airfoil used in this thesis. The figure shows the output of the *brep* file produced by the *geo* script with the parameters mentioned above.

geometry will equal the first one but extracted by $z = 1$. The resulting geometry is visualised in Figure 4.4.

After both geometries are constructed, generating an initial mesh for them is necessary. The respective *geo* files can define the meshing parameters. For the two-dimensional mesh, it is required to turn off the recombination in *Gmsh*. Recombination is the step in the meshing procedure where two triangular elements are combined to form a quadrilateral. Because of the curvature, especially around the airfoil's leading edge, it is helpful to control the mesh size according to curvature.

In *Gmsh*, the option `Mesh.MeshSizeFromCurvature` enables the user to automatically set the mesh size to a target number of elements per 2π radians [33]. The initial mesh has to be as coarse as possible to take advantage of the AMR provided by `t8code`. To compare curved elements to linear elements of different resolutions, the `Mesh.MeshSizeFromCurvature` option will be set to three for the initial mesh and will get increased for meshes with higher geometry resolution.

The two-dimensional meshing algorithm has to be selected as well. While the default algorithm for two-dimensional meshing in *Gmsh* is the `Frontal-Delaunay` algorithm, the standard `Delaunay` algorithm is chosen for this thesis. Delaunay triangulation aims to maximise the minimum of angles inside triangular elements [43]. While the `Frontal-Delaunay` algorithm tries to maximise the minimum angles from one advancing front throughout the domain, the standard `Delaunay` algorithm tries to maximise the minimum angles for the whole domain.

Even though the standard `Delaunay` algorithm is a little bit slower, it produces a better mesh in the case of the Joukowsky airfoil. The lower performance of the algorithm does not affect the runtime of the simulations after the initial mesh is created. As mentioned above, the initial mesh should be as coarse as possible, but elements curved to the geometry must not intersect themselves. Only if invalid elements are barred the initial mesh can be used.

For the three-dimensional mesh with tetrahedral elements, *Gmsh* uses the standard `Delaunay` algorithm by default. The initial coarse mesh will also have a `Mesh.MeshSizeFromCurvature` value of three. Later, the initial resolution of the tetrahedral mesh will also be steered by increasing that value.

The resulting triangle and tetrahedral element meshes can be seen in Figure 4.5. The triangular mesh in Figure 4.5a has 176 triangle elements, and the tetrahedral mesh in Figure 4.5b has 748 tetrahedra elements.

For more detail about the geometry construction and the meshing commands, it is recommended to take a look into the *geo* files, which are available on the *GitHub* of `t8code` [12].

4.2.2 Joukowsky airfoil flow field

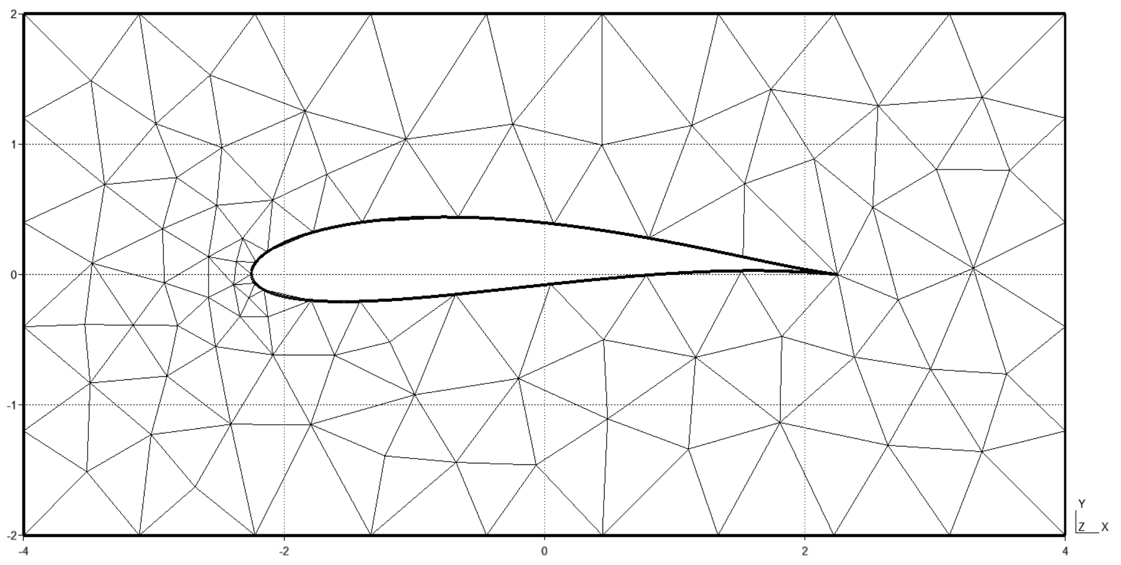
The potential flow around a circle or a cylinder can be calculated analytically. The potential flow of inviscid and incompressible fluids is well known and understood today [44, 45, 46]. With the calculation of the potential flow around a circle in the ζ-plane and the following application of the Joukowsky transform (see equation (4.2)), it is possible to receive the analytical potential flow around a Joukowsky airfoil.

A flow field is a vector field of a given domain. The vector field assigns a vector to each point in the domain, indicating the direction and velocity of a flow. Hence, it is possible to pass any point lying inside the domain to the flow field, and it will return the direction and velocity of the flow at that point.

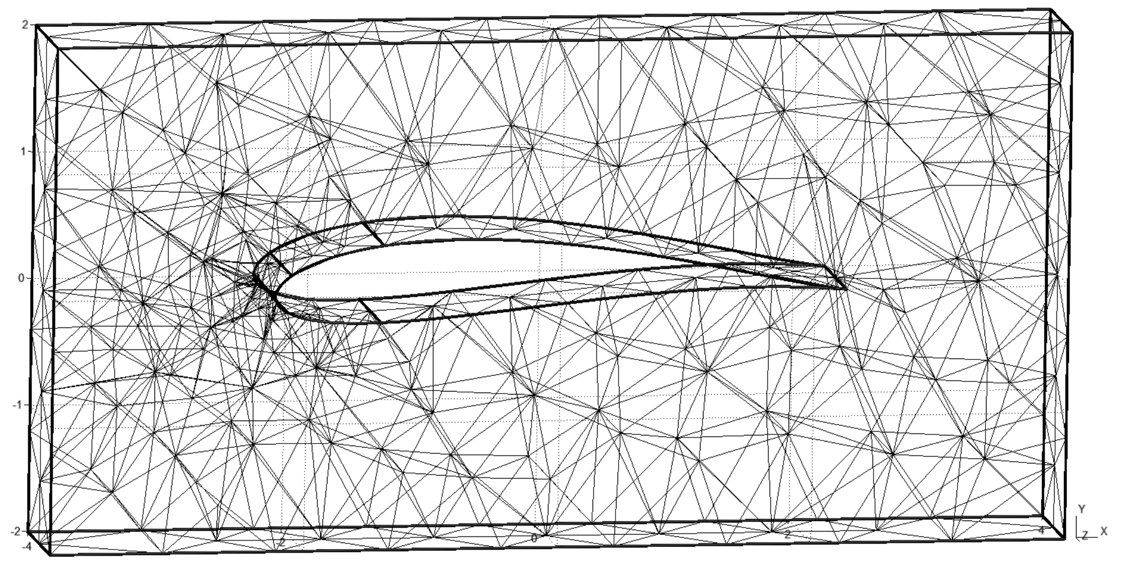
A potential flow is constructed by adding simple elementary flows together. For example, the flow around a rotating cylinder is the superposition of a uniform flow, a doublet and a vortex. For a detailed description of the mathematical background of potential flows and how they can be combined to form flows around arbitrary bodies in two dimensions, the book ”Theoretical aerodynamics” by Louis M. Milne-Thomson is recommended [47]. For the present thesis, the focus should lie on the Joukowsky airfoil example.

