Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

DLR Deutsches Zentrum
für Luft- und Raumfahrt
German Aerospace Center

Institute for Software
Technology

Masterthesis

Maschinenbau (M.Eng.)

**Fachbereich Elektrotechnik,
Maschinenbau und Technikjournalismus**

Evaluation and generic application scenarios for curved hexahedral adaptive mesh refinement

Vorgelegt von: Sandro Elsweijer
Sandro.Elsweijer@gmail.com
Erstprüfer: Prof. Dr. Dirk Reith
Zweitprüfer: Dr. Jan Kleinert
Eingereicht am: 28.03.2022

Abstract

In (dynamic) adaptive mesh refinement (AMR) an input mesh is refined or coarsened to the need of the numerical application. This refinement happens with no respect to the originally meshed domain and is therefore limited to the geometrical accuracy of the original input mesh. We presented a novel approach to equip this input mesh with additional geometry information, to allow refinement and high-order cells based on the geometry of the original domain.

We already showed a limited implementation of this algorithm. Now we evaluate this prototype with a numerical application and we prove its influence on the accuracy of certain numerical results. To be as practical as possible, we implement the ability to import meshes generated by `Gmsh` and equip them with the needed geometry information. Furthermore, we improve the mapping algorithm, which maps the geometry information of the boundary of a cell into the cell's volume.

With these preliminary steps done, we use out new approach in a simulation of the advection of a concentration along the boundary of a sphere shell and past the boundary of a rotating cylinder. We evaluate the accuracy of our approach in comparison to the conventional refinement of cells to answer our research question: How does the performance and accuracy of the hexahedral curved domain AMR algorithm compare to linear AMR when solving the advection equation with the linear finite volume method?

To answer this question, we show the influence of curved AMR on our simulation results and see, that it is even able to outperform far finer linear meshes in terms of accuracy. We also see that the current implementation of this approach is too slow for practical usage. We can therefore prove the benefits of curved AMR in certain, geometry-related application scenarios and show possible improvements to make it more feasible and practical in the future.

Contents

List of Figures

List of Tables

Acronyms

AMR adaptive mesh refinement

API application programming interface

CAD computer aided design

CFD computational fluid dynamics

CFL Courant–Friedrichs–Lewy

DG discontinuous Galerkin

FE finite element

FV finite volume

MPI Message Parsing Interface

NURBS non-uniform rational B-spline

OCCT OpenCASCADE Technology

PDE partial differential equation

SFC space-filling curve

Symbols

Latin symbol	Description
A	Area
C	Cell of a coarse mesh
d	Dimension
E	Element in a cell or a cell itself, if the cell is not refined
$\mathscr{E}$	Edge of a cell
$\mathscr{E}_{\mathscr{F}}$	Edge of a face $\mathscr{F}$ of a cell
$\mathscr{F}$	Face of a cell
G	Improved volume map
g	Old volume map
L_1, L_2, L_3	Linear, bilinear or trilinear interpolation
$\mathscr{N}$	Node of a cell
$\vec{n}$	Normal vector
R	Radius of a cylinder
r	Refinement level of an element
t	Time
$\vec{u}$	Flow field
u, v	Parameters of a geometry
V	Volume

Greek symbol	Description
Γ	`OCCT` edge; the topological, bounded section of a curve γ
γ	`OCCT` curve which maps $[u_{\min}, u_{\max}] \to \mathbb{R}^3$
γ_ξ	Map which converts $[u_{\min,\gamma}, u_{\max,\gamma}] \to u_\xi \times v_\xi$
ΔC	Scaled version of Δc
Δc	Coordinate error correction induced by a geometry
ΔP	Scaled version of Δp

Greek symbol	Description
Δp	Parameter error correction induced by an OCCT curve on an OCCT surface
Δt	Time step
K	OCCT vertex; topology of κ
κ	OCCT point which lies in $\mathbb{R}^3$
Ξ	OCCT face; topological, bounded section of a surface ξ
ξ	OCCT surface which maps $[u_{\min}, u_{\max}] \times [v_{\min}, v_{\max}] \rightarrow \mathbb{R}^3$
ϕ	Concentration of an advected property
ϕ_{vp}	Velocity potential of a flow
ψ	Flux of an element into another element
ψ_{sf}	Stream function of a flow
Ω	Computational domain

1 Introduction

The accurate modeling of partial differential equations (PDEs) is an always relevant problem in the modern industry. An example is the calculation of stress in structural components or fluid forces on a vehicle. Nowadays, it is possible to model these problems with limited computing power on modern workstations, but there is an everlasting strive to get even more accurate results or to model even more complex problems on comparable hardware [1, 2].

This strive is not limited to the industry. Research fields like earth system modeling [3, 4] or direct and turbulent flow simulations [5, 6] simulate vast systems on thousands of computing cores while having computing time restrictions. Hence, it is useful to improve the models and make them more efficient.

One technique, which has led to more efficiency, is the usage of high-order meshes. High-order meshes increase the number of nodes on the cell boundary and inside the cell's volume. These can be used to generate polynomials, which describe the solution and geometry of the cell more accurately. Because of the more accurate cells, it is generally possible to use coarser meshes and therefore to save computational power or to have faster simulations [7].

The construction of high-order meshes from linear meshes is an ongoing topic in research because of the possible generation of self-intersecting cells. Generally, high order meshes are constructed by the generation of additional nodes on the cell's edges, faces, and in its volume. If the face or edge lies on a geometry, these nodes follow the curvature of the geometry (see Fig. 1.1). More on that in Sec. 3.3.

This approach approximates the geometry better than linear cells, but it is still an approximation, and information is lost. Furthermore, this approximation is static. If a high-order mesh is generated, the order of the mesh is not able to change during a simulation [1, 8].

In [9, 10], we proposed a novel approach which mitigates these problems. We defined a hexahedral volume map, which can link cell faces and edges to computer aided design (CAD) geometries like curves and surfaces and map the deformations generated by these geometries into the cell volume. A two-dimensional example of this is shown in Fig. 1.2. In this example, an edge of the cell is linked to a spline and the refinement of this cell happens with respect to that. The volume map even maps the deformation generated by the spline into the volume of that cell.

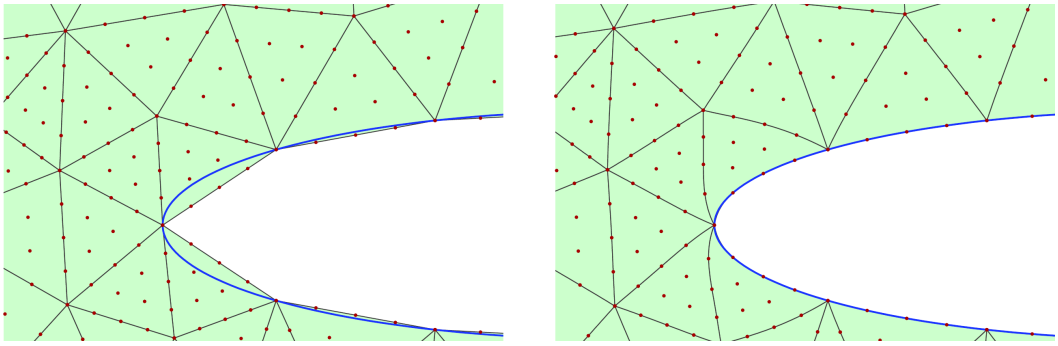


Figure 1.1: Construction of a high-order curved mesh (right) from a linear mesh (left). The edges of the two-dimensional mesh get curved to the boundary of the geometry. Even edges, which do not lie on the boundary of the geometry, get curved to avoid invalid cells. By courtesy of Per-Olof Persson taken from [8].

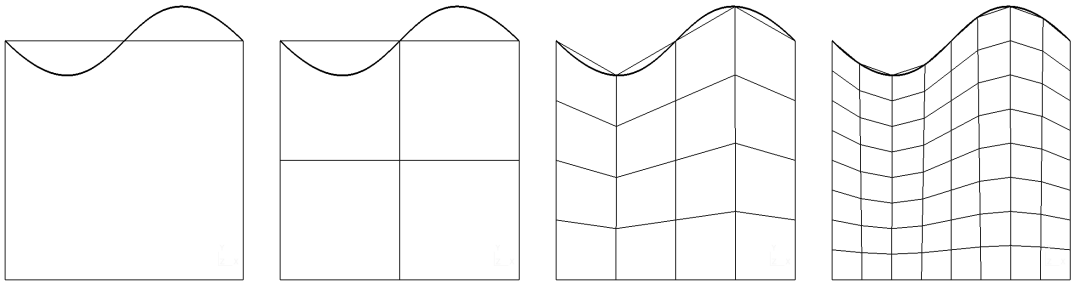


Figure 1.2: Level 0 - 3 refinement of a quadrilateral cell. The cell has a one edge linked to a spline and the volume map influences the whole two-dimensional space inside the cell during the refinement.

This process of creating sub-elements is called adaptive mesh refinement (AMR) and is another way to save computational resources. With AMR, we can refine the cells of an input mesh and can therefore increase the spatial discretization anywhere needed. This leads to less computational overhead in regions, where the solution does not need to be as accurate [11]. More on that in Sec. 2.1. The resulting mesh management can refine high-order, geometry-linked cells and is for that reason able to provide the numerical application with even more options than before.

With the proposed approach, which combines high-order, curved meshes with the usage of AMR, no information is lost, because we directly link the geometries to the cells and are hence always able to generate new points during a simulation. This could be used to refine or coarsen elements or to reduce the order of an element, all while staying true to the original geometry.

1.1 Research question

The in [9, 10] constructed volume map was created to generate geometry-accurate meshes, with the expectation, that this would benefit the accuracy of simulations. Now have to evaluate if these expectations hold true. Therefore, we are going to evaluate the accuracy and performance of the curved hexahedral AMR with a numerical application.

The hexahedral volume map was implemented in `t8code` [12, 13, 14], a parallel mesh management library, which can perform tree-based, dynamic AMR on hybrid meshes. In summary, this means that `t8code` uses a mesh generated by a third-party application and manages it during a numerical simulation. The management consists of mesh alteration, like the refinement of cells or the partitioning of the mesh data across processes. More on that in Sec. 2.1.

Currently, the `t8code` developers provide two solvers together with t8code. A linear finite volume (FV) solver for the advection equation and the discontinuous Galerkin (DG) solver `t8dg` [15, 16] for the advection-diffusion equation. Both solvers can use the features of `t8code`, but not to the same extent. Because the advection solver is a linear FV solver, it cannot take advantage of the high-order mesh elements `t8code` can provide. But it can take advantage of the curved AMR, which describes the geometry with linear elements as best as possible.

Contrary to that, `t8dg` can take advantage of the high-order mesh capabilities. But `t8dg` has other drawbacks. Due to the simultaneous development of `t8code` and `t8dg`, not all features in `t8code` are usable in `t8dg`. One of these features is the implementation of different cell geometries. `t8dg` has its own geometry implementation and hence can not use the geometry implementation of `t8code`.

To use the geometry implementations from `t8code`, the functionality of `t8code` has to be expanded. `t8dg` for example needs to know the orientation of a cell to its neighboring cells so that the nodes and polynomials on the cell faces align with each other. This is for example not necessary for the linear advection solver because it only computes the surfaces of the faces and uses this to calculate the flux to a neighboring cell, no matter the orientation.

Moreover, `t8dg` needs to evaluate the Jacobian of the cell geometry, to calculate determinants, gram determinants, and the normal vectors of the cell faces. The Jacobian of the cell volume description could be evaluated numerically with the difference quotient. But it is to assume, that the numerical evaluation of the Jacobian with a difference quotient is more expensive than an analytical definition. Furthermore, the approximation with difference quotients can be too inaccurate and relies on the definition of step sizes, which are not easy to define, if the volume of an element changes with each refinement level. Therefore an analytic calculation of the Jacobian should be the goal.

Additionally to the setup of the solver and the actual simulations for the evaluation of curved hexahedral AMR, we have to implement two more features. The first feature is the import and geometry linking algorithm for meshes generated by a mesh generator.

Currently, `t8code` can import linear meshes in `Gmsh`'s [17] *msh* format of version 2.x. But to take full advantage of the curved hexahedral AMR, we have to implement the import of `Gmsh`'s newest *msh* 4.x format, which supports parametric nodes, which enable us to use the curved hexahedral AMR on it. More on that in Sec. 3.1. With this, we can use more complex and therefore more practical meshes for our evaluation.

The second feature we have to implement is an improvement of the hexahedral cell volume description proposed in [9, 10]. Due to the import of parametric meshes from `Gmsh`, some flaws in the cell volume description become observable and need to be addressed, which is explained in more detail in Sec. 3.2.

Due to the aforementioned features, which have to be implemented additionally to the evaluation, the adaptation and usage of `t8dg` are beyond the scope of this thesis. We expect to obtain sufficient results from the usage of the advection solver, which is already capable of using the cell geometry description, which also benefits from the curved hexahedral AMR. Hence, the research question can be defined as:

How does the performance and accuracy of the hexahedral curved domain AMR algorithm compare to linear AMR when solving the advection equation with the linear FV method?

1.2 Methodology

To answer this research question, we will give a short overlook of all needed fundamental concepts like AMR and geometry representations in Chapter 2.

After that, in Chapter 3, we are going to look at the preliminary works introduced in Sec. 1.1. These have to be performed before the actual evaluation can start. This includes the import of `Gmsh`'s parametric meshes in the *msh* version 4.x file format and the improvement of the cell volume description. After we looked into these topics, we can also look at different high-order mesh generation implementations. We will look at the problems high-order mesh generation faces, and how other implementations deal with them.

Then, we will define and perform the actual evaluation in Chapter 4. This includes the definition of our application scenarios, which have to be based on the usage of CAD geometries and their influence on an advected concentration of particles. The scenarios will be fairly limited in their complexity, due to meshing and flow field restrictions. But we can see them as a simplified form of engineering problems like the mixing of two components in chemical machines or the mixing of gas and air in carburetors.

Furthermore, we will define a metric, which we use to rate the performance and accuracy of curved hexahedral AMR. We then set the simulations up and evaluate the results with the a priori defined metric.

In Chapter 5 we will give an overlook where curved hexahedral AMR is applicable and if we get any benefits like a higher accuracy from it. Last, in Chapter 6, we will conclude our results and give an outlook of the research that will be conducted subsequently to this thesis.

2 Fundamental concepts

In this chapter, we will take a look at the fundamental concepts needed later on. First, we will give a brief introduction into AMR and the AMR library `t8code` [13, 18]. After that, we will examine which geometry representation we use in our hexahedral cell volume description and the geometry library `OpenCASCADE Technology (OCCT)` [19]. Last, we explore how the advection solver of `t8code` works.

2.1 Adaptive mesh refinement

For many numerical applications (e.g. for solving PDEs), a mesh is needed to describe the computational domain Ω and the geometries therein. The adaptation of this mesh to the need of the numerical application is called adaptive mesh refinement (AMR). Adaptation means here that the elements of the mesh are not restricted to one size and can therefore adapt their size to a priori defined criteria. Often used criteria are for example the distance to a certain geometry, the curvature of a geometry, different terms in the solution of a PDE in this element or discontinuities in the solution of a PDE [20, 21].

There are different concepts for AMR, but because of their relevance for `t8code` we want to describe unstructured and tree-based AMR in particular; an example for each can be seen in Fig. 2.1.

Unstructured adaptive mesh refinement

Unstructured AMR describes the gradually varying size of mesh cells in an unstructured mesh. The variation in mesh size can for example be defined wit the proximity of this cell to an geometry or a specific region in the mesh.

These unstructured meshes are usually constructed by mesh generators, which use different methods for this mesh generation. One method for example is the technique `blockMesh` and `snappyHexMesh`, two mesh generators of `OpenFOAM` [22], use.

`blockMesh` creates a block-structured hexahedral background mesh. The shape of this mesh is defined in a dictionary, from which `blockmesh` builds the mesh. This step happens without any geometry, the user defines parameters for the mesh and then it gets generated.