The flow velocity:

$$\tilde{W} = \tilde{u}_x - i\tilde{u}_y, \tag{4.6}$$



(a) Initial triangle mesh for airfield around the Joukowsky airfoil in two dimensions.



(b) Initial tetrahedra mesh for airfield around the Joukowsky airfoil in three dimensions.

Figure 4.5: Figure **(a)** shows the triangle mesh with mesh size controlled by curvature and the two-dimensional `Delaunay` meshing algorithm. Figure **(b)** shows the tetrahedral mesh created with the three-dimensional `Delaunay` meshing algorithm and without further mesh size control.

around the Joukowsky circle in the ζ-plane is complex conjugate. As mentioned above, it consists of a uniform flow, doublet and vortex, which are added respectively subtracted from each other to form the flow around the circle in two dimensions.

The complex conjugate velocity is calculated by the function:

$$\tilde{W} = V_\infty e^{-i\alpha} + \frac{i\Gamma}{2\pi(\zeta - \mu)} - \frac{V_\infty R^2 e^{i\alpha}}{(\zeta - \mu)^2}. \tag{4.7}$$

The first summand in equation (4.7) describes the uniform flow, the second summand describes the vortex and the third summand, which is subtracted in this case, represents the doublet. V_∞ is the velocity of the fluid, far away from the circle, without any influence and is known as the freestream velocity. The angle of attack α prescribes at which angle the flow hits the circle. After the Kutta condition, bodies with sharp trailing edges moving through a fluid will create a circulation about themself [48].

That circulation is represented by Γ and is calculated by the function:

$$\Gamma = 4\pi V_\infty R sin(\alpha + arcsin(\frac{\mu_y}{R})). \tag{4.8}$$

In equation (4.7) ζ is a point inside the domain, for which the velocity $\tilde{W}$ should be calculated and μ is the centre coordinate of the circle. Both points ζ and μ are complex. The last variable is the radius R of the circle. The reverse transformation of ζ to z with:

$$z_{1,2}(\zeta) = \frac{\zeta}{2} \pm \sqrt{\frac{\zeta^2}{4} - R^2}, \tag{4.9}$$

is unambiguous for all points, except of $\zeta = \pm R$ respectively $z = \pm 2R$. At these points, representing the leading and the trailing edge, equation (4.9) produces singularities of the flow field, where the velocity gets infinity large. These singularities disappear outside the flow domain at the leading and trailing edge because of the slightly larger airfoil geometry (see Subsection 4.2.1). Hence, the singularities of the flow field lay inside the airfoil boundary and, therefore, outside of the flow domain.

Equation (4.7) provides the velocity and direction of any given point ζ inside the domain of the ζ-plane. The Joukowsky transformation has to be applied to $\tilde{W}$ to get the flow around the Joukowsky airfoil, with the complex velocity W on the z-plane. This can be achieved by dividing $\tilde{W}$ by the derivative of z with respect to ζ as follows:

$$W = \frac{\tilde{W}}{\frac{d_z}{d_\zeta}} = \frac{\tilde{W}}{1 - \frac{1}{\zeta^2}}. \tag{4.10}$$

The complex velocity W is described as:

$$W = u_x - iu_y, \tag{4.11}$$

with u_x and u_y being the velocity components in the x and y direction.

It would be possible to calculate the lift per unit and coefficient of pressure for a given Joukowsky airfoil with the application of the Kutta-Joukowsky theorem [49], but this is optional for the evaluations in this thesis.

With equations (4.10) and (4.11), it is now possible to get the velocity and direction of each point inside the domains presented in the previous subsection. The advection solver (see Section 2.3) calculates the flow field with the centroids of each element in the domain. The flow field gets refined accordingly with the higher refinement around the advection domain.

The flow field used for the simulations in the following Section 4.3 is generated with the parameters shown in Table 4.1.

Table 4.1: The table contains the parameters used to construct the flow field around the circle in the ζ-plane and around the airfoil in the z-plane, respectively.

Flow field parameters		
freestream velocity	V_∞	1
angle of attack	α	0
centre of the circle on the ζ-plane	μ	(-0.12, 0.08)
radius of the circle on the ζ-plane	R	$R = \sqrt{(1-\mu_x)^2 + \mu_y^2}$

The same flow field will be applied for the triangle and the tetrahedra mesh (see Figure 4.5). The flow will be zero in the tetrahedral mesh's z-direction. Figure 4.6 shows the resulting flow field over the triangular mesh with higher resolution. The plotted lines, indicating the direction of the flow, show that the flow splits up at the leading edge and rejoins at the trailing edge of the airfoil. The flow is parallel to the airfoil surface near the airfoil, while it is almost constant in the horizontal direction further away. The colouring of the domain shows the magnitude of the flow velocity.

As expected, the flow has increased velocity on the top side and decreased on the bottom side. The flow magnitude is clipped to a range between two and four for better visualisation. The lowest velocity at the leading edge is actually 0.19, and the highest at the trailing edge is 5.2. The difference in pressure resulting from that difference in velocity on top and under the airfoil enables the aircraft to generate lift. Note that the magnitude shown in the figure is calculated for

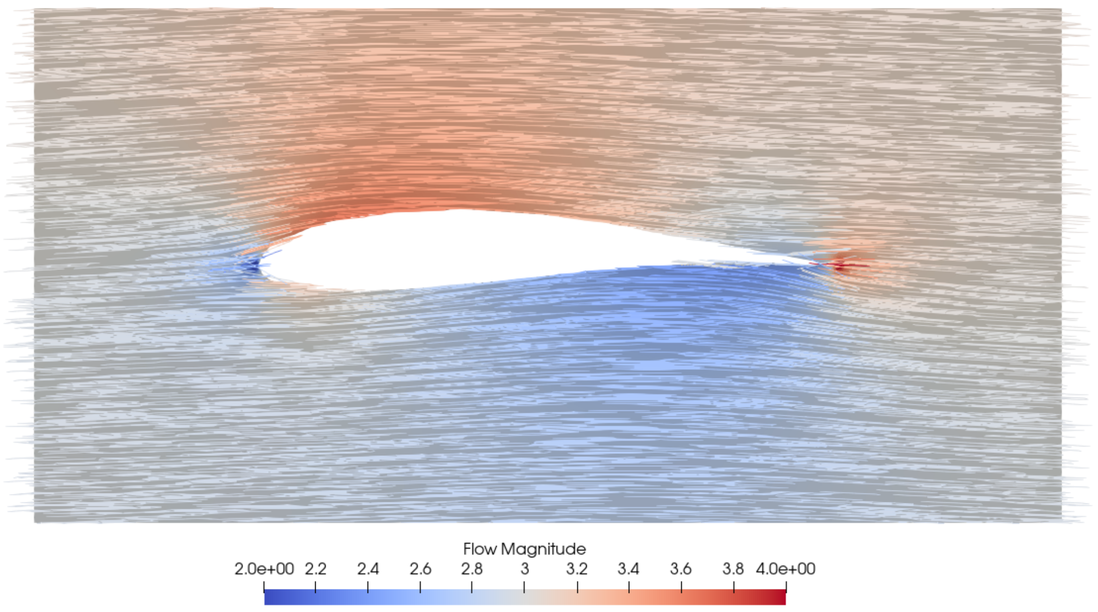


Figure 4.6: The flow field around the triangle mesh of the airfoil domain. The colouring indicates the velocity of the flow. The white lines show the direction of the flow throughout the domain.

the centroid of every triangle element in the mesh. Hence, increasing or decreasing the number of elements in the mesh will affect the flow magnitude.

Even though the singularities of the flow field at $z \pm 2R$ are inside the airfoil geometry and, therefore, outside of the flow domain, their effect is still visible at the leading and trailing edge. Nevertheless, the results are still realistic. Decreased flow velocity at the leading edge, which is a focal point of the freestream flow and increased velocity at the trailing edge, where the flow stalls, are noticeable on physical airfoils too.

4.3 Simulation

The advection simulations for triangle and tetrahedral meshes around a Joukowsky airfoil profile will be conducted in this section. The setup for both simulations can be found in Section 4.2. The simulations should deliver information to help answer the research question, stated in Section 1.4. Hence, comparing an advection through a linear and a curved mesh is inevitable.

The main criteria for the comparison are the volume loss, the amount of elements in the coarse mesh and during the simulation and the run time of the simulation. The volume loss for linear meshes is assumed to be decreased with a higher geometry resolution by the `Mesh.MeshSizeFromCurvature` option. Therefore, the coarsest mesh, presented in Subsection

4.2.1, will be re-meshed with higher resolution for the triangle and the tetrahedra mesh, to gain four different linear meshes for each.

Furthermore, it is assumed that runs with the curved option have less volume loss than the linear runs because the refinement happens concerning the curvature of the airfoil geometry. Less flux over the boundary of the flow domain should, therefore, occur. The volume loss should also be constant for the curved runs, even with a better geometry representation of the initial mesh. Hence, two different curved runs will be conducted, one with low and one with high initial geometry representation.

Summarising, four linear runs with increasing geometry representation and two curved runs with different geometry representations will be simulated for both the triangular and the tetrahedral mesh.

The advection solver of `t8code` takes several input parameters, which will be described in the following. The first option is the flow field, which should be calculated for the present mesh and is set via the `-u` argument. The user can choose between different predefined options. For all runs in this thesis, the flow around the Joukowsky airfoil is used and set with `-u8`. The following options control the refinement of the simulation. Option `-l` sets the initial refinement level, which will be set to `-l0` for all runs to get the coarsest initial mesh possible.

The `-r` option steers the refinement level on the boundary between the advection and the flow volume, set by the level-set-function (see Section 2.3). The argument `-r5` is passed for all runs with the triangle mesh. All runs with the tetrahedral mesh will be run with `-r3` because of the increased refinement ratio from two to three dimensions. Option `-b` controls how thick the refinement band around the advection volume border is. The larger the number, the smoother the transition between different refinement levels. All simulations will be run with `-b4`.

The mesh and geometry information is given via the `-f` option. The name and path of the *msh* file without the ending are provided with the argument. In the case of a curved run, there should also be a *brep* file with the same name in the same directory. Without the geometry information in the *brep* file, `t8code` cannot curve elements to the geometry. The dimension of the mesh is set with the `-d` option and is `-d2` for the runs with triangle meshes and `-d3` for the runs with tetrahedral meshes. The CFL number, explained in Section 2.3, is controlled by the `-C` option. The triangular runs converge with a CFL number of `-C0.5`, and for the tetrahedral runs, a CFL number of `-C0.05` is chosen, which grants conversion for all simulations.

The positioning of the advection volume happens with the options `-X`, `-Y` and `-Z`. For all runs, the advection volume starts in front of the airfoil at `-X-2.5` and `-Y0.5`. That will result in the volume being advected over the top side of the airfoil geometry. Because the triangle mesh has no third dimension, option `-Z0` is used for these runs. The tetrahedra mesh runs will be run with `-Z0.5`. After the centre point of the advection volume is set, the radius has to be set as well.

The radius will be 0.4 for all runs and is set with `-R0.4`. Option `-o` disables *vtk* output. That accelerates the run time. All run times in the Tables 4.2 and 4.4 are conducted with the `-o` option. The simulation time is set with the `-T` option and will be `-T1` for all simulations. Lastly, curved runs will be run with `-O` to tell the advection solver that the mesh should be curved.

The complete commands for the simulation are given in the following to enable the reader to retrace the simulations and the results.

The full command for all runs with triangular meshes is:

```
./t8_advection -u8 -l0 -r5 -b3 -f airfoil_windtunnel_triangles \
-d2 -C0.5 -X-2.5 -Y0.5 -Z0 -R0.5 -T1 (-o) (-O)
```

The full command for all runs with tetrahedral meshes is:

```
./t8_advection -u8 -l0 -r4 -b3 -f airfoil_windtunnel_tetrahedra \
-d3 -C0.1 -X-2.5 -Y0.5 -Z0.5 -R0.5 -T1 (-o) (-O)
```

All simulations will be run on an Intel Core i7-1185G7 processor with four cores and 32 GB of RAM. The run time of each run is affected by the processor's capacity. Hence, this has to be considered for the potential reproduction of the results presented in the upcoming chapter.

4.4 Results

The results of all simulations will be presented and discussed in this section. The assumptions made in the previous section will be checked to determine if curved elements provide advantages over linear elements when simulating curved geometries. All results presented in this section were run with the options and arguments given in Section 4.3.

Evaluation of triangle elements

The results for all simulations with the triangular mesh are summarised in Table 4.2. The information about the order of the mesh is given in the first column. There are four linear and two curved runs, as mentioned above. The initial resolution for all runs is represented by the `Mesh.MeshSizeFromCurvature` option, abbreviated as *MSFC*, of *Gmsh*. The number of elements in the coarse initial mesh called *cmesh* in `t8code`, is also provided. The next column contains the average amount of elements per time step. The advection solver outputs the number of elements in steps of ten percent during the simulation. The average of these ten values is shown in the column. The last two columns contain the percentage volume loss and the run time in seconds. As mentioned above, the run times are conducted without *vtk* output.

Table 4.2: The table shows the results of all simulation runs conducted with the triangular meshes. The exact simulation setup is described in detail in Section 4.3.

Evaluation of triangle elements					
mesh	initial resolution [MSFC]	cmesh elements	average elements	volume loss [%]	run time [s]
linear	3	182	6261	23.9622	41.4565
linear	5	202	7209	24.466	59.0176
linear	10	208	6869	24.7052	67.82
linear	15	232	7728	25.0507	86.4127
curved	3	182	6439	24.2485	42.289
curved	15	232	7699	25.269	81.902

The coarsest initial mesh is constructed with a mesh size from curvature (MSFC) value of three. With that value, no invalid elements or elements with sharp angles arise. That is especially important for the simulations with curved elements because they might intersect themselves after curving to the geometry. The first refinement step of the geometry happens with a MSFC value of five and is increased by five for the next two refinement steps. The two runs with the curved elements are conducted with the lowest and the highest initial geometry resolution. The exact number of elements for each coarse mesh can be found in the third column of Table 4.2. The number of elements in the coarse mesh and the average number of elements during the simulation does not rise linearly with the MSFC value.