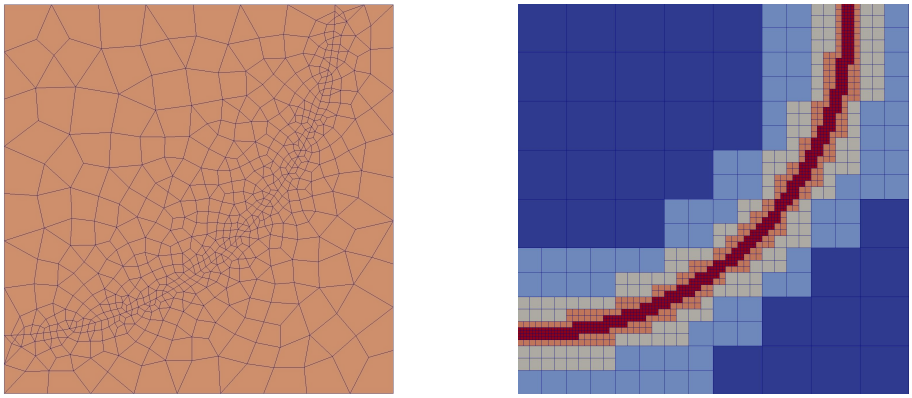


Figure 2.1: Two different examples of AMR. Unstructured AMR (left) is able to alter its mesh size in accordance to a a priori defined criterion, here the distance to the quarter circle. Tree-based AMR (right) changes its mesh size through refinement, here in the form of quadtrees. The different colors show how often an element is refined, its refinement level r. By courtesy of Johannes Holke taken from [13].

This background mesh is already usable, but it is often necessary to adapt it to a geometry. `snappyHexMesh` loads a geometry into the background mesh and then removes all cells, which are either inside or outside of the geometry. In further steps, `snappyHexMesh` can refine user-defined cells of the mesh as octrees (a hexahedral cell, refined into eight hexahedral subelements, two in each direction) to enhance the discretisation of the domain. Moreover, `snappyHexMesh` can snap the exposed nodes of the mesh to the geometry and inflate boundary layers around the geometry, to enhance the geometry discretisation and boundary layer.

The resulting mesh is unstructured, which means that it cannot be saved efficiently in two- or three-dimensional arrays and the neighborhood relations between the cells have to be expressed explicitly. Furthermore, this mesh is nonconforming, so hanging nodes, edges and faces occur. We call a node, edge or face hanging when it does not align with the nodes, edges or faces around it. A hanging node therefore lies on an edge or a face of a cell. This is observable in the right picture of Fig. 2.1. Hanging nodes, edges, and faces are not generally supported by every numerical application. PDE solvers need to be adapted to handle them and hence a mesh is more applicable if it is conforming.

Another way to generate meshes with unstructured AMR is used by the mesh generator `Gmsh` [17]. `Gmsh` can use different meshing algorithms, but here we focus on the advancing front algorithms [23, 24]. First, `Gmsh` sets a node on each zero dimensional entity, which refers to the vertices of the geometry. To influence the mesh size, the user can determine a mesh size at these vertices and `Gmsh` accounts for this mesh size in the following meshing process.

Then, Gmsh places nodes and one dimensional elements (line elements) on the one dimensional entities (curves) in accordance with the defined mesh sizes at the points. After that, an advancing front algorithm is used to place nodes and two-dimensional elements (triangles and quadrilaterals) on the two-dimensional entities (surfaces). Last, Gmsh uses a three-dimensional advancing front algorithm to generate nodes and the three-dimensional elements (tetrahedra, hexahedra, prisms and pyramids) in the volumes.

We yield an unstructured mesh, but this time it is conforming and therefore does not have any hanging nodes, edges, and faces. Such a mesh is shown in the left picture of Fig. 2.1.

One drawback of unstructured AMR is that each cell, its nodes and relations to other cells have to be saved and accessed every time the cell is evaluated. This can become a problem with large-scale meshes, where the memory does not suffice. With structured meshes for example, we can deduct information like that from their storage arrangement in two- or three-dimensional arrays.

Generally, unstructured AMR is static. That means, that the geometry of the mesh is constant throughout the whole simulation. But there are methods like mesh morphing or a moving mesh, which allow to alter it during a simulation, without the need to re-mesh the whole domain [25, 26]. Furthermore, unstructured AMR can be equipped with a dynamic AMR structure like tree-based AMR, to dynamically change the local mesh size during a simulation.

Tree-based adaptive mesh refinement

Tree-based AMR is based on refinement trees. A refinement tree is a type of data structure, which saves the subelements of a cell. Picture a cell shaped like the unit square (see Fig. 2.2). We call this cell the root of the refinement tree. We can split the cell into four smaller square-shaped, non-overlapping elements, which we can also split independently into smaller square-shaped elements. By doing this, we get a finer mesh and increase the number of elements. All these elements grow as branches out of the root or their ancestral elements and we obtain a tree-like structure, which is depicted on the right of Fig. 2.2. We denote this structure a quadtree. The three-dimensional form of this is called an octree [11, 13].

But the elements are not saved in the tree-like structure. They are transformed into a one-dimensional array with the help of a space-filling curve (SFC). The array includes all elements at the outermost end of each branch. This has many advantages for the mesh management, which is important during the numerical computation later on. It is for example not necessary to save the coordinates of the nodes of each element, because they can be computed from the node coordinates of the cell every time the element's geometry is accessed. Furthermore, it is easier to partition the elements on different Message Parsing Interface (MPI) ranks because of their storage in an array. More on the advantages of this form of storage can be found in [11, 13, 27].

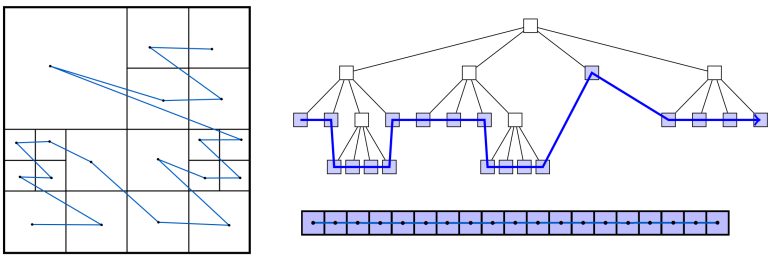


Figure 2.2: A refinement tree is a data structure, which saves the subelements of a root cell inside a space-filling curve (SFC). By courtesy of Johannes Holke taken from [13].

Generally, there are two types of practical implementations of tree-based AMR. The first one uses one single refinement tree and cell for the whole domain. This cell is refined to the necessary level at the boundaries of the geometries inside the domain. After that, all cells inside the geometry are discarded (see left of Fig. 2.3) [28]. The second way uses an input mesh generated by a mesh generator and uses each cell of this input mesh as a root for a refinement tree (see right of Fig. 2.3). This way no elements have to be cut out and the geometry is already approximated by the initial mesh [13, 29, 30].

Implementation in `t8code`

`t8code` is a scalable, parallel mesh management library built on the concept of tree-based AMR and uses a conforming input mesh to compute its trees. We denote this input mesh a coarse mesh or short cmesh. Because of the often interchangeable usage of the words cell and element, we want to clarify that we use the word cell to refer to the cell/tree of a coarse mesh and we use the word element to refer to a cells descendant. This is important, because the cell includes the tree, the nodes, their coordinates and its sub-elements. The cell's geometry is used to calculate the element's geometry.

`t8code` expands the concept of quad- and octrees to more element types and provides SFCs for them. `t8code` supports vertices, lines, triangles, quadrilaterals, tetrahedra, hexahedra, prisms and pyramids as element types.

Furthermore, `t8code` provides management algorithms like mesh adaptation (`adapt`), 2:1 balancing (`balance`), load balancing (`partition`) and ghost layer creation (`ghost`). The `adapt` algorithm refines or coarsens the mesh in accordance with a user defined refinement function. The `balance` algorithm balances the mesh, so that neighboring elements have maximum level difference of one. `partition` redistributes the workload among different MPI ranks, so that each rank has a maximum of one more or less element than the other MPI ranks. Last, the `ghost` algorithm searches for neighboring elements across cell faces and across MPI ranks. It ensures

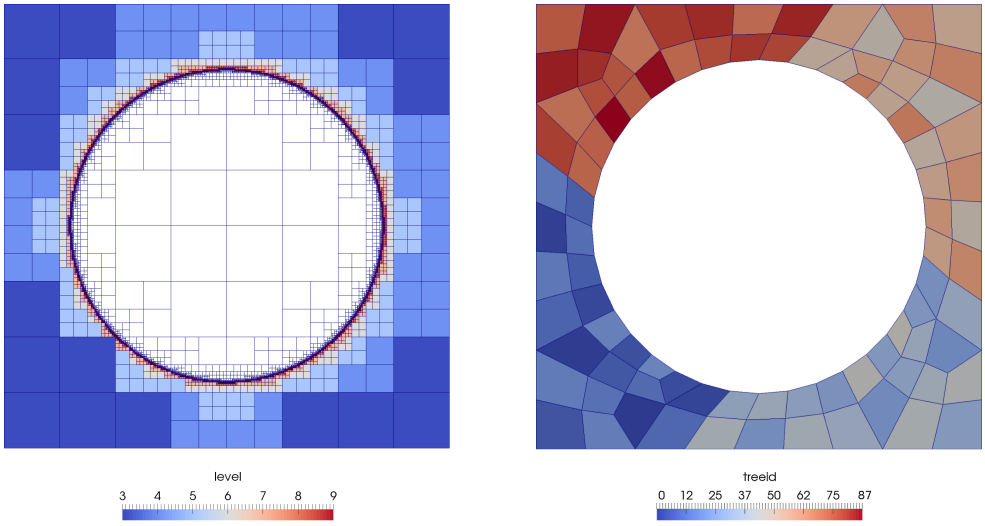


Figure 2.3: Tree-based AMR can be adapted in two different ways. The left image shows an adaptation, which uses only one tree and discards every element, which is not part of the domain. The colors show the refinement level of each element. The right image shows a coarse mesh generated by an external mesh generator. Each cell of the coarse mesh now houses its own refinement tree. By courtesy of Johannes Holke taken from [13].

that every element has access to all its neighbors. More on these algorithms can be found in [13, 14, 27]

`t8code` also provides auxiliary functions like a reader for *msh* files generated by the mesh generator `Gmsh`, [vtk] file output, different cell geometry representations, and cell and element data management. More on that in the tutorials and examples of `t8code` [18].

2.2 Geometry representation

In this section, we want to take a look on the management of our geometries. We use the `OpenCASCADE Technology (OCCT)` [19] CAD library for this task, hence we will focus on the geometry representation used in this library. Due to the complexity and size of the geometry library, we will only give a brief summary of the aspects relevant for our concepts.

To prevent misunderstandings while mentioning `OCCT` topologies and curves, we will refer to them as `OCCT` faces, `OCCT` surfaces, `OCCT` edges and so forth. This is necessary, because parts of mesh elements have the same name (faces, edges and vertices).

`OCCT` uses the boundary representation to represent shapes. This means that the shapes are represented by their limits, their boundary. A shape contains topological components like an

OCCT vertex K, OCCT edge Γ and OCCT face Ξ and these hold their geometrical components; OCCT points κ, OCCT curves γ and OCCT surfaces ξ [31].

The geometrical components describe the spatial shape in our domain: An OCCT point is of dimension zero, therefore it can be defined with its coordinates. An OCCT curve is of dimension one and is a map

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

and an OCCT surface is of dimension two and also a map

$$\xi : \Omega_\xi := [u_{\min},\, u_{\max}] \times [v_{\min},\, v_{\max}] \to \mathbb{R}^3,$$
$$(u,\, v) \mapsto \xi(u,\, v).$$

Contrary to that, solids of dimension three are not a map and have no geometry.

The topological components describe the relations and the spatial extent of the geometric components. A volume is represented by its boundary; a set of connected OCCT faces. This set of connected OCCT faces sets the points inside the solid apart from points outside of the solid. The same applies to OCCT faces and edges. An OCCT face is a surface bounded by OCCT edges, while an OCCT edge is a curve bounded by OCCT vertices. An OCCT vertex is defined by its OCCT point. Furthermore, solids, OCCT faces and OCCT edges know which shapes they are bounded by and what their neighboring shapes are.

The geometries itself are represented via different analytic models, dependent on their properties. A cylindrical OCCT surface is, for example, defined via a cylindrical coordinate system and is therefore not bounded. Both parameters have no limits. The u-parameter has a period of 2π and is defined in radians, whereas the v-parameter defines the axial position.

But the geometries implemented are not limited to elementary curves and surfaces like for example parabola, circles, planes and spheres. There are also more complex geometries like B-spline curves and surfaces or non-uniform rational B-spline (NURBS) curves and surfaces. These parametric geometries are controlled via points and piece-wise defined basis functions and have a much broader flexibility and application areas. The generation and the implementation of these are huge topics on their own and we recommend the NURBS book by Les Piegl and Wayne Tiller as a further reference [32].

These are the basic concepts needed for the next chapters. But as already stated, OCCT is a huge software library and there are many more algorithms for handling and working with

geometries, we therefore encourage the reader to look into the OCCT documentation[1] for more information.

2.3 Advection solver of t8code

t8code features its own solver for the advection equation. Due to the usage of this solver for evaluating the curved hexahedral AMR, we give a brief overview of this solver. Note, that the solver is already existent and that this chapter is based on [13, 18].

2.3.1 Advection equation and the finite volume method

The solver uses the linear finite volume (FV) method to solve the advection equation

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

which tracks, how a given quantity of ϕ is advected by a flow $\vec{u}$. In our case the flow u is divergence free, therefore we can simplify the equation to

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

To adapt the equation to our FV mesh, we integrate the equation over each volume and after applying the Gaussian divergence theorem

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

with the normal vector $\vec{n}$, we get

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

We can now discretize the domain $\Omega \in \mathbb{R}^3$ into non-overlapping elements E with i faces $\mathscr{F}$. Moreover, we approximate $\phi(\vec{x},\, t)$ with the constant value $\phi_{E,\, t}$ which holds for each element E. We also introduce the flux $\psi(E,\, E'_j;\, \mathscr{F}_j)$ through face $\mathscr{F}_j$ into the neighboring element E'_j.

[1] https://dev.opencascade.org/doc/overview/html/

The flux is always orthogonal to the face and the magnitude of the flux is determined by the flow $\vec{u}_{\mathscr{F}_j,\,t}$ at the midpoint of the face $\mathscr{F}_j$ and the area A of the face:

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

For faces without hanging nodes $\psi(E,\,E'_j;\,\mathscr{F}_j) = -\psi(E'_j,\,E;\,\mathscr{F}_j)$ is true. But if a face has hanging nodes, the flux has to be computed with the area and flow of the sub-face F'. More on this and the challenges of finding face neighbors of a different refinement level can be found in [13].

When we apply this to our mesh we get

$$\frac{\partial}{\partial t}\iiint_E \phi_{E,\,t}\mathrm{d}V + \sum_{j=0}^{i-1}\iint_{\mathscr{F}_j}(\phi_{E,\,t}\vec{u}_{\mathscr{F}_j,\,t}\cdot\vec{n}(\mathscr{F}_j))\mathrm{d}A = 0. \tag{2.1}$$

After the spatial discretisation, we can discretize in time. For this we use the difference quotient and the volume of an element V. We yield

$$\begin{aligned} 0 &= \mathrm{V}(E)\frac{\phi_{E,\,t+\Delta t} - \phi_{E,\,t}}{\Delta t} + \sum_{j=0}^{i-1}\psi(E,\,E'_j;\,\mathscr{F}_j) \\ \Leftrightarrow \phi_{E,\,t+\Delta t} &= \phi_{E,\,t} - \frac{\Delta t}{\mathrm{V}(E)}\sum_{j=0}^{i-1}\psi(E,\,E'_j;\,\mathscr{F}_j). \end{aligned}$$

This equation is solved for each time step Δt for each element E and their faces $\mathscr{F}$.

2.3.2 Courant–Friedrichs–Lewy number

The Courant–Friedrichs–Lewy (CFL) number originally described how many identical rectangular elements E with the length $\mathrm{len}(E)$ an event with the velocity u can pass in a given time step Δt:

$$\mathrm{CFL} = \Delta t\frac{u}{\mathrm{len}(E)}.$$

It is stated, that the solution of a finite difference equation is not able to converge with the solution of the corresponding differential equation, if $\mathrm{CFL} > 1$ [33].

In our case, the elements E are not identical, they are not even all rectangular and the flow $\vec{u}_{E,\,t}$ has different magnitudes throughout t and Ω. Hence, we use an adapted version of the CFL number, which is defined for each element at each time separately:

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

Here, $\mathrm{V}(E)$ is the volume of en element E and $\mathrm{V}(E)^{1/d}$ resembles the mean diameter of the element of dimension d. The global CFL number is the maximum CFL number of all elements.