That is, since the `Delaunay` algorithm positions the elements differently each time the geometry gets refined around the border. It is noticeable that the average number of elements during the simulation differs between the linear and curved runs, even though the number of cells in the initial mesh is equal. For both runs with poor initial geometry representation, the simulation with curved elements has a more significant average number of elements per time step. That changes for the runs with high initial geometry representation. In that case, the simulation with linear elements has more average elements per time step. The centroid of an element gets shifted when the element is curved. Some elements may be inside or outside the level-set function of the advection solver for simulations with curved elements, while they are not for simulations with linear elements.

The volume loss for the linear and curved simulations behaves against the assumption that it decreases with higher initial geometry resolution. In fact, the volume loss increases almost linearly with the MSFC value. The effect should be much smaller for the curved runs because the geometry resolution should not vary much between runs with different initial geometry resolutions. The effect is also noticeable for the curved runs but is smaller compared to the linear runs. The rise of volume loss is 4.54% for the simulations with linear elements and 4.21% for the runs with curved triangles. Because the volume gets advected over the top side of the airfoil, the effect of the MSFC value is slightly decreased. Therefore, it would be interesting to conduct a simulation with a smaller mesh size overall to see if the volume loss still increases or decreases in that case.

The run time rises with the amount of elements in each simulation. The most significant number of elements are present in the early time steps because the advection volume is complete and the boundary is refined. The number of elements shrinks with the advection over the airfoil because less boundary has to be refined. Analogue to the volume loss, the run time for curved elements is slightly higher for the runs with poor initial geometry resolution. It ends up being smaller for runs with higher initial geometry resolution. Curving triangles to the geometry only happens while reading the *msh* and *brep* files. The run time resulting from that is less and less influential the longer the run times get. The run times should be assessed with a constant time step for the linear and curved runs, to see if the same effect is visible.

Two additional simulations are conducted for the triangle mesh to examine the volume loss and run time for meshes with smaller elements. The initial mesh will have a MSFC value of three and a maximum mesh size of 0.25 overall. The advection through the new mesh will be simulated, once for linear and once for curved elements.

Table 4.3: The table shows the results of the two simulation runs conducted with a globally refined triangular mesh. The maximum mesh size is set to 0.25.

Evaluation of triangle elements (globally refined)					
mesh	initial resolution [MSFC]	cmesh elements	average elements	volume loss [%]	run time [s]
linear	3	1507	13604	19.0951	143.009
curved	3	1507	13591	19.1311	127.538

Table 4.3 shows the results of the simulation runs with a globally refined mesh. The simulations confirm the assumptions made above. Both runs have less volume loss with the linear and curved mesh than those shown in Table 4.2. Again, the curved run's volume loss is slightly larger than the linear run. The difference, however, is smaller than before. The runs with smaller global

element size confirm that curved elements seem to decrease the run time. The simulations with curved triangles, shown in Table 4.3, are faster by 12.13% compared to the simulation with linear elements.

A shorter run time for simulations with curved elements seems to be illogical, because the geometry has to be evaluated every time an element laying on the geometry is refined. For curved elements that means, that the implemented algorithm has to be run additionally, which should result in a longer run time. It was recognised after some time, that the partitioning of the initial mesh over multiple processes is executed for linear meshes but not for curved meshes, because that functionality is not yet implemented. The partitioning algorithm has a effect on the run time, even though all runs are conducted on a single process.

After disabling the partitioning of the initial mesh for linear coarse meshes, the runs, presented in Table 4.3, are run again. Each run is conducted five times and an average of the run times is calculated to prevent any effects from warming up the central processing unit (CPU) or similar.

The results show that the linear runs have an average run time of $115.703s$ and the curved runs take $126.8304s$ on average. All runs have a number of 1795 time steps. The linear runs have a constant volume loss of 19.0951% and the curved runs of 19.1311%.

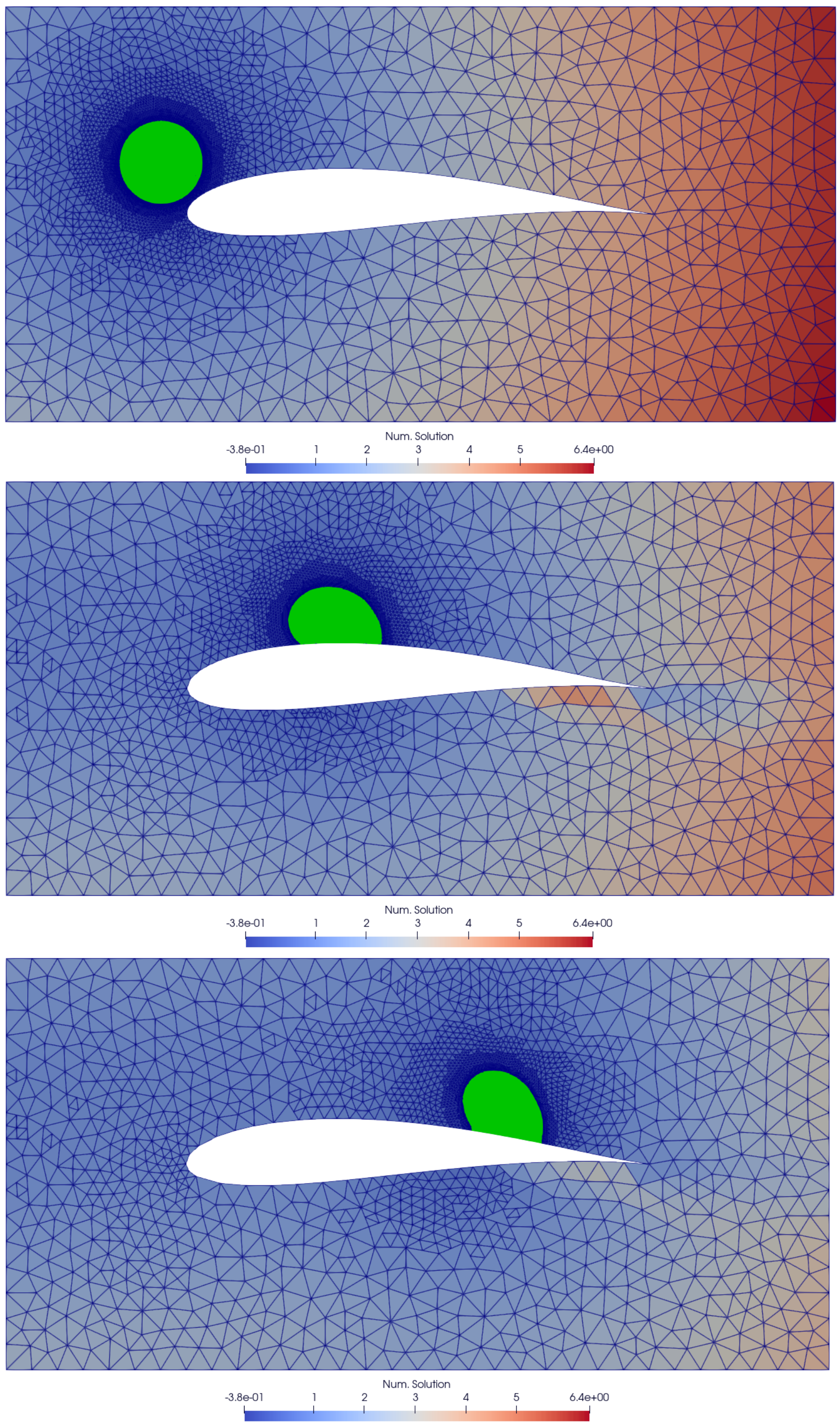


Figure 4.7: The figure shows three frames of the advection through the triangle mesh. The simulation chosen for this visualisation is the globally refined mesh with curved elements (see Table 4.3).