In practicality, the global CFL number we use to define the length of out time steps Δt needs to be much smaller than 1 to ensure stability. One reason for that could be the ever changing volumes of our elements during refinement or coarsening.

2.3.3 Level-set function and refinement criterion

The advection solver solves the advection equation for a domain Ω. At the start of the simulation we want to have two different sub-sets, one for the advected concentration $\Omega_{1,\,t=0}$ and one for the remaining space $\Omega_{2,\,t=0}$. Together they form the whole domain $\Omega_{1,\,t} \cup \Omega_{2,\,t} = \Omega$. We can distinguish them in their value of $\phi_{E,\,t}$. An appropriate initial condition would be

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

where $\mathrm{dist}(\overline{\Omega_{1,\,0}} \cap \overline{\Omega_{2,\,0}})$ is the distance of the element from the interface between both subsets $\Omega_{1,\,0}$ and $\Omega_{2,\,0}$.

This way, we can distinguish them during the whole simulation by their sign:

$$\Omega_{1,\,t} = \{E \in \Omega | \phi_{E,\,t} \geq 0\},$$
$$\Omega_{2,\,t} = \{E \in \Omega | \phi_{E,\,t} < 0\}.$$

With this distinction, it is possible to use the value of $\phi_{E,\,t}$ as a refinement criterion. This way we get a higher resolution for the interface between both subsets and it is easier to track it. The used refinement criterion consists of a parameter b and h, which is defined as the average diameter of an element $h = \mathrm{V}(E)^{1/d}$, where d is the dimension of the element. We refine an element if

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

is true. The parameter b steers, how thick the refinement band around the interface is.

3 Preliminary works

Before we begin the evaluation of curved hexahedral AMR, we have to implement some preliminary features. These features help us in the set up of the simulations later on. First, we have to enable `t8code` to read parametric meshes. This enables us to use an external mesh generator and therefore to use more practical and complex meshes. Second, we have to improve the cell volume description we introduced in [9] due to occurring inaccuracies. Last, we will look at other implementations of high-order curved meshes and what we can learn and adapt from these implementations. But because the adaptation is not yet required and beyond of the scope of this evaluation, it remains fully theoretical.

3.1 Import of `Gmsh`'s parametric meshes

In this section, we will take a look at the import of parametric meshes generated by the mesh generator `Gmsh` [17]. This includes the extension of the existing mesh reader of `t8code` to the newest mesh format and addition of the ability to parse geometry information. Furthermore, we will introduce a short example of how to use this new file reader.

3.1.1 Update to msh format 4.x

`t8code` can already import meshes in `Gmsh`'s *msh* version 2.x format[1]. This format is sufficient for the pure import of nodes and cells. But with the introduction of the geometric cell volume description, we have new requirements. These requirements can be met by the *msh* version 4.x format[2]. It supports the storage of additional geometry information, which we can use for this cell volume description. More on the import and processing of this information in Sec. 3.1.2. Here in this section, we give a brief overlook on what changed and what we have to adapt. We will not get into much detail, due to the triviality of the node and cell import.

The *msh* format uses different sections, which are defined via a start and an end tag. We need the section with the nodes and the section with the cells. The section with the cells is called ”elements” in the *msh* file, but due to our in Sec. 2.1 introduced distinction between cells and

[1] `https://gmsh.info/doc/texinfo/gmsh.html#MSH-file-format-version-2-_0028Legacy_0029`
[2] `https://gmsh.info/doc/texinfo/gmsh.html#MSH-file-format`

elements, we will refer to this section as the cells section. The other sections carry information we do not need or whose usage is not yet implemented in t8code.

First, we have to adapt the file reader to the changed node section. In *msh* version 2.x, the first line gives the total amount of nodes in this file. Each following line then holds a node index and the coordinates of the node. This changes in *msh* version 4.x due to the introduction of geometries, called entities by Gmsh. The nodes are now sorted by the dimension of the entity they lie on. First, all nodes on one-dimensional entities are listed. After that, all nodes on two-dimensional entities are listed. The nodes are also clustered by their specific entity. This means, that all nodes on curve 1 are clustered, then all on curve 2 and so on.

To parse the *msh* version 4.x format, we first have to parse the first line of the section. It describes how many clusters and nodes there are and what the smallest and biggest node index is. We can use this information later on to control if the parsing was successful or if any errors occurred.

After that, we can iterate over each cluster and parse the information therein. Due to Gmsh always saving cells of all dimensions (0D: point; 1D: line; 2D: triangle, quadrilateral; 3D: tetrahedron, hexahedron, prism, pyramid), we can omit the clusters of the unwanted dimension.

The clusters comprise a description of the entity the nodes lie on. This is relevant for the linkage with the OCCT geometries described in Sec. 3.1.2. Hence we save this information for each node in this cluster. After that, the node indices and the node coordinates and parameters follow. The parameters are also relevant for Sec. 3.1.2 and are saved along the other information.

After the parsing, we now have a list of all relevant node indices, their entities, parameters and coordinates. For easier searchability, which is relevant in the next step, we store these nodes inside a hash table with their index modulo the total amount of nodes as hash.

Now we can import the cells from the cells section of the *msh* file. Just like in the nodes section, the cell section is clustered by the entities the cells are generated on. But since we already saved this information with our nodes, we can ignore it here.

The first line again comprises the number of clusters, number of cells and the min and max cell index. We can use this information for error checking, just like in the nodes section.

After that, there are the clusters in which the cells are stored. Each cell has an index and its nodes, but the index is irrelevant for us. We can use the hash table with the nodes to quickly find all nodes of the cell. We then use the node coordinates to build a coarse mesh cell in our cmesh. We also save a list of the *msh* node indices for each cmesh cell for later usage.

Furthermore, we can also retrieve the geometry information of the nodes and use our algorithm described in Sec. 3.1.2, to link the geometries to our coarse mesh cell.

Lastly, after the import of all cells in the *msh* file, we have to link the faces of neighboring cells together. This is needed to store the face connectivity of the mesh. This algorithm does not need

to be modified for the import of *msh* version 4.x files, but we briefly discuss this step for the sake of completeness.

For this, we can use the list of *msh* node indices of each cmesh cell which we saved beforehand. We know that the neighboring faces have the same *msh* node indices and therefore, we can iterate over each cmesh cell and all their faces. The cmesh cell id and face id are stored in another hash table, using the sum of the *msh* node indices as hash value.

During the iteration over each cell face, the hash table can tell us, if the corresponding neighbor face was already saved. The hash table checks this via the hash value (neighboring faces have the same nodes and therefore the same hash) and compares the nodes of each face with the same hash value. Last, we can compute their orientation and link them together.

3.1.2 Parsing and processing of the geometry information

As already stated, Gmsh has the ability to export parametric meshes in the *msh* format. In this format, each node has information about the geometry it was generated on. The information includes an index and dimension of the geometry and the parameters on the geometry. The nodes on an OCCT vertex have no parameters, because no parameters are needed on a zero-dimensional object. Nodes on OCCT edges have one parameter u and nodes on OCCT faces have two parameters u, v. Nodes generated inside a solid are also labeled with the respective geometry index and dimension, but due to the lack of three-dimensional parameters in the boundary representation, no parameters are given. But we do not need them because of the already mentioned and implemented cell volume description.

The cell volume description interpolates between the parameters of a geometry to evaluate how the space inside the cell has to be curved. Let us assume a hexahedral cell with a OCCT surface linked to one of its faces. If we want the coordinates of an arbitrary point on a cell face, we normally interpolate bilinearly between the coordinates of the four nodes of the face. But in case of our curved cell, we also have the parameters of the four nodes on the OCCT surface. This means that we obtain the coordinates C of a node $\mathcal{N}$ if we evaluate the OCCT surface ξ with the parameters P of the node $C = \gamma(P_{\mathcal{N}})$. But in case of our quadrilateral, geometry-linked face, we interpolate between the parameters of the four nodes and evaluate the OCCT surface with these interpolated parameters. This way we retrieve the coordinates of an arbitrary point on the geometry-linked cell face. The already implemented cell volume description then uses this point and maps it inside the cell, so that we are also able to evaluate points inside the cell volume.

To import a parametric mesh generated by Gmsh we have to convert the data supplied by Gmsh into the data the cell volume description needs. We already mentioned the processing of the coordinates and cells in Sec. 3.1.1. But now we discuss the processing of the geometry

information. Note, that the proposed algorithms are valid for all cells, not only hexahedra. This way they do not have to be adapted to other cell types in the future.

The cell volume description needs to know, which geometries each edge and face of the cell has to be linked to. And if an edge or face is linked to a geometry, the cell volume description needs the parameters of the edge and face nodes on said geometry. This poses a problem, because this information is not always given directly by Gmsh. Since a node can only contain the information about one geometry, while lying on more than one geometry, too few parameters are provided. An illustration of this problem is shown in Fig. 3.1.

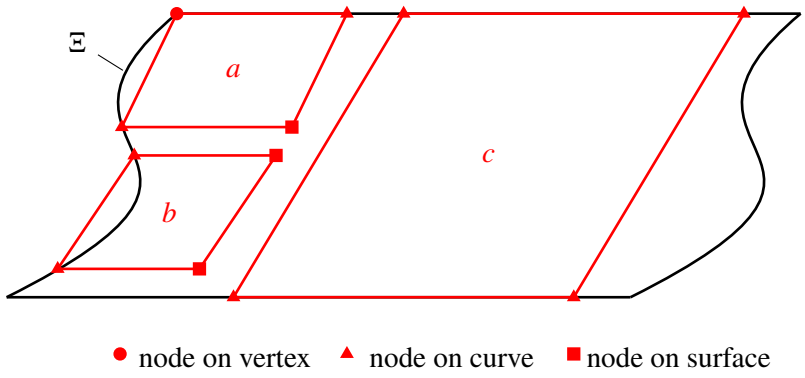


Figure 3.1: Shown is an OCCT face Ξ and different exemplary cell faces a, b, c generated on Ξ. The nodes of the cell faces are not always on the surface, but some are on vertices or curves.

Here we can see a OCCT face Ξ and some exemplary cell faces a, b, c generated on this OCCT face. We see that the nodes of face a lie on different geometries. The node on the OCCT vertex has no parameters, the two nodes on the OCCT edge have one parameter on their respective OCCT edge and the node on the OCCT face has two parameters. This means we cannot interpolate between the OCCT face parameters, because three of the four nodes do not know their parameters on the OCCT face.

Moreover, we do not even know, that face c was generated on the OCCT face, because no node of c has parameters on the OCCT face, and therefore, the nodes only tell us about the OCCT edges they were generated on.

We can solve both of these problems algorithmically using OCCT. OCCT provides us with functions to retrieve the parameters of an OCCT vertex on an OCCT edge or OCCT face and there is also a function to convert an OCCT edge parameter into an OCCT face parameters. Due to the existence of multiple functions, which could be used to convert these parameters, we give a short discussion about them in Appendix A.

Moreover, we can use OCCT to explore the topology of the OCCT vertices, edges, and faces of the meshed shape. The topology tells us, on which OCCT edges an OCCT vertex lies and on which

OCCT faces an OCCT edge lies. We can use this functionality to look at which OCCT face both of the OCCT edges of face c lie on. Then we can convert the curve parameters of the nodes of c into the OCCT face parameters and provide them to the cell volume description.

Algorithm for the linking of faces

The used algorithm, which covers all possible generation scenarios for faces is shown in Alg. 3.1.1. It uses, as mentioned before, OCCT to access the topology information of our shape and to retrieve and convert parameters.

Algorithm 3.1.1: `t8_link_faces` (Nodes $\mathcal{N}$ of a cell from the *msh* file, a cmesh cell C)

```
   Result: The resulting cell C with geometry-linked faces
1  for each face 𝓕 of C do
2      for each node 𝒩_𝓕 of the face 𝓕 do
3          if Ξ in 𝒩 then
4              store Ξ in 𝓕
5              break
6      if no Ξ in 𝓕 then
7          for each node 𝒩_𝓕 of the face 𝓕 do
8              search and save different Γ
9          for each two neighboring nodes 𝒩_𝓕1, 𝒩_𝓕2 of the face 𝓕 do
10             if two different vertices K in 𝒩_𝓕1, 𝒩_𝓕2 and the two K share a Γ then
11                 save Γ
12         if found two different Γ and the two Γ share a Ξ then
13             store Ξ in 𝓕
14     if Ξ in 𝓕 then
15         for each node 𝒩_𝓕 of the face 𝓕 do
16             if 𝒩_𝓕 has parameters P on Ξ of 𝓕 then
17                 store P in 𝓕
18             else
19                 if 𝒩_𝓕 is on Ξ then
20                     use OCCT to get P of 𝒩_𝓕 on Ξ
21                     store P in 𝓕
22                 else
23                     remove Ξ and all P from 𝓕
24                     break
```

It gets a cmesh cell C and iterates over each face $\mathscr{F}$ of C. It then iterates over each node $\mathscr{N}_{\mathscr{F}}$ of $\mathscr{F}$ and checks if any node has information about an OCCT face Ξ. If so, it saves the OCCT face in $\mathscr{F}$. This way we already know on which OCCT face the faces a and b from Fig. 3.1 lie. But if no node lies on an OCCT face, like face c from Fig. 3.1, we still have no information about the OCCT face.

Therefore, we then again iterate over each node $\mathscr{N}_{\mathscr{F}}$ of $\mathscr{F}$ and look for two different OCCT edges Γ. In face c we would find two different OCCT edges, but if a face's nodes would lie only on OCCT vertices, we still would not have identified two different OCCT edges. Hence, we again iterate over each node and search for two different OCCT vertices K. If we have found them we can look if they share an OCCT edge Γ and save this OCCT edge. After we have identified two different Γ, we can finally look, if they share an OCCT surface Ξ.

Now, in line 14, we know if the cmesh face $\mathscr{F}$ lies on an OCCT face Ξ. But we still have to check, if all nodes are really on this OCCT face, and if so, we have to retrieve their parameters and save them in $\mathscr{F}$. To do this, we iterate once more over each $\mathscr{N}_{\mathscr{F}}$. In this loop, we check for each $\mathscr{N}_{\mathscr{F}}$ if it already has parameters P on Ξ. If true, we save them in $\mathscr{F}$. If not, we firstly check, if $\mathscr{N}_{\mathscr{F}}$ lies on Ξ. If true, we use OCCT to convert or retrieve its parameters and then we save them in $\mathscr{F}$. But if $\mathscr{N}_{\mathscr{F}}$ does not lie on Ξ, we remove Ξ and all saved parameters from $\mathscr{F}$, because a face can only be linked, if all nodes lie on the OCCT face. Therefore we then break the loop and start with the next face of C.

Algorithm for the linking of edges

The algorithm for the linking of edges (Alg. 3.1.2) works similarly to the algorithm to link faces. We iterate over each edge $\mathscr{E}$ of the cell C to search for the appropriate OCCT edges Γ. We search by looking at both nodes $\mathscr{N}_{\mathscr{E}1}$, $\mathscr{N}_{\mathscr{E}2}$ of $\mathscr{E}$ and if at least one of them has information about an OCCT edge, we save this edge. If both of the nodes do not have information about an OCCT edge, we look if both nodes have information about two different OCCT vertices K and if these vertices share an OCCT edge. If that is the case, we save this OCCT edge.

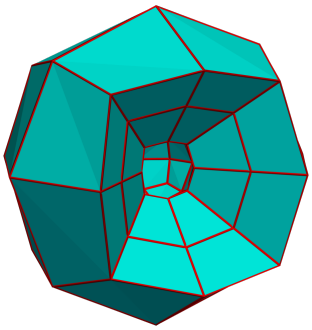
We now know which OCCT edge the cell edge could be linked to, but we still have to look, if both nodes really are on this OCCT edge and maybe we have to retrieve their parameters. We check this for each node separately. First, we look if the node already has a parameter P on Γ. If that is the case, we can store this parameter in our edge. If not, we have to retrieve the parameter. For this, we check if the OCCT vertex K of the node lies on the OCCT edge Γ. If so, we use OCCT to retrieve its parameter and store it in $\mathscr{E}$. If not, we know, that this edge $\mathscr{E}$ is not linked to any OCCT edge and therefore we remove Γ and all already saved parameters from $\mathscr{E}$. We then repeat this process with the next edge of the cell.

Algorithm 3.1.2: `t8_link_edges` (Nodes $\mathcal{N}$ of a cell from the *msh* file, a cmesh cell C)

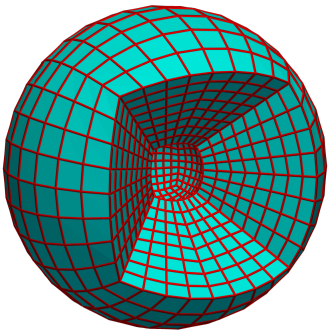
Result: The resulting cell C with geometry-linked edges

```
1  for each edge ℰ of C do
2      if Γ in 𝒩_ℰ1 or Γ in 𝒩_ℰ2 then
3          save Γ in ℰ
4      else
5          if K in 𝒩_ℰ1 and K in 𝒩_ℰ2 and both K share a Γ then
6              save Γ in ℰ
7      if Γ in ℰ then
8          for each node 𝒩_ℰ of the edge ℰ do
9              if 𝒩_ℰ has parameters P on Γ of ℰ then
10                 store P in ℰ
11             else
12                 if 𝒩_ℰ is on Γ then
13                     use OCCT to get P of 𝒩_ℰ on Γ
14                     store P in ℰ
15                 else
16                     remove Γ and all P from ℰ
17                     break
```

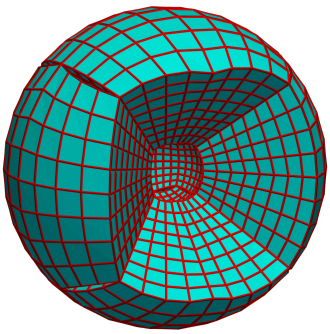
By using the aforementioned algorithms, we now can import Gmsh's parametric meshes. One such imported mesh is shown in Fig. 3.2a and the level $r = 2$ refined mesh is shown in Fig. 3.2b.



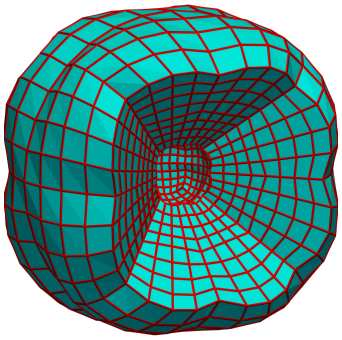
(a) Coarse mesh from *msh* file .



(b) Level $r = 2$ refined sphere shell using the improved cell volume description $G()$ (Eqn. 3.3).



(c) Level $r = 2$ refined sphere shell using the cell volume description $g()$ (Eqn. 3.2).



(d) Level $r = 2$ refined sphere shell using the cell volume description $g()$ (Eqn. 3.2), but with the first improvement from Fig. 3.5.

Figure 3.2: Shown is the mesh of a sphere shell which is imported from a *msh* file and then all cells are refined to level $r = 2$. The refinement was performed with different versions of the cell volume description.

3.1.3 NACA airfoil example

To show some of the new capabilities enabled by the import of parametric *msh* files, we implemented example code[3]. In this code, we use `Gmsh` to generate a NACA 6412 airfoil profile and mesh it structurally with hexahedral cells. We can now use the by `Gmsh` generated *msh* and *brep* file to generate a geometry-linked cmesh in `t8code` and show some explanatory applications.

The first application is a general adaptation and balancing of the curved geometry. `t8code` enables us to define a refinement criterion for the `adapt` algorithm. For this, we use a wall, which

[3] `https://github.com/sandro-elsweijer/t8code/blob/masters_thesis_elsweijer/example/geometry/t8_occ_naca.cxx`

moves through our cmesh in a user-specified amount of time steps. Every element in user-defined proximity now gets refined and every element outside of that proximity gets coarsened.

After running the `balance` algorithm to ensure a 2:1 balancing of the mesh we can export it via the `VTK` [34] application programming interface (API). As we can see in Fig. 3.3, the elements of the geometry-linked approach (right) are curved to the geometry and the refinement happens with respect to the initial NACA geometry. The unlinked approach on the left has no geometry information and can therefore not produce curved elements or refine with respect to the NACA geometry.

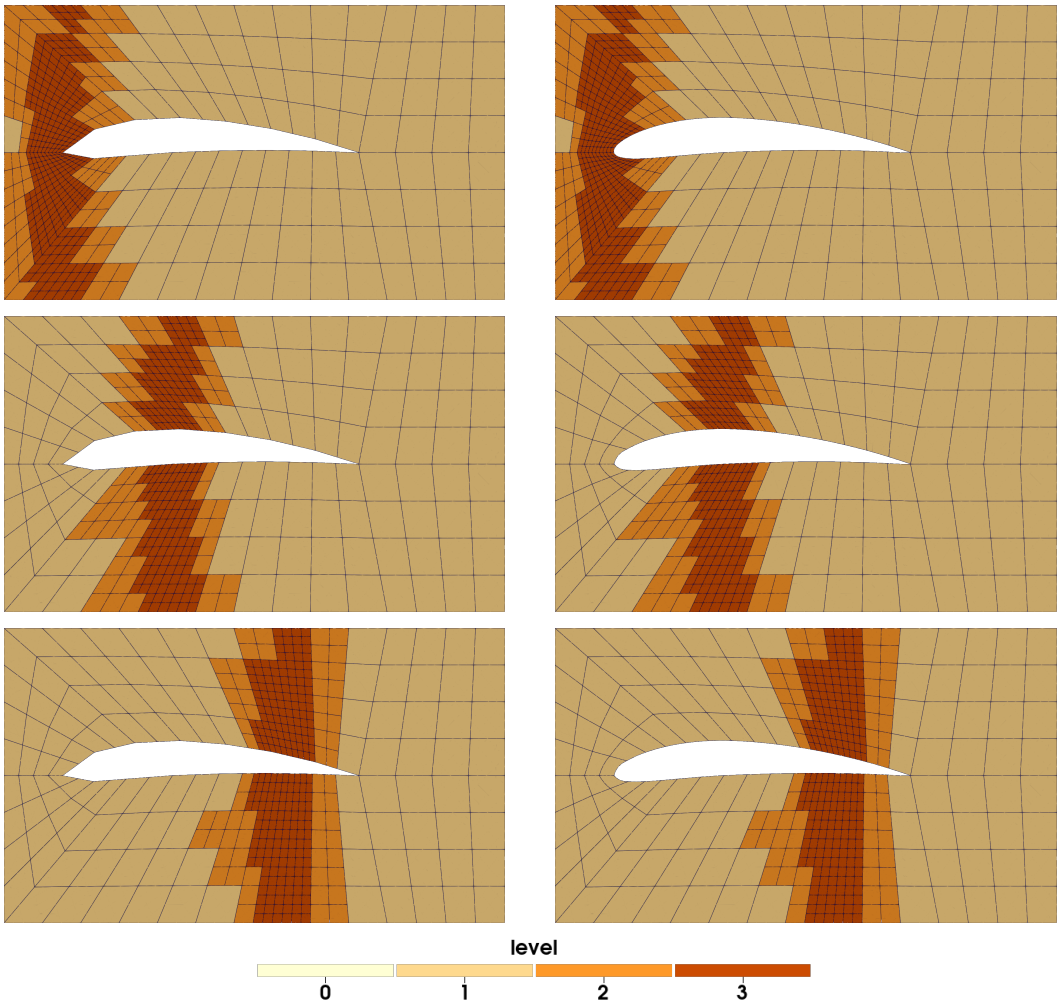


Figure 3.3: Output of the NACA moving wall example. A wall moves through a cmesh and every element in a certain proximity to it gets refined. We can see, that the geometry-linked approach on the right produces curved elements as well as a better geometry representation as the unlinked approach on the left.

The second application of the NACA airfoil example shows how the user can use the geometry data of the cells to define a refinement criterion. In this example, we use Gmsh to get the surface indices of all surfaces of the airfoil. We can then define a refinement criterion, which looks at the linked geometries of a cell and checks if a specific surface is linked.

To do this, we check if the element we are looking at is touching a linked face. We have to look at this first, because refinement happens to elements, but the geometries are linked to the cells. We can now look, which geometry it is linked. This way we can refine our mesh based on the geometries the elements touch. This is shown in Fig. 3.4, where we refined the ventral side of the airfoil to level 2 and the dorsal side to level 3. Note, that this is only possible with a geometry-linked cmesh, because the unlinked, linear cmeshes have no geometry information.

Due to the linkage to OCCT, we can also use all geometry evaluation tools provided by it to generate refinement criteria. We could use them for example to evaluate the curvature of the geometry at specific points and refine based on that curvature.

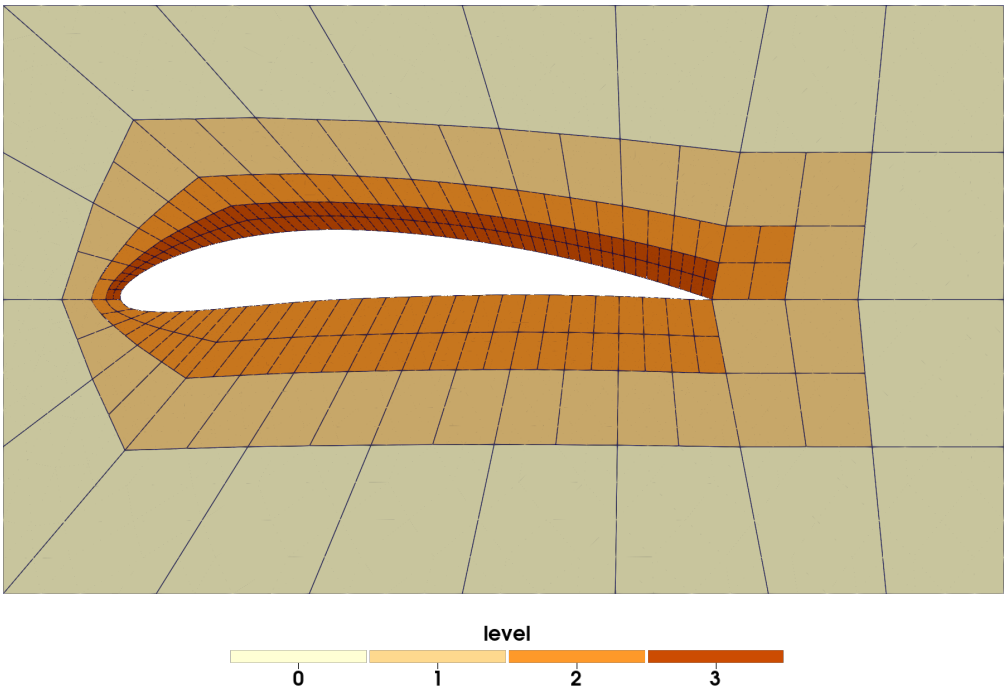


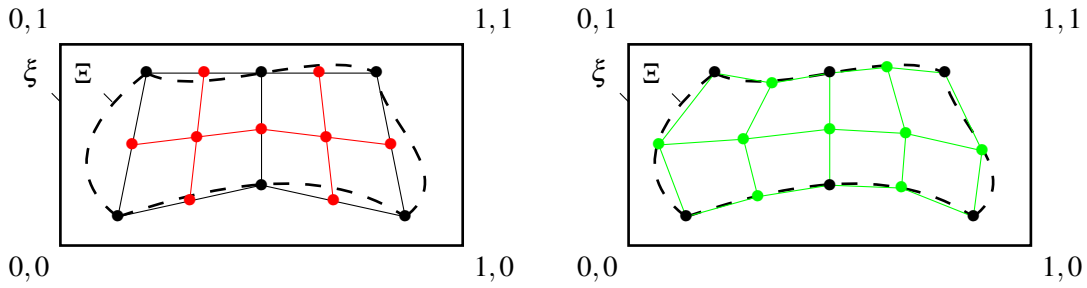
Figure 3.4: Output of the NACA surface index example. We can see, that the elements touching specific surfaces of the NACA profile are refined. This is only possible, because the cmesh cells know, which geometries they are linked to. This cannot be reached with the unlinked approach, because no geometry information is available.

3.2 Improvement of the cell volume description

During the testing of the aforementioned import algorithm for parametric meshes, some inaccuracies in the cell volume description from [9] became apparent. These inaccuracies only happen due to specific circumstances which apply in the import of geometries. An example of the inaccuracy can be seen in Fig. 3.2c.

The level $r = 2$ refined sphere has some slits along the volume boundaries. The cause of this misrepresentation is how the boundary representation of OCCT works. Fig. 3.5 shows what currently happens and what should happen. Here we can see a surface ξ and its bounded section Ξ. The parameters of the corners of ξ are denoted. On Ξ are two cell faces, shown in black. The faces of the refined level $r = 1$ elements are shown in red. We can see, that the red nodes generated by bilinear interpolation of the parameters of the black nodes are not on the bounding OCCT edges. Therefore, we have to calculate a factor to correct the misrepresentation of the bilinear interpolation of the surface parameters. We denote this the parameter error correction $\Delta p_{\mathscr{F}}$.

From now on, we will refer to the correction factors to counteract the errors we make with a (tri-/bi-)linear interpolation as error corrections. There will be coordinate error corrections Δc responsible for the representation of the geometries and parameter error corrections Δp responsible to mitigate the misrepresentation shown in Fig. 3.2c and Fig. 3.5.



(a) Linear interpolation between OCCT surface parameters

(b) Linear interpolation between OCCT surface parameters and correction with OCCT curve parameters

Figure 3.5: Shown are two versions of the surface interpolation. In Fig. 3.5a we can see two faces (black) generated on the bounded OCCT face Ξ of the OCCT surface ξ. After the refinement, the emerging new nodes (red) are not on the boundary of Ξ, due to the usage of linear interpolations. In Fig. 3.5b we can see the improved interpolation, which takes the OCCT edges of the OCCT face Ξ into account.

Furthermore, all error corrections are of a similar structure. Generally, they consist of a projection of the input coordinates $X \in [0,1]^3$ onto the edges or faces of the linear cell and onto the geometries, which are linked to these edges and faces. This is shown in a two-dimensional

example in Fig. 3.6. To yield the coordinate error correction Δc_1, we interpolate linearly between the coordinates C of the nodes $\mathcal{N}_2$ and $\mathcal{N}_3$. Furthermore, we also interpolate between the parameters P of the OCCT curve γ_1 of the same nodes. We use this parameter to evaluate γ_1 and the difference between this evaluation of γ_1 and the linear interpolation of the node coordinates is our coordinate error correction Δc_1.

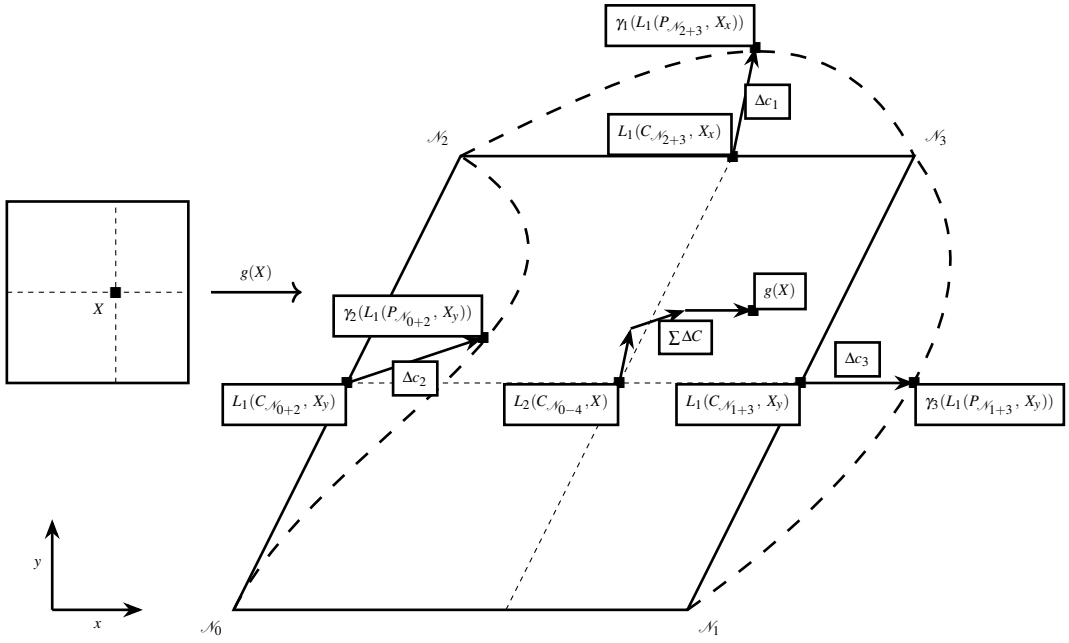


Figure 3.6: Mapping of the unit square to the actual, geometry-deformed cell volume. The map is based on linear (L_1) and bilinear (L_2) interpolations and the coordinate error corrections Δc induced by the linked surfaces and linked edges. The error corrections correct the error we make, if we interpolate linearly between the node coordinates.