After evaluating the simulation results, Figure 4.7 shows an example of how the advection through the triangle mesh looks like. The simulation shown in the figure is the run with global refinement and curved elements, whose results are presented in Table 4.3. The green-coloured elements represent the advection volume. These elements can be filtered by the $\phi_{E,\,t}$ values during each time step. As described in Section 2.3, the values inside the advection volume are negative, and those outside the advection volume are positive.

The figure shows how the flow on the top side of the airfoil profile deforms the advection volume. As it hits the airfoil, it is squeezed, slightly widening the horizontal radius of the volume. That is visible in the middle frame of the figure. Towards the trailing edge, the advection volume slowly detaches from the airfoil surface, stretching the radius vertically. That is visible in the last frame of the figure. In the middle and last frame, some numerical anomalies occur on the bottom side of the airfoil and at the trailing edge.

The trailing edge is placed manually in the *geo* file, which constructs the geometry and mesh to function as a corner point, ensuring valid elements. By manually setting the trailing edge, a slight deviation between the geometry and the geometry used for the flow field is present. That results in numerical anomalies, which do not affect the advection results on the top side of the airfoil.

Evaluation of tetrahedra elements

After the results of the triangle evaluation are reviewed in detail, the tetrahedra elements have to be evaluated as well. The setup of the simulations is similar to the setup for the triangle simulations, with some minor adjustments. The concrete setup for the tetrahedral simulations can be found in Section 4.3.

Again, four runs with linear elements and rising geometrical accuracy controlled by the MSFC option are conducted. Additionally, two curved runs, one with the lowest and one with the highest geometrical accuracy, are simulated to see if the assumptions above prove true or false.

Table 4.4 presents the results of all six runs with the tetrahedral mesh shown in Subsection 4.2.1. Compared to the triangle runs, the much larger number of elements in the coarse mesh and the more extended run times are immediately noticeable. First, the simulations with linear elements will be compared before looking into the runs with curved tetrahedra.

The four linear runs are distinguished by the MSFC value of the initial mesh. That is reflected in the number of coarse mesh elements, which increases from 748 to 1083 with rising geometrical resolution. When looking at the average number of elements for each linear run, the same rise is noticeable. Only the simulation with MSFC value 7 falls out of the trend. The average number of elements in that run is lower than in the run with MSFC value 5, although the number of elements in the coarse mesh is higher. That could be due to the re-structuring of the mesh while re-meshing.

Table 4.4: The table shows the results of all simulation runs conducted with the tetrahedral meshes. The exact simulation setup is described in detail in Section 4.3.

Evaluation of tetrahedral elements					
mesh	initial resolution [MSFC]	cmesh elements	average elements	volume loss [%]	run time [s]
linear	3	748	87588	74.5041	1374.83
linear	5	892	100738	68.5397	2171.05
linear	7	1014	94668	75.6385	2620.02
linear	9	1083	100019	71.4485	2860.48
curved	3	748	89436	74.495	2220.29
curved	9	1083	108614	71.5912	4451.79

Finer elements could have appeared in a mesh region, where the advection volume does not pass through. Refinement of these finer elements would be lower. Therefore, fewer elements occur on average per time step.

The different behaviour of the third linear simulation carries over to the volume loss. There is a downward trend in volume loss with rising initial geometry resolution. It starts at 74.5041% and ends at 71.4485%. The run with MSFC $= 7$ falls out of that trend with a volume loss of 75.6385%. Again, that might be due to the re-structuring of the initial mesh, resulting in a lower resolution of the geometry on the top side of the airfoil. The downward trend of volume loss with rising geometrical resolution verifies the assumption. While the effect was not noticeable for the simulations with linear triangle elements, the tetrahedra elements immediately confirmed the hypothesis.

The geometry boundary in a two-dimensional triangle mesh is a curve. In a three-dimensional tetrahedral mesh, the boundary is a surface, meaning more elements must be used to discretise it. Steering of geometrical resolution with the MSFC value of the initial mesh is, therefore, more powerful for the three-dimensional mesh than the two-dimensional mesh.

The run times of the simulations with linear tetrahedra elements rise with the MSFC value and the number of elements in the initial mesh. Even though the number of coarse mesh cells rise by 30.93% from 748 to 1083, the run time rises by 51.94% form $1374.83s$ to $2860.48s$. That shows that a high-quality initial mesh, with high resolution only in the critical regions, can save much run time.

Now, the simulations with curved tetrahedra will be analysed in detail. The average number of elements per time step is higher for the curved runs than their linear pendants. The same effect was discovered for triangle elements. The volume loss is higher for the curved runs. Again, the same effect was noticeable for triangle elements and will be examined by the runs with global mesh refinement.

Still, a decreasing volume loss is noticeable with rising initial geometry resolution for the curved simulations. The run times are significantly longer compared to the linear runs. The run time of the curved run with the lowest initial geometry resolution is longer by a factor of 1.615 compared to the linear run with the same MSFC value. Between both curved runs, the rise of run time is with 50.13%, almost similar to the linear runs, where the run time increased by 51.94%.

There is no need to run simulations with globally refined mesh size for the tetrahedral mesh because the above assumptions were confirmed by the simulations presented in Table 4.4.