Since the influence of the geometry should decrease with the distance of X from the cell face or edge and reach zero on the opposite side of the cell, we scale the obtained coordinate error correction with all coordinates of X, which are orthogonal to the edge or face we are looking at. In this case, we scale Δc_1 with X_y to yield the scaled coordinate error correction ΔC_1. We would scale Δc_2 with $(1 - X_x)$ and Δc_3 with X_x, because they are on opposite sides of the cell.

In addition to the already mentioned misrepresentation, a second inaccuracy becomes apparent. The second inaccuracy occurs only if two neighboring faces carry an OCCT surface. It is visualized in Fig. 3.7 on the left. Due to $\Delta C_{\mathcal{F}}$ being the error correction for a bilinear interpolation between the face nodes, some of the scaled coordinate error correction of neighboring faces overlaps (orange volumes on the left). This can also be observed in Fig. 3.2d. Here the overlap leads to a buckling effect at each linked face which at least on neighboring face, which is also linked.

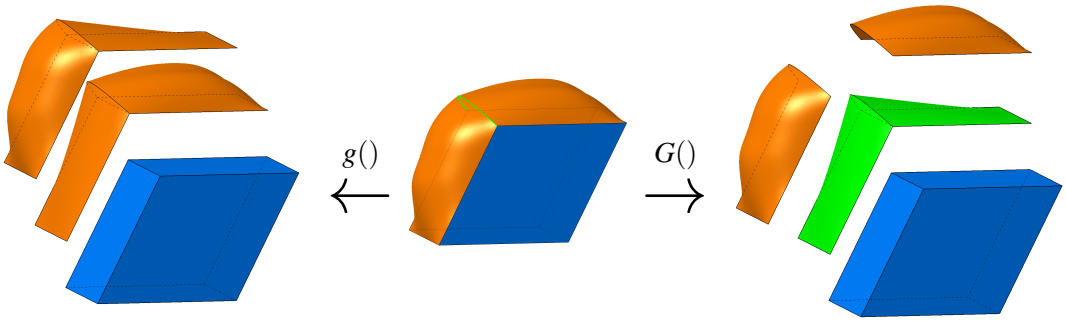


Figure 3.7: Shown is a hexahedral cell (middle), which has two OCCT faces (orange) and an OCCT edge (green) linked to it. The old cell volume description (left) is a sum of the trilinear interpolation in the hexahedral cell (blue) and the error corrections of the OCCT faces (orange). The new cell volume description (right) is a sum of the trilinear interpolation (blue) the error correction from the OCCT edge (green) and the error corrections of the OCCT faces (orange).

The old cell volume description $g(X)$ is defined as

$$g(X) = L_3(X) + \sum_{\mathscr{F}=0}^{5} \Delta C_{\mathscr{F}}(X), \tag{3.2}$$

where $L_3(X)$ is a trilinear interpolation between the cell nodes (blue volume).

To counteract the overlap of the error corrections, we remove it from each face and add it back later. The overlap is the scaled coordinate error correction $\Delta C_{\mathscr{E}}$ of the edge (green volume on the right), which is in between both faces (green line). The resulting geometry description is

$$G(X) = L_3(X) + \sum_{\mathscr{F}=0}^{5} \left(\Delta C_{\mathscr{F}}(X) - \sum_{\mathscr{E}\mathscr{F}=0}^{3} (\Delta C_{\mathscr{E}\mathscr{F}}(X)) \right) + \sum_{\mathscr{E}=0}^{11} (\Delta C_{\mathscr{E}}(X)) . \tag{3.3}$$

Alg. 3.2.1 shows how the improved geometry description $G(X)$ is implemented. The algorithm maps the volume of a reference unit cube $X \in [0,1]^3$ to the cell volume in the computational domain $Y \subset \Omega \in \mathbb{R}^3$. It includes the scaled coordinate and parameter error corrections according to the OCCT curves γ and OCCT surfaces ξ linked to the cell edges $\mathscr{E}$ and cell faces $\mathscr{F}$.

Line 1, in accordance to Fig. 3.7, is the trilinear interpolation $L_3()$ between the node coordinates C using X. The following line starts the calculation of the scaled coordinate error correction of each face $\mathscr{F}$. We iterate over each face and check, if an OCCT surface $\xi_{\mathscr{F}}$ is present. If not, we can skip this face.

To be able to correct the errors addressed in Fig. 3.5 and Fig. 3.7, we then iterate over each edge $\mathscr{E}_{\mathscr{F}}$ of the face and check in line 5 if an OCCT curve $\gamma_{\mathscr{E}\mathscr{F}}$ is present. If not we can skip the edge. But if an OCCT curve is present, we calculate the scaled coordinate error corrections $\sum \Delta C_{\mathscr{F}}$

Algorithm 3.2.1: `t8_geom_evaluate` (coordinates in the reference unit cube $X \in [0,1]^3$)

Result: The mapped coordinates Y of the cell in the computational domain

1 $Y = L_3(C,\ X)$
2 **for** each face $\mathscr{F}$ **do**
3 **if** no $\xi_{\mathscr{F}}$ in $\mathscr{F}$ **then continue** `/* skip if no surface is present */`
4 $\sum \Delta C\mathscr{F} = 0$
5 $\sum \Delta P\mathscr{F} = 0$
6 **for** each edge $\mathscr{E}_{\mathscr{F}}$ of face $\mathscr{F}$ **do**
7 **if** no $\gamma_{\mathscr{E}\mathscr{F}}$ in $\mathscr{E}_{\mathscr{F}}$ **then continue** `/* skip if no OCCT curve is present */`
  `/* Coordinate error correction induced by the face edge */`
8 $\sum \Delta C\mathscr{F} \mathrel{+}= (L_1(C_{\mathscr{E}\mathscr{F}}, X) - \gamma_{\mathscr{E}\mathscr{F}}(L_1(P_{\mathscr{E}\mathscr{F}}, X))) \cdot (X \cdot (n_{\mathscr{F}} \times d_{\mathscr{E}}))$
  `/* Parameter error correction induced by the face edge */`
9 $\sum \Delta P\mathscr{F} \mathrel{+}= \left(L_1(P_{\mathscr{F}\cap\mathscr{E}}, X) - \gamma_{\xi\mathscr{E}\mathscr{F}}(L_1(P_{\mathscr{E}}, X))\right) \cdot (X \cdot (n_{\mathscr{F}} \times d_{\mathscr{E}}))$
  `/* Coordinate error correction induced by the face */`
10 $\sum \Delta C \mathrel{+}= (L_2(C_{\mathscr{F}}, X) - \xi_{\mathscr{F}}(L_2(P_{\mathscr{F}}, X) + \sum \Delta P\mathscr{F}) - \sum \Delta C\mathscr{F}) \cdot (X \cdot n_{\mathscr{F}})$
11 **for** each edge $\mathscr{E}$ **do**
12 **if** no $\gamma_{\mathscr{E}}$ in $\mathscr{E}$ **then continue** `/* skip if no OCCT curve is present */`
  `/* Coordinate error correction induced by the edge */`
13 $\sum \Delta C \mathrel{+}= (L_1(C_{\mathscr{E}}, X) - \gamma(L_1(P_{\mathscr{E}}, X))) \cdot (X \cdot n_{\mathscr{E}1}) \cdot (X \cdot n_{\mathscr{E}2})$
 `/* Apply all error corrections to the trilinear interpolation result */`
14 $Y \mathrel{+}= \sum \Delta C$

(green volume on the right in Fig. 3.7) induced by the edges of the face. These correspond to the term $\sum_{\mathscr{E}\mathscr{F}=0}^{3}(\Delta C_{\mathscr{E}\mathscr{F}}(X))$ from Eqn. 3.3.

We can subtract this later on from the scaled coordinate error correction of the face to get $\sum_{\mathscr{F}=0}^{5}\left(\Delta C_{\mathscr{F}}(X) - \sum_{\mathscr{E}\mathscr{F}=0}^{3}(\Delta C_{\mathscr{E}\mathscr{F}}(X))\right)$ from Eqn. 3.3 (orange volumes on the right of Fig. 3.7).

For the computation of $\sum \Delta C\mathscr{F}$, we use X to interpolate linearly ($L_1()$) between the coordinates of the nodes of the edge $C_{\mathscr{E}\mathscr{F}}$. From this, we subtract the the coordinates we obtain by evaluating $\gamma_{\mathscr{E}\mathscr{F}}$ with the interpolated parameters $P_{\mathscr{E}\mathscr{F}}$ of the nodes of the edge. Lastly, we scale everything with the component of X, which is orthogonal to the normal vector n of $\mathscr{F}$ and the direction d of $\mathscr{E}$ in the reference element. The scaling factor $(X \cdot (n_{\mathscr{F}} \times d_{\mathscr{E}}))$ is swapped to $(1 - X \cdot (n_{\mathscr{F}} \times d_{\mathscr{E}}))$ if $\mathscr{E}$ lies on the opposite site on $\mathscr{F}$.

In line 9, we do almost the same we did in the previous line, but this time we interpolate between the `OCCT` surface parameters $P_{\mathscr{F}\cap\mathscr{E}}$ of the two nodes, which the `OCCT` curve lies on. From this, we subtract the `OCCT` surface parameters we obtain from $\gamma_{\xi\mathscr{E}\mathscr{F}}$. γ_{ξ} converts a parameter on

an OCCT curve to the corresponding parameters on an OCCT surface. Into $\gamma_{\xi\mathscr{E}\mathscr{F}}$, we put the linear interpolation between the OCCT curve parameters $P_{\mathscr{E}}$. After that, we scale similar to line 8 and accumulate all scaled parameter error corrections for this face in $\sum \Delta P_{\mathscr{F}}$. We use these in the next line to mitigate the problem described in Fig. 3.5.

Next, we step out of the loop over each edge of the face and calculate the scaled coordinate error corrections for the OCCT surfaces (orange volumes on the right of Fig. 3.7) and accumulate them in the sum of all scaled coordinate error corrections $\sum \Delta C$. This is done similarly to the other calculations. But this time we evaluate $\xi_{\mathscr{F}}$ with the sum of the parameters we obtain by a bilinear interpolation $L_2()$ between the parameters of the nodes of the face and the sum of the scaled parameter error corrections $\sum \Delta P_{\mathscr{F}}$ of the edges. We subtract the from $\xi_{\mathscr{F}}$ obtained coordinates from the coordinates obtained by bilinear interpolation of the coordinates of the face nodes. Moreover, we subtract the sum of the scaled coordinate error corrections $\sum \Delta C_{\mathscr{F}}$ of the edges from it. After that, we scale the result with X and the normal vector n of $\mathscr{F}$.

After calculating $\sum_{\mathscr{F}=0}^{5} \left(\Delta C_{\mathscr{F}}(X) - \sum_{\mathscr{E}\mathscr{F}=0}^{3} (\Delta C_{\mathscr{E}\mathscr{F}}(X))\right)$, which is represented by the orange volumes on the right in Fig. 3.7, we have to calculate $\Delta C_{\mathscr{E}}$, which is represented by the green volume. We did this step already in line 8 to subtract it from the scaled coordinate error corrections from the OCCT surfaces. To add them back in we iterate over each edge and check if an OCCT curve is present. If that is the case, we do the same as in line 8 and add the value to the sum of all scaled coordinate error corrections $\sum \Delta C$.

Lastly, we add $\sum \Delta C$ to Y and Y now contains the output value of our cell volume description.

3.3 High-order curved mesh generation

After taking a detailed look at the principles of the *msh* import and the mapping algorithm, we can look at other implementations of high-order meshes. Due to the similarities of our curved AMR approach to classical high-order meshes, we will eventually face the same challenges. Hence we will take a look at the challenges of high-order mesh generation and at approaches to overcome them.

The generation and usage of high-order meshes have led to a significant decrease in run times, but with comparable numerical accuracy. This is achieved by combining the principles of the finite volume (FV) method with the cell polynomials of finite element (FE) method. This lead to the development of the discontinuous Galerkin (DG) method by Reed and Hill [35] and the further improvement by Cockburn and Shu [36, 37, 38, 39, 40], which can achieve a high numerical accuracy even with coarser meshes in comparison to the FV method, due to the polynomials inside the mesh cells.

To get the maximum accuracy of the DG method for a given mesh, the additional nodes of the high-order cells should curve with the geometry the domain describes. There are two methods to generate these curved high-order meshes. The first is a direct approach [41], where the mesh is generated directly as a high-order curved mesh.

The second is an a posteriori approach [1, 8, 42, 43, 44, 45, 46, 47]. First, a linear, first-order mesh is created. Then, the cells of the mesh get their additional nodes. In the cell, which has an edge or a face on a geometry, these additional nodes get placed with respect to the geometry.

Both approaches have their advantages and disadvantages, but here we want to focus on the second approach because it is more similar to ours. The biggest challenge for the construction of a high-order mesh from a linear mesh is called mesh tangling. It describes the phenomenon when valid linear cells become invalid due to the addition of curvature. Invalidation can be caused by near tangent edges (faces) or the intersection of edges (faces). This is shown in Fig. 3.8.

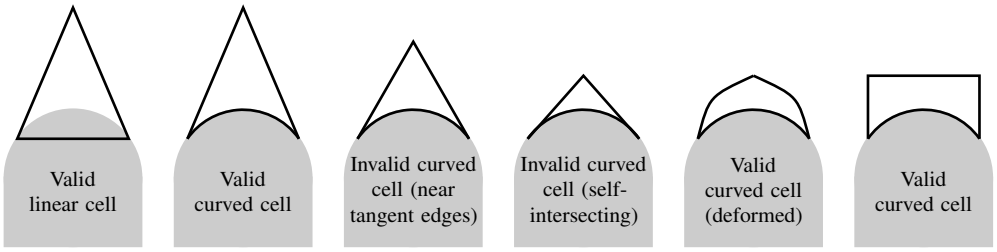


Figure 3.8: Shown are different cells, which lie on the same geometry (gray). The curved cell constructed from a linear cell is not always valid, but the invalid cells can be corrected via a deformation of the edges or faces, which are not linked to a geometry. Inspired by [1].

There are different approaches to untangle the mesh. Generally, they all move the curvature of the boundary cells into the domain. This means, that, if necessary, also the edges and faces, which do not lie on a boundary, are curved as well (see Fig. 3.9). Sometimes triangular or tetrahedral cells are converted to quadrilaterals or hexahedra, because they have a higher sum of interior angles and are therefore less prone to getting tangled. Some triangular or tetrahedral approaches even try to swap as many invalid edges and faces to a valid state as possible, to minimize the number of edges and faces, which have to be curved.

Since the self-intersection of mesh cells can also happen in our case, we have to take this into account. Due to the higher internal angles of hexahedral cells, we currently have a lower risk of near tangent edges and faces. Furthermore, we have to structurally mesh our geometries due to the limited three-dimensional hex-only meshing capabilities of `Gmsh` (more on that in Sec. 4.2). In our case, this leads to more control over the coarse mesh cells and helps us to eliminate the risk of generating self-intersecting cells.

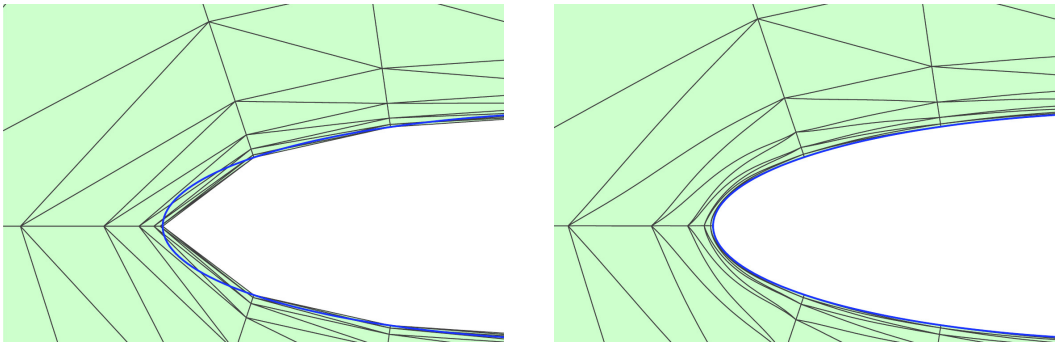


Figure 3.9: Untangling of a high-order curved mesh. The curvature of the boundary cells gets mapped into the volume, so that no overlapping of cells happens. By courtesy of Per-Olof Persson taken from [8].

For this evaluation, it is therefore not necessary to develop an algorithm to curve faces and edges, which do not contact the boundary of the geometries, but it has to be implemented in the future. This application of curvature can be a preprocessing step along with the coarse mesh import. In this step, the invalid edges and faces, which do not already have a curvature, get an analytically defined curvature, which is determined by the already curved edges and faces. This way the application of curvature has to be performed one time before the simulation and remains during each refinement or coarsening in the numerical application. Suggestions for the calculation of these curvatures can be found in [8, 42, 46, 47, 45].

4 Evaluation

In this chapter, we are going to select the application scenarios for the evaluation of curved hexahedral AMR and define the metrics we want to use for the evaluation. After that, we build the models for the chosen application scenarios. This includes the construction and meshing of our flow domain, the definition of our flow field $\vec{u}(\vec{x})$, and the setup of each simulation. Last, we will run and compare the simulations.

4.1 Application scenarios

We start with the definition of our application scenarios. From the fact, that only hexahedra are supported and from the chosen advection solver (see Sec. 2.3) as well as the chosen mesh generator (see Sec. 3.1), we can derive two requirements for our application scenarios. The first requirement is, that the domain has to be meshable with hexahedra only. `Gmsh` can generate hex-only meshes, but only in specific circumstances. We will address this during the construction and meshing of our domains in Sec. 4.2.

The second requirement is the flow field $\vec{u}(\vec{x})$. Although it is possible to set up a computational fluid dynamics (CFD) analysis for the given domain, we want to use analytical definitions for $\vec{u}(\vec{x})$. First of all, would it require additional resources to set up a CFD analysis, which would not benefit the informative value of this evaluation. Furthermore, would it limit the resolution of $\vec{u}(\vec{x})$ to discrete values. To make sure, that the resolution is fine enough even with high refinement levels of the advection mesh, we would be obliged to use an overly fine mesh throughout the whole CFD analysis.

One application scenario, which fulfills these requirements, is the flow around a sphere shell. The sphere shell is meshable hex-only in `Gmsh` and an analytical flow field without in- and outflow can be defined. More on this in Sec. 4.2 and Sec. 4.3. As a metric for the quality of the resolution of the geometry we propose the tracking of the volumes of the subdomains $\Omega_{1,\,t}$, $\Omega_{2,\,t}$ over time. Due to the flow field having no in- and outflow, there should be no flux across the geometry boundary and the volume of both subdomains should remain constant over time:

$$\begin{aligned} V(\Omega_{1,\,t}) &= V(\Omega_{1,\,t+\Delta t}), \\ V(\Omega_{2,\,t}) &= V(\Omega_{2,\,t+\Delta t}). \end{aligned} \tag{4.4}$$

To test if this conservation is preserved, we will put $\Omega_{1,\,t}$ near the boundary of the sphere and observe, if the volume changes. If the geometry resolution is poor, we expect some in- and outflow of the domain, because the flow is only parallel to the geometry, as shown in Fig. 4.1. We will test this for the convex, as well as the concave boundary of the sphere, to check if any differences become apparent.

Moreover, we would expect the shape of both subdomains to remain the same, because they rotate around the origin with a constant angular velocity. But the poor geometry resolution leads to an uneven flow distribution (see Fig. 4.1), which could affect the shape of the subdomains. This will be the second metric we check with this application scenario.

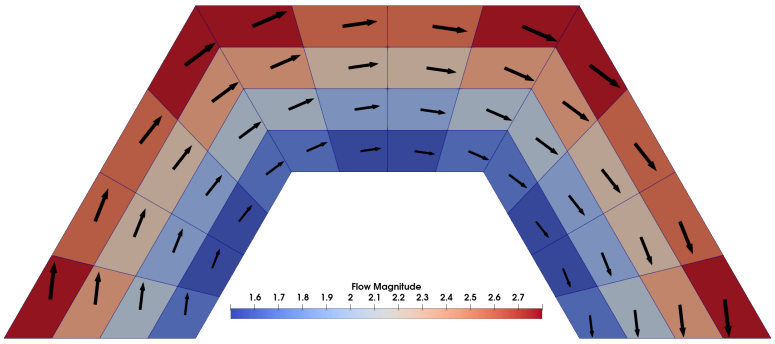


Figure 4.1: The analytically defined flow around a circle is applied to a circular mesh with poor geometry resolution. Due to the poor geometry resolution, we could get some in- and outflow of the domain. Moreover, the poor geometry resolution leads to an uneven distribution in the flow magnitude.

The second application scenario, we are going to use to evaluate curved hexahedral AMR, is the advection of $\Omega_{1,\,t}$ past a cylinder with a translational and angular velocity around its symmetry axis. A rectangular domain with a cylindrical hole is also meshable hex-only in `Gmsh` and an analytic flow field is also definable. In this scenario, we can check the same metrics as before.

4.2 Geometry and mesh

The construction and the meshing of the geometries are done in Gmsh because it can use the OCCT geometry kernel and is, therefore, able to export the geometry in the needed *brep* file format. Moreover, we can read in the parametric meshes in the *msh* file format from Gmsh and link them back to the original geometry provided by the *brep* file, as shown in Sec. 3.1.

Gmsh has many functions to build a hex-only mesh, but some of these features are unsuitable for our use case. The way our *msh* file input works is, that we read in our mesh nodes with the geometry information, like the geometry indices and we then retrieve the geometries from our OCCT *brep* file. This only works if the geometry indices of the *msh* and *brep* refer to the same geometries. This is only the case, if we load the *brep* file into Gmsh, mesh it, and then export the parametric mesh.

If we construct the geometry at the same time as we define the mesh, Gmsh changes the indices of the geometries, but the already defined nodes keep their geometry indices and the geometry then gets mismatched by our file reader. This means, that we are limited to the post-construction meshing algorithms Gmsh supplies.

Due to the absence of unstructured hex-only meshing algorithms in Gmsh, we have to use the structured algorithm based on transfinites. We can use this algorithm by specifying the number of points, which Gmsh then places on the specified OCCT curve. This is what Gmsh specifies as creating a transfinite curve. We have to transfinite all OCCT curves before going over to the next step.

After that, we can transfinite the OCCT surfaces. This is done automatically by Gmsh. We only have to specify four corner points of the OCCT surface (the OCCT surface itself can have more than four corners, this can be seen in Fig. 4.2). Gmsh then checks, if the specified amount of transfinite points of the opposite transfinite curves match and creates a transfinite surface. This transfinite surface can then be meshed with triangles or if we recombine it (that is how Gmsh names this process), we are also able to mesh it with quadrilaterals.

Last, we can automatically transfinite the volumes. Currently, it is not supported to transfinite volumes with more than six faces (Gmsh version 4.8.4). This means, that we are limited to constructing six-faced volumes only.

The first geometry, the sphere shell, hence consists of six equal sections. All of them have six faces and it is possible to use the transfinite algorithm for hexahedral meshing. Each section is then meshed with 2 by 2 by 2 cells, this is shown in Fig. 3.2a.

Gmsh supports a $1:8$ refinement of each hexahedral cell and with this refinement, we can create coarse meshes with different geometry resolutions. But during the first simulations, it became apparent, that the mesh obtained by $1:8$ refinement is of a lower quality than the mesh

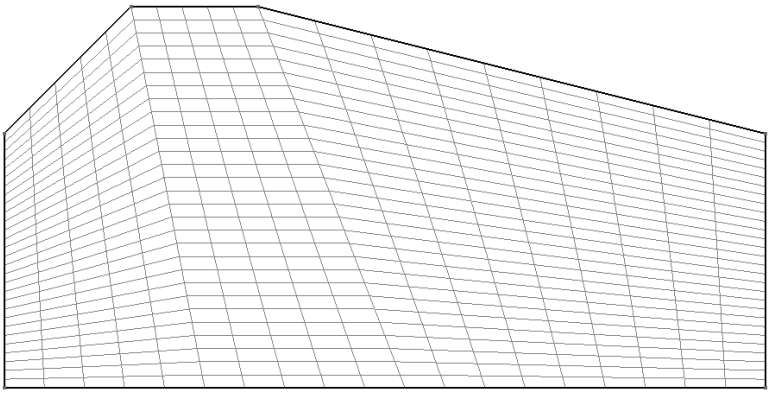


Figure 4.2: `Gmsh`'s transfinite algorithm is a structured meshing algorithm which can generate quad-only and hex-only meshes. It converts each surface into a quadrilateral and structurally meshes it using triangles or quads. Volumes can only be meshed if they have six faces. In this case we can see an `OCCT` face, which comprises six `OCCT` edges and is meshed with the transfinite algorithm.

obtained by re-meshing the whole domain with more cells. This led to numerical inaccuracies, which could be traced back to the mesh refinement of `Gmsh`. Our goal is to give a fair comparison and therefore we use the more practical scenario and re-mesh the domain to get finer coarse meshes. In the next finer mesh we hence use 4 by for by 4 cells per section, then 8 by 8 by 8, and so on.

The second geometry, the flow domain around a cylinder, is also divided into sections with six faces and is meshed just like the sphere shell. The geometry and coarse mesh are shown in Fig. 4.3.

4.3 Flow field

The flow of the sphere shell is rather simple, as we define it in polar coordinates. We place the center of the sphere shell at the origin and we want a steady flow rotating around the z-axis with an equal angular velocity ω at every point. For this, we use the function

$$u_{\mathrm{r}} = 0,$$
$$u_\theta = \frac{\omega}{2\pi} r,$$

with the radial coordinate r, angular coordinate θ, the velocity u_{r} in radial direction and u_θ being the velocity perpendicular to that.

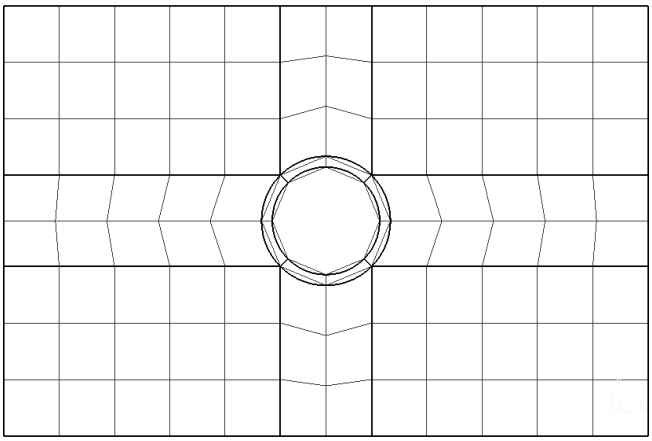


Figure 4.3: Shown is the geometry (thick) and mesh (thin) of the flow domain with a cylindrical cutout. Due to the requirement of structured hex-only meshing, the channel is divided into multiple sub-volumes.

Due to u_r always being zero, the flow is always parallel to the boundary of the sphere shell. This should result in no in- and outflow of both subsets, which is required by Eqn. 4.4.

The flow around a rotating cylinder can be modeled with a combination of three different two-dimensional solutions for Laplace's equation [48]. The solutions consist of the velocity potential ϕ_vp and the stream function ψ_sf, whereby the velocity field $\vec{u}$ can be obtained by

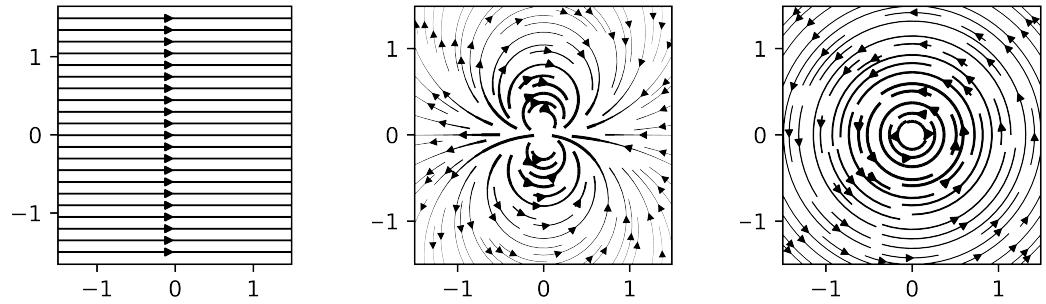
$$\begin{aligned} u_\mathrm{r} &= \frac{\partial \phi_\mathrm{vp}}{\partial r}, \\ u_\theta &= \frac{1}{r}\frac{\partial \phi_\mathrm{vp}}{\partial \theta}, \end{aligned} \tag{4.5}$$

for irrotational flows $\nabla \times \vec{u} = 0$. If the flow is source and sink free $\nabla \cdot \vec{u} = 0$, the velocity field can be obtained by

$$\begin{aligned} u_\mathrm{r} &= \frac{1}{r}\frac{\partial \psi_\mathrm{sf}}{\partial \theta}, \\ u_\theta &= \frac{\partial \psi_\mathrm{sf}}{\partial r}. \end{aligned} \tag{4.6}$$

Note, that the symbols ϕ and ψ are already used in this thesis and therefore we label them with vp for velocity potential and sf for stream function.

First, we start with the constant and inviscid potential flow past a circle with radius R. The flow is a combination of two basic flows. The first flow is a uniform flow in x-direction (see Fig. 4.4a) with the velocity potential $\phi_{\mathrm{vp\ uniform}} = ur\sin(\theta)$, the stream function $\psi_{\mathrm{sf\ uniform}} = ur\cos(\theta)$ and



(a) Uniform flow **(b)** Doublet flow **(c)** Vortex flow

Figure 4.4: Shown are three different two-dimensional flows, which satisfy Laplace's equation. The uniform flow has a velocity field which is constant everywhere. The doublet flow consists of a source (inlet) and a sink (outlet) infinitely close together. The vortex flow has a constant angular velocity around its center and has no radial velocity.

the velocity u. The second flow is a flow of a doublet (see Fig. 4.4b) with the velocity potential $\phi_{\text{vp doublet}} = -k\sin(\theta)/r$, the stream function $\psi_{\text{sf doublet}} = k\cos(\theta)/r$ and the strength k. The flow of a doublet is the flow induced by a sink and a source of equal strength infinitely close together.

We want the flow at the boundary of the circle $r = R$ to be parallel to said boundary $u_r(R,\ \theta) = 0$. Due to the combination of both flows being irrotational $\nabla \times \vec{u}_{\text{uniform + doublet}} = 0$ we can use Eqn. 4.5 and get

$$u_{\text{r uniform + doublet}}(R,\ \theta) = \frac{\partial \phi_{\text{vp uniform + doublet}}}{\partial r} = u\cos(\theta) - \frac{k\cos(\theta)}{r^2} = 0,$$
$$\Leftrightarrow k = ur^2 = uR^2.$$

Applying this to our velocity components $u_{\text{r uniform + doublet}}$ and $u_{\theta\ \text{uniform + doublet}}$ we get

$$\begin{aligned} u_{\text{r uniform + doublet}} &= \frac{\partial \phi_{\text{vp uniform + doublet}}}{\partial r} = u\left(1 - \frac{R^2}{r^2}\right)\cos(\theta), \\ u_{\theta\ \text{uniform + doublet}} &= \frac{1}{r}\frac{\partial \phi_{\text{vp uniform + doublet}}}{\partial \theta} = -u\left(1 + \frac{R^2}{r^2}\right)\sin(\theta). \end{aligned} \tag{4.7}$$

The resulting velocity field is shown in Fig. 4.5a. This is the analytical solution to the inviscid, incompressible, two-dimensional potential flow past a circle [48].