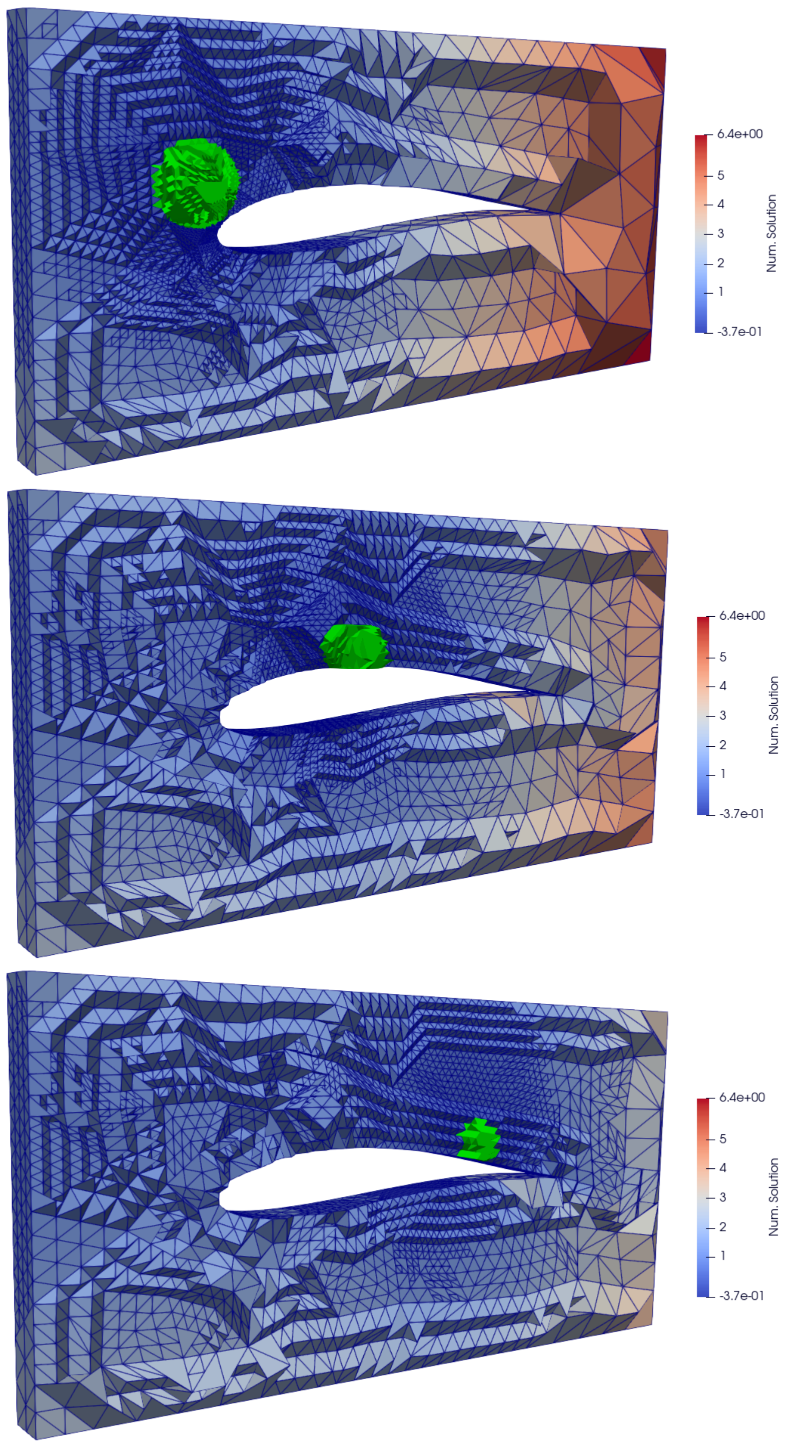


Figure 4.8: The figure shows three frames of the advection through the tetrahedra mesh. The simulation chosen for this visualisation is the run with MSFC = 3 and curved elements (see Table 4.4).

Figure 4.8 visualises the advection through the tetrahedra mesh with MSFC $= 3$ and curved elements. The same three frames are displayed, as in Figure 4.7 with the triangle mesh. The mesh of the flow domain in Figure 4.8 is clipped at $Z = 0.5$ to show the advection volume at the maximum radius. While the volume loss is barely visible for the triangle mesh in Figure 4.7, the volume loss of 71.5912% for the tetrahedra simulation is noticeable.

The resolution at the border of the advection volume is with a refinement level of $r = 3$ minor, compared to $r = 5$ for the triangle simulations. That is due to the exponential increase in run time to refine three-dimensional elements. While two-dimensional elements are refined by $1 : 4$ per level, three-dimensional elements are refined by $1 : 8$. That means that a triangle element with $r = 5$ refines to 1024 sub-elements, and a tetrahedron element with the same r value refines to 32768 sub-elements. Hence, a lower refinement level is chosen for the tetrahedra simulations.

However, a refinement level of $r = 3$ produces 512 sub-elements, which is less compared to the triangle refinement. That is why the volume losses for the tetrahedral simulations are much higher, with 72.7% averaged over all six runs, compared to only 24.62% over the triangle runs, excluding the runs with global mesh size refinement. The lower resolution is noticeable comparing Figure 4.7 and 4.8.

Besides the lower resolution, both simulations look similar, and the behaviour of the advection volume is the same. The numerical anomaly mentioned above is again present for the tetrahedral simulation but does not affect the airfoil geometry's advection volume on the top side.

5 Conclusion

The answer to the research question:

How can the implementation of curved triangular elements in tree-based AMR be realised, and does it prove advantageous over a linear approach?

set at the beginning of this thesis, has to be split into an answer regarding the implementation of curved triangle and tetrahedral elements and an answer for their performance compared to linear elements.

First, the implementation of curved elements will be concluded. The curving of elements to arbitrary geometries happens right after `t8code` has read a *msh* file. Together with the geometrical parametrisation information provided by a *brep* file, the implemented algorithms for triangle and tetrahedra elements shift specific reference points inside the element area (2D) or volume (3D) according to the geometry curvature at the element. The accurate scaling of the displacement at edges E or faces K of the element through the area (2D) or volume (3D) is critical to maintaining the edge or face connection to adjacent elements.

Furthermore, it is crucial to cover all possible cases of linkage between an element and a geometry. For triangles, those are the linkage of a face K against an OCCT surface κ and the linkage of an edge E to an OCCT surface κ or OCCT curve ε. The linkage of a triangle element's edge E to an OCCT surface is unlikely to occur in most cases.

Possible edge cases have to be considered as well. For example, an element face K could be linked to an OCCT surface κ and an edge E of that face is linked against an OCCT edge ε. In this case, the reference points used to evaluate the surface displacement must be shifted by the scaled edge displacement before the surface displacement can be assessed.

The implemented algorithm for curving tetrahedral elements parallels the algorithm for curving triangles. The displacement scaling for triangle elements (see Figure 3.5) gets expanded to scale displacement through the tetrahedron volume (see Figure 3.7). The scaling on the triangle faces of the tetrahedron is adapted slightly (see Figure 3.6).

The before-mentioned linkage of tetrahedron edges E to OCCT surfaces κ occurs regularly. In all cases where an edge E is linked against any OCCT geometry, the displacement has to be scaled over both neighbouring faces K of the edge E. Again, the algorithm must consider any edge cases of linkage between a tetrahedron and an OCCT geometry.

The detailed implementation of the curving algorithms for triangle and tetrahedra elements can be found in Chapter 3. The first part of the research question, concerning implementing such algorithms into tree-based AMR software like `t8code`, is thus answered.

The second part of the research question, concerning the performance of curved elements compared to linear elements, has to be answered.

Before conducting any simulations with curved elements and the advection solver of `t8code` (see Section 2.3) it was assumed that curved elements generate less volume loss on a curved geometry, compared to linear elements, but have increased simulation run times. The simulations conducted in Chapter 4 show that the volume loss for the chosen geometry example of a Joukowsky airfoil is not lower for curved elements compared to linear elements. That is true for triangles as well as tetrahedra elements.

Because all simulations ran the same scenario, the assumption that the volume loss is lower for curved geometries simulated with curved elements, compared to linear elements, is not wrong per se. Different scenarios, for which a coarser initial mesh can be constructed, could provide different results. The simulation of different scenarios with curved geometries could give more information. Previous work shows that curved elements provide more accurate results in specific scenarios [15].

The second assumption regarding the run time of simulations with curved elements is valid for tetrahedral elements. The run times of these simulations with curved tetrahedra are about 1.6 times longer than those with linear elements. However, the run times with curved triangle elements are faster than linear runs. After disabling the partitioning for linear and curved runs, the assumption was met with longer run times for curved meshes.

The implementations in this thesis can be understood as a proof of concept. In further development, the focus should lie on the execution and run time acceleration of the code.

Even though, curved elements diffidently prove helpful in some cases, all simulations show that the curving of elements to gain a better geometrical representation of the simulation mesh does not correct poor initial meshes. Hence, curving elements should be seen as a way to achieve even better geometrical representation for already sufficiently discretised curved geometries.