To make this flow field a little more complex, we add the flow of a vortex (Fig. 4.4c) to simulate the rotation of our circle. This is no longer the analytical solution to the potential flow past a

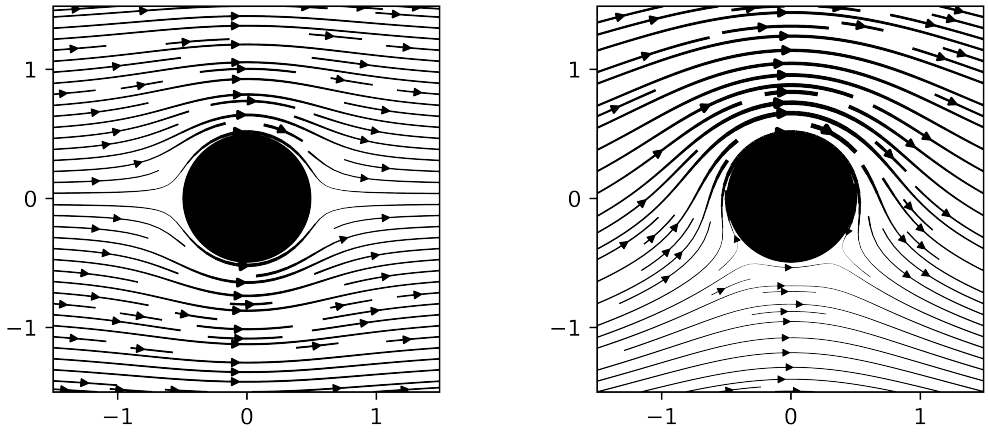
circle because it is not rotation free. Moreover, the rotation of a circle would not have any effect on the flow, because the potential flow is inviscid.

The vortex flow has the velocity potential $\phi_{\text{vp vortex}} = \frac{\omega\theta}{2\pi}$ and the stream function $\psi_{\text{sf vortex}} = -\frac{\omega\theta}{2\pi}\ln(r)$ with the angular velocity ω. Due to $\vec{u}_{\text{vortex}}$ being not rotation free $\nabla \times \vec{u} \neq 0$, we have to use Eqn. 4.6 to retrieve the velocity field and add it to $\vec{u}_{\text{uniform + doublet}}$. We get the resulting velocity field $\vec{u}_{\text{uniform + doublet + vortex}}$:

$$\begin{aligned} u_{\text{r uniform + doublet + vortex}} &= u\left(1-\frac{R^2}{r^2}\right)\cos(\theta), \\ u_{\theta\text{ uniform + doublet + vortex}} &= -u\left(1+\frac{R^2}{r^2}\right)\sin(\theta) - \frac{\omega}{2\pi}r. \end{aligned} \tag{4.8}$$

The velocity field is visualized in Fig. 4.5b.

Due to $u_{\text{r uniform + doublet + vortex}} = 0$ for $r = R$ the flow is always parallel to the boundary at the circle. If we let $\Omega_{1,\,t}$ only near the circle boundary and keep a distance from the remaining boundaries of the flow domain, Eqn. 4.4 holds true.



(a) Potential flow past a circle (Eqn. 4.7) **(b)** Flow past a rotating circle (Eqn. 4.8)

Figure 4.5: Shown are flows past a circle. The first flow is the potential flow past a circle. It is a combination of the uniform flow (Fig. 4.4a) and the doublet flow (Fig. 4.4b). The second flow is the same combination, but here we added a vortex flow (Fig. 4.4c) as well. Due to its rotation it is not longer a potential flow.

4.4 Simulation

After the setup of our evaluations, we are going to perform them. For the sphere shell evaluations, we decide to create four different cmeshes. The coarsest cmesh is the cmesh described in Sec. 4.2. We re-mesh the geometry with different resolutions to yield four different geometry representation accuracies. We measure the geometrical accuracy with the amount of cmesh cells the meshes use to approximate the circular cutout we get if we slice the meshes. The accuracies reach from 8 to 64 and the respective amount of cmesh cells in the whole mesh can be seen in Tab. 4.1 and Tab. 4.2.

Our main goal is to compare the results of the curved geometry mapping to the linear geometry mapping. Generally, we expect no difference in accuracy between curved geometry simulations with different cmesh resolutions, because the geometrical accuracy of the curved geometry mapping will always be as accurate as it can be.

But in [9] we have seen, that the run time of two simulations with different cmesh resolutions varies, due to the varying ratio of curved and linear cells (the curved cells are slower to compute than the linear ones). The finer a cmesh is, the more cells are completely inside the volume and are therefore linear. To verify this expectation, we will do two runs with the curved geometry mapping, one with the cmesh of resolution 8 and one with the cmesh resolution 16.

The runs with the linear geometry mapping will be done with each cmesh resolution. This should give us enough data to compare it to the curved geometry mapping. Note, that we reduce the refinement level if we use a finer cmesh. This way the smallest elements have the same size, we have roughly the same amount of elements in each simulation and we have the same amount of time steps for a given simulation time and CFL number.

Since the output of the mesh data for visual analysis can affect the run times, we will run each simulation twice. One simulation without VTK file output to evaluate the run time and accuracy and one simulation for the VTK file output itself.

The evaluation runs for the sphere shell evaluations are conducted using the tag `masters_thesis_elsweijer`[1] of `t8code` and one computing node with two Intel Xeon Gold 6130 and 192 GB of RAM. The command to run the sphere shell simulation with the concentration at the convex boundary is as follows:

```
mpirun -n 32 ./t8_advection -u4 -l0 -r<refinement level> \
-f<path/to/msh/file> -d3 -C0.5 -b4 -X1.1 -Y0 -Z0 -R0.3 -T0.25 \
(-o for no file output) (-O for curved geometry)
```

And the command for the concentration at the concave boundary:

[1] `https://github.com/sandro-elsweijer/t8code/releases/tag/masters_thesis_elsweijer`

```
mpirun -n 32 ./t8_advection -u4 -l0 -r<refinement level> \
-f<path/to/msh/file> -d3 -C0.5 -b4 -X1.9 -Y0 -Z0 -R0.3 -T0.25 \
(-o for no file output) (-O for curved geometry)
```

The simulations with the rotating cylinder are conducted similarly. We use the mesh introduced in Sec. 4.2 and re-mesh it with different resolutions to get different geometrical accuracies. But this time we run the simulations also with different refinement levels. This way, we can comprehend, if a higher geometric accuracy or a higher mesh resolution has more influence on the accuracy of the results.

Note, that we run out of memory with the aforementioned computing node. This happens, because the partitioning of the OCCT geometries is not yet implemented and we are therefore not able to partition the cmesh. This leads to one instance of the whole cmesh and geometry being reserved in the memory for each MPI rank. The partitioning of the cmesh would partition this data for each MPI rank so that each rank has only the information it needs.

If we partition the cmesh anyway, we get memory errors. Because of cross-references and reference counters in OCCT, it is not trivial to partition the OCCT geometries and this is why the partitioning of the geometry-linked cmesh is out of the scope of this thesis.

The cmesh partitioning would also enable us to take a look at the memory usage and compare it to the usage of the linear mapping algorithm. But with the current state of the algorithm, the results would become invalid as soon as the curved cmesh partitioning is implemented.

The not yet implemented cmesh partitioning for curved meshes is why we switch the hardware configuration and use one computing node with two AMD EPYC 7702 and 1024 GB of RAM. Despite the higher core count of this computing node we still use 32 MPI ranks to save memory. We use the following command and parameters to start the simulation:

```
mpirun -n 32 ./t8_advection -u7 -l0 -r<refinement level> \
-f<path/to/msh/file> -d3 -C0.5 -b4 -X-0.7 -Y0 -Z0 -R0.3 -T1 \
(-o for no file output) (-O for curved geometry)
```

4.5 Results

After performing the simulations we take a look at the results. We apply our previously defined metrics and look if any differences between the results of the two application scenarios arise.

Results of the sphere shell evaluations runs

In Tab. 4.1 we can see the result of the sphere shell simulations, where the concentration is advected along the inner, convex boundary. In Fig. 4.6, we can see a visualization of the run with

curved geometry and a cmesh resolution of 16 as a reference. A visualization of the last time step of each simulation can be seen in Fig. B.1.

Table 4.1: Shown are the results of the sphere shell evaluation runs, where the concentration is placed at the inner, convex boundary of the sphere shell. The circle resolution describes how many cmesh cells are used to approximate the circular section. C is the amount of cmesh cells, E the mean amount of elements per time step and r the max refinement level. Artifacts shows the presence of numerical anomalies.

geometry mapping	circle res	C	E	r	volume loss	arti-facts	run time
curved	8	48	2.71e5	6	16.1 %	yes	2.447e3 s
curved	16	384	2.46e5	5	15.9 %	no	1.042e3 s
linear	8	48	2.89e5	6	14.5 %	yes	0.097e3 s
linear	16	384	2.48e5	5	15.5 %	no	0.084e3 s
linear	32	3072	2.39e5	4	14.7 %	no	0.077e3 s
linear	64	24576	2.46e5	3	14.4 %	no	0.094e3 s

As already mentioned, we ran two curved and four linear simulations. We expect two curved simulations to have the same accuracy, but different run times. But this is not what we see. If we take a look at Fig. B.1a, we can see, that in the curved simulation, with a resolution of 8, some artifacts occur, which seem to be induced by the poor resolution of the cmesh. The curved geometry only affects the boundaries of the cmesh cells and the space between the inner and outer boundary, where no geometry is present, is unaffected. This results in sudden changes in the orientation of the elements and the solver are not able to resolve these changes accurately.

Due to the higher cmesh cell count in the cmesh with a resolution of 16, these artifacts do not occur. This shows us, that the curved geometry description of the cells is not able to correct a poor cmesh, it is only able to correct inaccurate geometry representations. The cmesh still has to meet certain quality standards.

Although the volume losses are, unlike expected, not the same, the run time of the mesh with the higher cmesh resolution is reduced massively. This is because of the higher ratio of linear cells in the cmesh. This is the same result as we have seen in [9].

If we move on to the comparison to the linear simulations, we can see, that the volume losses do not vary much. Over all simulations (curved and linear), we have a variation of 1.7 percentage points. If we exclude the linear, resolution 8 run due to its artifacts, we can see a downwards trend in volume loss with higher resolutions. But if we compare the volume losses of the linear runs to the curved runs, we see, that the linear runs have less volume loss in general. This contradicts our expectations and we will have a look at this in the next evaluation scenarios as well.

We now move on to the comparison of the results of the runs, where the concentration moves along the outer, concave border of the domain. The results are shown in Tab. 4.2 and an exemplary visualization is shown in Fig. 4.7. A visualization of the last time step of each simulation can be seen in Fig. B.2.

We again start with the curved runs. In this case, the run with the lower cmesh resolution has less volume loss, contrary to the last scenario and there are no artifacts in the run with the lower

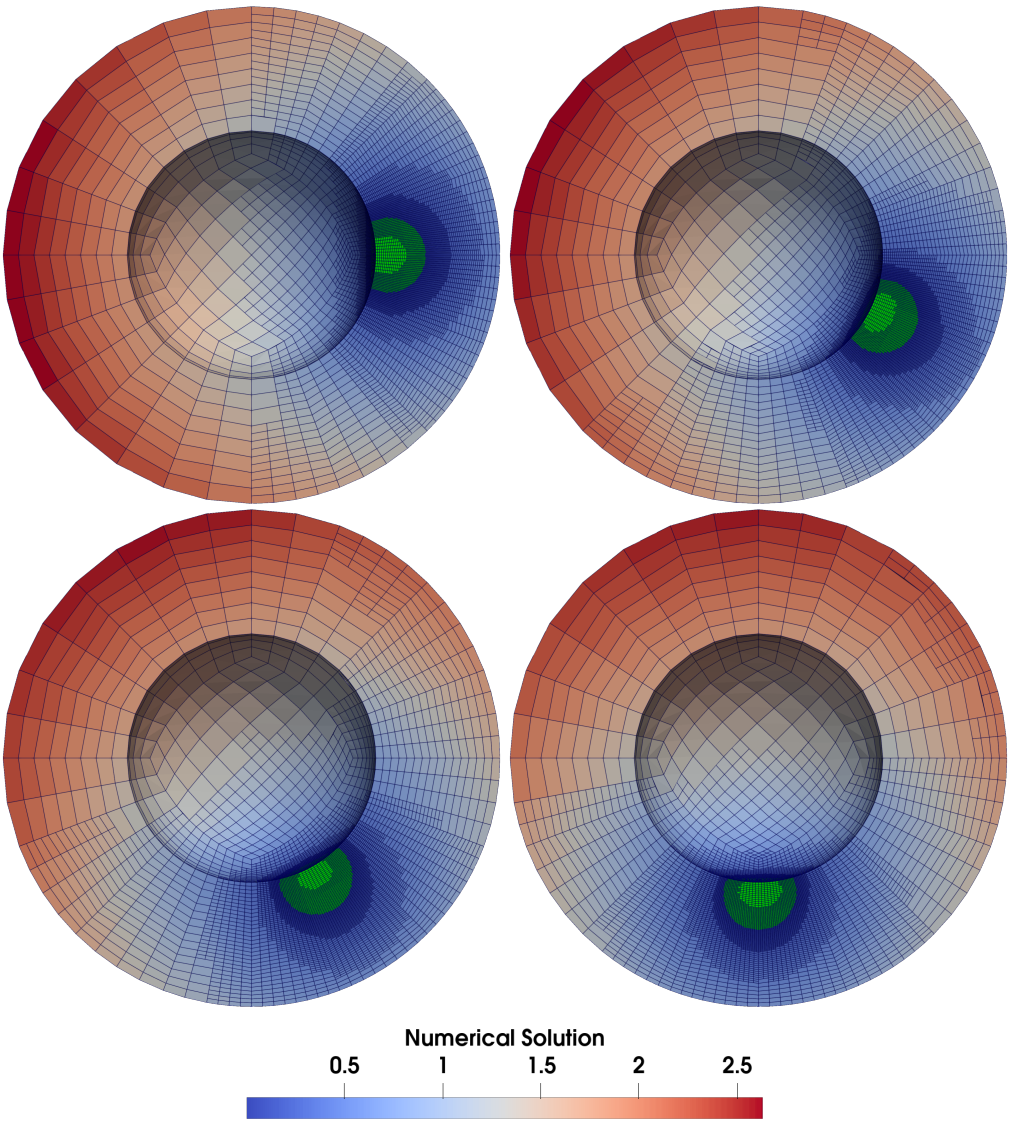


Figure 4.6: Shown is a visualization of the curved sphere shell simulation with a resolution of 16 and where the concentration moves along the inner, convex boundary.

Table 4.2: Shown are the results of the sphere shell evaluation runs, there the concentration is placed at the outer, concave boundary of the sphere shell. The circle resolution describes how many cmesh cells are used to approximate the circular section. C is the amount of cmesh cells, E the mean amount of elements per time step and r the max refinement level. Artifacts shows the presence of numerical anomalies.

geometry mapping	circle res	C	E	r	volume loss	arti-facts	run time
curved	8	48	1.02e5	6	31.6 %	no	0.937e3 s
curved	16	384	1.11e5	5	32.9 %	no	0.367e3 s
linear	8	48	0.87e5	6	33.4 %	yes	0.040e3 s
linear	16	384	1.10e5	5	32.7 %	no	0.048e3 s
linear	32	3072	1.17e5	4	33.5 %	no	0.049e3 s
linear	64	24576	1.34e5	3	33.9 %	no	0.054e3 s

cmesh resolution. But just like in the scenario before, the difference in the volume loss is not that significant and the run time with the higher cmesh resolution is reduced.

We now move on to the results of the linear runs. We can see, just like in the convex scenario, that in the run with the linear geometry and the cmesh resolution of 8, artifacts occur. Furthermore, we can see an upwards trend in the volume loss with rising cmesh resolution. This is again contrary to the results of the last scenario.

In the convex scenario, we had an increase in volume loss with the curved geometry mapping in comparison to the linear geometry mapping. Here, with the concave boundary, we have the opposite effect. An explanation could be, that the misrepresented boundary results in a gain in volume at the concave boundary and loss of volume at the convex boundary. This results in these opposite effects and suggests, that the curved geometry mapping produces results without volume gain or loss generated by the misrepresentation of the geometry. But with the given solver and the volume loss it generates, this theory cannot be proven.

The second metric, we defined for this scenario, is the deformation of the shape of the advected concentration. If we look at Fig. B.1 and Fig. B.2, we cannot determine any significant differences in the shape of the advected concentrations, apart from the obvious differences, where artifacts occur. Maybe a longer simulated time span would reveal some differences, but this is not possible because of the high volume loss of the advection solver. We get 100 % volume loss during one full rotation around the sphere shell.

Last, we take a look at the run times. It is visible, that the curved geometry runs are much slower. The curved runs with circle resolution 8 are about 25 times slower than their linear pendants and the curved runs with the circle resolution of 16 are around 8 to 12 times slower. The

curved runs even get outperformed in comparison to the much finer cmesh with a resolution of 64.