Still, curved to a geometry, a coarse initial mesh cell can reduce the number of elements in the coarse mesh when the same curvature would otherwise be approximated with more than one linear element. That effect would be noticeable, especially for extensive simulations with a high amount of coarse mesh cells. Memory efficiency, one of the main strengths of tree-based AMR, can be improved.

In summary, the answer to whether curved elements are advantageous over linear elements is as differentiated as the results. Curved elements can potentially be used as an additional accuracy or memory improvement for simulations with curved geometries. Still, they also have some disadvantages regarding the run time. The conduction of further simulation scenarios is necessary to answer the question certainly. The results presented in Chapter 4 prove that curved elements are not advantageous in all cases. The results do not allow the deduction of specific guidelines for using curved elements.

6 Outlook

The ability to curve triangle and tetrahedra elements to arbitrary geometries is now implemented in `t8code` and enqueues behind curved quadrilaterals and hexahedra. The performance of the curved triangle and tetrahedra elements showed some promising results regarding memory efficiency.

Still, the expected increased numerical accuracy could not be determined. Hence, conducting simulations of different scenarios and geometries besides the advection around a Joukowsky airfoil would be interesting to determine if the curved elements provide higher accuracy in other cases.

Besides simulating different scenarios with the advection solver, it would be interesting to conduct simulations with other solvers, which can concern higher-order elements. The effect of curved elements is only noticeable by the correction of element corner points under the refinement of the mesh. The numerical calculation of the advection solver does not consider the actual curvature of an element. The solver calculates the elements as if they were linear.

Solvers, which can handle higher-order elements, would respect the curvature, leading to higher numerical accuracy of these simulations. Currently, there are no solvers available that can handle curved elements in `t8code`. However, multiple projects are running, which aim to either create a more powerful solver for `t8code` exclusively [50] or to adapt third-party solvers to be able to handle `t8code` managed meshes [51, 52, 53]. The two mentioned projects are *t8dg* and *Trixi*. Both solvers are based on the discontinuous Galerkin solving method. As soon as these solvers can simulate curved elements managed by `t8code`, further evaluation possibilities arise.

As mentioned before, the implementations of this thesis can be viewed as a proof of concept. Hence, the code has to be optimized by applying performance engineering. That will accelerate the code execution, and memory efficiency will also benefit.

The curving algorithms' stability heavily depends on the initial mesh quality. If coarse initial elements are placed at locations with high geometrical curvature, it is possible to create self-intersecting elements which are not valid. Also, elements with sharp corner angles could result. In the future, the research of mesh untangling methods should be promoted. Another possibility would be to distribute the geometrical curvature not only on one element but on multiple elements in a certain proximity to the geometry. That would relieve the element right on the geometry, and fewer invalid elements would arise.

The research and implementation of curving algorithms for triangle and tetrahedral elements are crucial in making `t8code` capable of curving all element types to arbitrary geometries. With the curving algorithms for quadrilaterals and hexahedra, it should be easy to implement equal algorithms for prism and pyramid elements.

Besides the ability of AMR, `t8code` can also be run in parallel, spreading the managed mesh over multiple processes to gain maximum performance. The feature was not used in this thesis because it cannot work perfectly with curved elements. While reading a *msh* file, `t8code` distributes the cells over a user-defined number of processes. That procedure is called partitioning. For the use of curved elements, a *brep* file containing the geometrical information must also be provided. Currently, `t8code` can partition the geometry data but cannot connect it with the right cells on different processes. That leads to the coarse mesh being saved on each process, which massively decreases the memory efficiency. Therefore, the accurate partitioning of coarse meshes with geometrical data would be necessary to implement in future work.

Closely connected to the partitioning is the load balancing of `t8code`. After the coarse mesh is partitioned on multiple processes, the number of elements managed per process is equal or ± 1 for an uneven number of elements. While running AMR, the number of elements per process is unbalanced. The load balancing algorithm re-distributes the elements over all processes so that the load is equal for all of them. With the implementation of curved elements, there are now two kinds of elements with different loads because the load introduced by curved elements is higher than linear ones. That is not yet taken into account by the load balancing algorithm. Even though two processes have the same amount of elements, one could have much more curved elements and, therefore, a higher load. Load balancing concerning different load levels should be promoted in future work.

Bibliography

[1] Rao V. Dukkipati. *Numerical Methods*. New Age International (P) Ltd., Publishers, 2010. ISBN: 978-81-224-2978-7.

[2] Carl Runge. “Über die numerische Auflösung von Differentialgleichungen”. In: *Mathematische Annalen* 46 (1895), pp. 167–178. DOI: `10.1007/BF01446807`.

[3] Wilhelm Kutta. *Beitrag zur näherungsweisen Integration totaler Differentialgleichungen*. Teubner, 1901.

[4] Michael Wagner. “Scalability Evaluation of the CFD Solver CODA on the AMD Naples Architecture”. In: (2023). Ed. by Michael M. Resch et al., pp. 95–106.

[5] Karl-Kien Cao et al. “A multi-perspective approach for exploring the scenario space of future power systems”. In: (2022). ISSN: 2197-9294.

[6] Timothy J. Baker. “Developments and trends in three-dimensional mesh generation”. In: *Applied Numerical Mathematics* 5.4 (1989), pp. 275–304. ISSN: 0168-9274. DOI: `https://doi.org/10.1016/0168-9274(89)90012-3`. URL: `https://www.sciencedirect.com/science/article/pii/0168927489900123`.

[7] Joachim Schöberl, H Gerstmayr, and R Gaisbauer. *NETGEN-automatic mesh generator*. 2012.

[8] Xiaojuan Luo et al. “p-Version Mesh Generation Issues.” In: *IMR*. 2002, pp. 343–354.

[9] Ivo Babuska and Barna Szabo. “Trends and new problems in finite element methods”. In: *MATHEMATICS OF FINITE ELEMENTS AND APPLICATIONS* 9 (1996), pp. 1–34.

[10] SH Lo. “Finite element mesh generation and adaptive meshing”. In: *Progress in Structural Engineering and Materials* 4.4 (2002), pp. 381–399.

[11] Srinivas Raghothama and Vadim Shapiro. “Boundary representation deformation in parametric solid modeling”. In: *ACM Transactions on Graphics (TOG)* 17.4 (1998), pp. 259–286.

[12] Johannes Holke et al. *t8code*. Version 1.4.1. July 2023. DOI: `10.5281/zenodo.7034838`. URL: `https://github.com/dlr-amr/t8code`.

[13] Johannes Holke. "Scalable algorithms for parallel tree-based adaptive mesh refinement with general element types". PhD thesis. Rheinische Friedrich-Wilhelms-Universität Bonn, 2018.

[14] Sandro Elsweijer. *Curved Domain Adaptive Mesh Refinement with Hexahedra*. Tech. rep. Hochschule Bonn-Rhein-Sieg, July 2021. URL: `https://elib.dlr.de/143537/` (visited on 03/14/2022).

[15] Sandro Elsweijer. *Evaluation and generic application scenarios for curved hexahedral adaptive mesh refinement*. Master thesis. 2022. DOI: `10.13140/RG.2.2.34714.11203`. URL: `https://elib.dlr.de/186561/`.

[16] Dietrich Braess. *Finite elements: Theory, fast solvers, and applications in solid mechanics*. Cambridge University Press, 2007.

[17] Randall J LeVeque. *Finite volume methods for hyperbolic problems*. Vol. 31. Cambridge university press, 2002.

[18] Bernardo Cockburn, George E Karniadakis, and Chi-Wang Shu. *Discontinuous Galerkin methods: theory, computation and applications*. Vol. 11. Springer Science & Business Media, 2012.

[19] Marsha J Berger and Joseph Oliger. "Adaptive mesh refinement for hyperbolic partial differential equations". In: *Journal of computational Physics* 53.3 (1984), pp. 484–512.

[20] Michael Griebel and Gerhard Zumbusch. "Parallel multigrid in an adaptive PDE solver based on hashing". In: *Advances in Parallel Computing*. Vol. 12. Elsevier, 1998, pp. 589–599.

[21] Vokan Akcelik et al. "High resolution forward and inverse earthquake modeling on terascale computers". In: *Proceedings of the 2003 ACM/IEEE Conference on Supercomputing*. 2003, p. 52.

[22] David A Kopriva, Stephen L Woodruff, and M Yousuff Hussaini. "Computation of electromagnetic scattering with a non-conforming discontinuous spectral element method". In: *International journal for numerical methods in engineering* 53.1 (2002), pp. 105–122.

[23] Michael Lahnert et al. "Towards lattice-Boltzmann on dynamically adaptive grids–minimally-invasive grid exchange in ESPResSo". In: *Proceedings of the ECCOMAS Congress*. 2016, pp. 1–25.

[24] Michel Rasquin et al. "Scalable implicit flow solver for realistic wing simulations with flow control". In: *Computing in Science & Engineering* 16.6 (2014), pp. 13–21.

[25] Umit V Catalyurek et al. “Hypergraph-based dynamic load balancing for adaptive scientific computations”. In: *2007 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2007, pp. 1–11.

[26] Øyvind Hjelle and Morten Dæhlen. *Triangulations and applications*. Springer Science & Business Media, 2006.

[27] NA Golias and RW Dutton. “Delaunay triangulation and 3D adaptive mesh generation”. In: *Finite elements in analysis and design* 25.3-4 (1997), pp. 331–341.

[28] Benjamin S Kirk et al. “libMesh: a C++ library for parallel adaptive mesh refinement/coarsening simulations”. In: *Engineering with Computers* 22 (2006), pp. 237–254.

[29] Carsten Burstedde, Lucas C Wilcox, and Omar Ghattas. “p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees”. In: *SIAM Journal on Scientific Computing* 33.3 (2011), pp. 1103–1133.

[30] Ian Stroud. *Boundary representation modelling techniques*. Springer Science & Business Media, 2006.

[31] *Open CASCADE Technology*. Accessed September 20, 2023. 2022. URL: `https://dev.opencascade.org/doc/overview/html/`.

[32] Richard Courant, Kurt Friedrichs, and Hans Lewy. “Über die partiellen Differenzengleichungen der mathematischen Physik”. In: *Mathematische annalen* 100.1 (1928), pp. 32–74.

[33] *Gmsh - msh file format*. Accessed September 21, 2023. URL: `https://gmsh.info/doc/texinfo/gmsh.html#MSH-file-format`.

[34] Zhijian J Wang et al. “High-order CFD methods: current status and perspective”. In: *International Journal for Numerical Methods in Fluids* 72.8 (2013), pp. 811–845.

[35] Jens M Melenk and Ivo Babuška. “The partition of unity finite element method: basic theory and applications”. In: *Computer methods in applied mechanics and engineering* 139.1-4 (1996), pp. 289–314.

[36] Nicolas Moës, John Dolbow, and Ted Belytschko. “A finite element method for crack growth without remeshing”. In: *International journal for numerical methods in engineering* 46.1 (1999), pp. 131–150.

[37] David J Benson et al. “A generalized finite element formulation for arbitrary basis functions: from isogeometric analysis to XFEM”. In: *International Journal for Numerical Methods in Engineering* 83.6 (2010), pp. 765–785.

[38] Charbel Farhat, Isaac Harari, and Leopoldo P Franca. “The discontinuous enrichment method”. In: *Computer methods in applied mechanics and engineering* 190.48 (2001), pp. 6455–6479.

[39] Devina P Sanjaya and Krzysztof J Fidkowski. “Improving high-order finite element approximation through geometrical warping”. In: *AIAA Journal* 54.12 (2016), pp. 3994–4010.

[40] Johannes et al. Holke. *t8code: Curved Geometry Fork*. Nov. 2023. URL: `https://github.com/jfussbro/t8code/tree/master-thesis-fussbroich`.

[41] Nikolai Joukowsky. “Über die konturen der Tragflächen der Drachenflieger”. In: *Zeitschrift für Flugtechnik und Motorluftschiffahrt* 1.22 (1910), pp. 281–285.

[42] Robert Thomas Jones. *Wing Theory*. Princeton: Princeton University Press, 1990. ISBN: 9781400860777. DOI: `doi:10.1515/9781400860777`. URL: `https://doi.org/10.1515/9781400860777`.

[43] Der-Tsai Lee and Bruce J Schachter. “Two algorithms for constructing a Delaunay triangulation”. In: *International Journal of Computer & Information Sciences* 9.3 (1980), pp. 219–242.

[44] Lord Rayleigh. “I. On the flow of compressible fluid past an obstacle”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 32.187 (1916), pp. 1–6.

[45] John L Hess and A_M O Smith. “Calculation of potential flow about arbitrary bodies”. In: *Progress in Aerospace Sciences* 8 (1967), pp. 1–138.

[46] Sangmo Kang, Haecheon Choi, and Sangsan Lee. “Laminar flow past a rotating circular cylinder”. In: *Physics of Fluids* 11.11 (1999), pp. 3312–3321.

[47] Louis Melville Milne-Thomson. *Theoretical aerodynamics*. Courier Corporation, 1973.

[48] Laurence Joseph Clancy. “Aerodynamics”. In: *Journal of Fluid Mechanics* 77.3 (1976), pp. 623–624.

[49] LQ Liu, JY Zhu, and JZ Wu. “Lift and drag in two-dimensional steady viscous and compressible flow”. In: *Journal of Fluid Mechanics* 784 (2015), pp. 304–341.

[50] Lukas Dreyer. “The local discontinuous galerkin method for the advection-diffusion equation on adaptive meshes”. Principal reviewer: Prof. Dr. Carsten Burstedde, Second reviewer: Dr. Johannes Holke. MA thesis. Rheinische Friedrich-Wilhems-Universität Bonn, Feb. 2021. URL: `https://elib.dlr.de/143969/`.

[51] Hendrik Ranocha et al. “Adaptive numerical simulations with Trixi.jl: A case study of Julia for scientific computing”. In: *Proceedings of the JuliaCon Conferences* 1.1 (2022), p. 77. DOI: 10.21105/jcon.00077. arXiv: 2108.06476 [cs.MS].

[52] Michael Schlottke-Lakemper et al. *Trixi.jl: Adaptive high-order numerical simulations of hyperbolic PDEs in Julia*. https://github.com/trixi-framework/Trixi.jl. Aug. 2020. DOI: 10.5281/zenodo.3996439.

[53] Michael Schlottke-Lakemper et al. “A purely hyperbolic discontinuous Galerkin approach for self-gravitating gas dynamics”. In: *Journal of Computational Physics* (June 2021), p. 110467. DOI: 10.1016/j.jcp.2021.110467. arXiv: 2008.10593 [math.NA].