In conclusion, the sphere shell evaluation scenario may show a benefit of the curved hexahedral AMR over the linear AMR in the form of not gaining or losing volume through the boundary. But this is only a theory. The uneven distribution of the flow magnitude shown in Fig. 4.1 does not lead to a significant deformation of the shape of the advected concentration. With the inconclusive

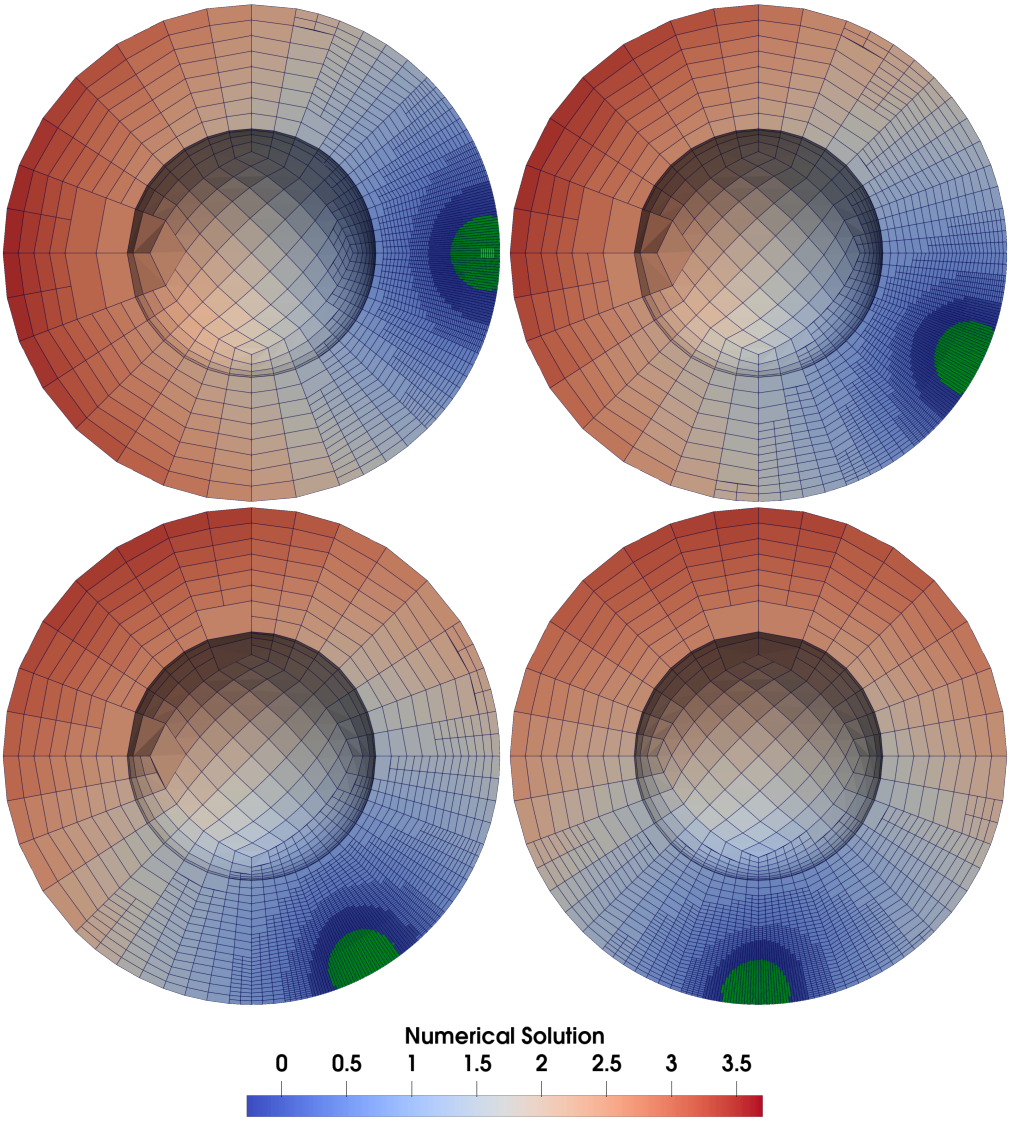


Figure 4.7: Shown is a visualization of the curved sphere shell simulation with a resolution of 16 and where the concentration moves along the outer, concave boundary.

improvements in accuracy and the massively increased run times, it may not be advisable to use the curved domain AMR in this application scenario.

Results of the rotating cylinder evaluations runs

We now move on to the results of the rotating cylinder scenario. We had to re-run these simulations because we initially set the magnitude of the analytic flow field inside the cylinder to zero. This resulted in elements with a flow magnitude of zero in the runs with the linear cell geometry. This was because they were positioned inside the cylinder, due to the misrepresentations of the geometry. This led to a divergence of the numerical results (see Fig. 4.8) and the runs were not comparable. This was no problem for the runs with the curved geometry, because of the exact placement of the mesh elements at the boundary of the cylinder.

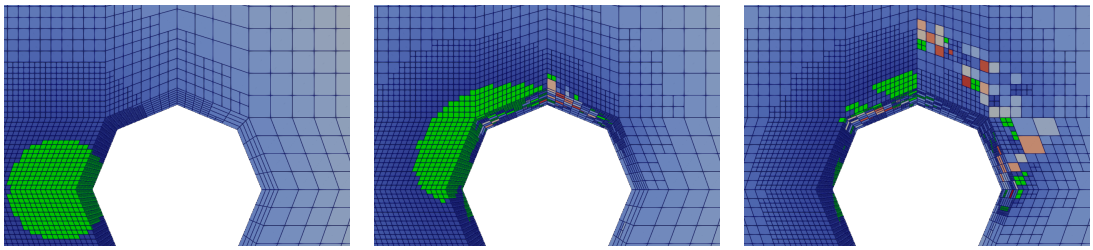


Figure 4.8: The simulations with a linear geometry and a circle resolution of 8 and 16 diverge when the magnitude of the flow field is set to zero inside the geometric cylinder. The curved simulations had no such problems.

We can see the results of the re-run with the continuous velocity field in Tab. 4.3 and we can see a visualization of the results in Fig. 4.9. The results of the last time step are depicted in Fig. B.3. In this scenario, we ran three different simulation sets. In the simulations of the same set, the smallest elements have the same size and therefore we have an approximately similar amount of time steps in each simulation.

We first look at the linear runs. Here we can see, that the number of elements rises with a higher resolution of the cmesh. Furthermore, we can see, that the linear simulations with lower circle resolutions produce some artifacts, which are the first sign of inaccurate results. The better the cmesh resolution gets, the fewer artifacts we get. In the last two simulation sets only the simulations with a circle resolution of 64 produce results without artifacts.

This is a contrast to the sphere shell evaluations. Here we only had artifacts with a circle resolution of 8. This means, that the application scenario has an influence on how important the resolution of the geometry is.

Furthermore, we can see, that the circle resolution, as well as the refinement level, have an influence on the volume loss. But we have to be careful with these conclusions because most of

Table 4.3: Shown are the results of the rotating cylinder evaluation runs. Just like before, the circle resolution shows how many cmesh cells are used to approximate the circular section. C is the amount of cmesh cells, E the mean amount of elements per time step and r the max refinement level. Artifacts shows the presence of numerical anomalies.

geometry mapping	circle res	C	E	r	volume loss	artifacts	run time
curved	8	500	0.103e6	4	46.6 %	no	0.285e3 s
linear	8	500	0.102e6	4	63.7 %	yes	0.036e3 s
linear	16	4000	0.102e6	3	48.3 %	yes	0.028e3 s
linear	32	32000	0.115e6	2	48.3 %	no	0.039e3 s
linear	64	256000	0.314e6	2	46.8 %	no	0.064e3 s
curved	8	500	0.294e6	5	27.7 %	no	1.169e3 s
linear	8	500	0.237e6	5	−2.5 %	yes	0.179e3 s
linear	16	4000	0.284e6	4	30.9 %	yes	0.163e3 s
linear	32	32000	0.304e6	3	28.2 %	yes	0.175e3 s
linear	64	256000	0.503e6	3	28.3 %	no	0.308e3 s
curved	8	500	0.994e6	6	19.4 %	no	6.552e3 s
linear	8	500	0.813e6	6	13.5 %	yes	1.168e3 s
linear	16	4000	0.959e6	5	21.2 %	yes	0.987e3 s
linear	32	32000	0.986e6	4	19.5 %	yes	0.972e3 s
linear	64	256000	1.195e6	4	19.4 %	no	1.412e3 s

the simulations produce artifacts and this can have an influence on the volume loss. In the case of the linear simulation with a circle resolution of 8 and a refinement level of 5, we even gain volume. But in general, we can say, that a higher refinement level in combination with a high circle resolution leads to more accurate results.

If we compare these results with the curved geometry evaluations we can see, that the curved evaluations have less or at least equal volume loss than every other simulation in their set. Moreover, they never produce any artifacts.

From the results, we can assume, that there are at least two factors influencing the amount of volume loss. One is the refinement level, which influences the size of the elements and the number of time steps. The other influence is the accuracy of the geometry. We cannot get to zero volume loss without a perfect geometry representation of the mesh (this is what we do with the curved cell geometry) and we also cannot get to zero percent volume loss without infinitesimal small elements and therefore infinitesimal small time steps. With the curved geometry, we can at least influence and max out one of these two requirements. But we still get an error of the other requirement.

We now move on to the shape our advected concentration has at the end of each simulation. In Fig. B.3, we can see a visualization of the last time step of each simulation from the first set. We stick to the first simulation set because the shape of the concentration does not change visibly with a higher refinement level.

First, we can observe the artifacts produced in the linear simulations with a circle resolution of 8 and 16. Second, we can see, that the concentration in the curved geometry sticks less to

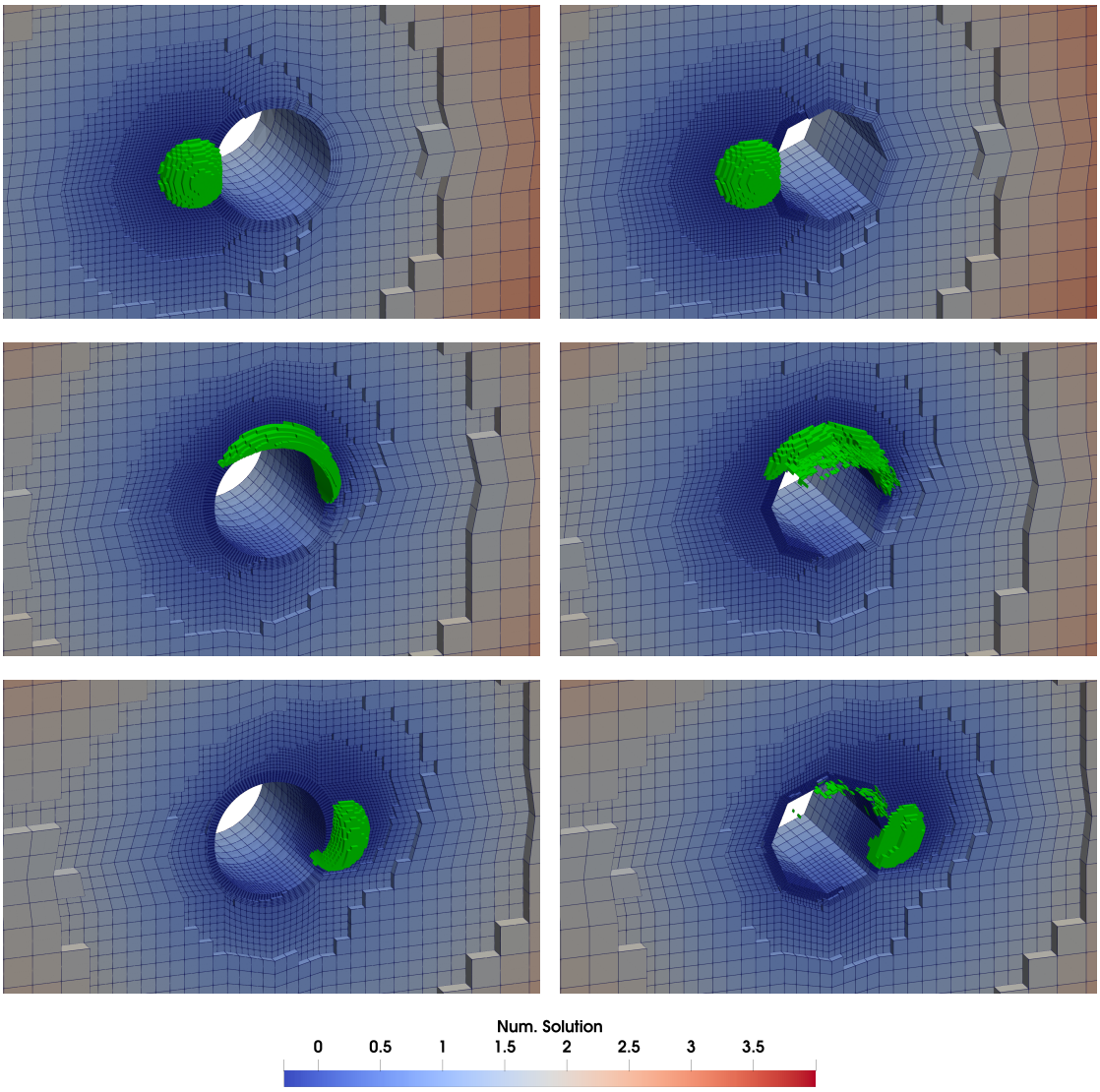


Figure 4.9: Shown is a visualization of the rotating cylinder evaluation. On the left we can see three time steps of the curved geometry simulation with a circle resolution and a refinement level of 4. On the right we can see the linear pendant. It is clearly visible, that the curved geometry evaluation is much more accurate.

the boundary of the cylinder. Only five elements of the concentration still touch the boundary of the cylinder in the section of the curved simulation (Fig. B.3e). In contrast, 7 elements stick to the boundary in the simulation with a resolution of 64 and 8 elements in the simulation with a resolution of 32. So even if the volume loss is nearly the same, the shape of the advected concentration is still different.

Last, we look at the run times of the simulations. We can see, that the run times with the linear geometry mapping do not always increase with the number of mean elements per time step. The anomalies are the linear simulations with a coarser cmesh and can be traced back to the uneven distribution of the concentration due to the artifacts. It leads to more `balance` rounds and also more time steps. But in general, we can say that a higher cmesh resolution leads to more elements and therefore longer run times.

Due to the linear simulations with a circle resolution of 64 being the only simulations with comparable accuracy, we can only compare their run times with the run times of the curved simulations. We can see, that the linear simulations are around 3.8 to 4.6 times faster than the respective curved simulations. In comparison to the results of the sphere shell evaluation, this is much more comparable. But this time the curved simulations are still more accurate, than the linear simulations with higher cmesh resolutions.

In conclusion, we can see a clear difference between the sphere shell and the rotating cylinder evaluation. While the curved geometry mapping had an inconclusive influence in the sphere shell simulations and increased the run time by up to a factor of 25, it had a measurable influence in the rotating cylinder example. It was even more accurate than linear simulation which had 512 times as many cmesh cells while being 3.8 to 4.6 times slower.

A short look at the run times per process

As we could see, the run times with the curved AMR are significantly longer than with the linear AMR. This can be partly attributed to the more complex cell geometry computation, but not all of it. This is why we want to take a short look at the run times per process.

We recorded the max and min run times of different algorithms per MPI rank. We monitored the main two algorithms, the solver and the AMR algorithms, which consist of the `adapt`, `balance`, `partition`, and `ghost` algorithm. Due to the AMR algorithms' need to wait for each other for the data exchange, we have to give the quotients of the AMR algorithms per time step. The solver quotient is given for the whole simulation.

If we look at the quotients of these max and min run times in Tab. 4.4, we can identify another reason for the long run times of the curved geometry mapping.

As we can see, the quotients of the max and min AMR run times in the curved simulations are up to 10 times higher than in the linear simulations and the same applies to the solver run time

Table 4.4: Shown are the quotients of the max and min run times of different algorithms during the curved and linear rotating cylinder runs with a circle resolution of 8.

geometry mapping	r	max AMR run time per time step / min AMR run time per time step	max total solver run time / min total solver run time
curved	4	1.291e4	2.640
linear	4	0.128e4	1.167
curved	5	0.765e4	2.091
linear	5	0.202e4	1.191
curved	6	0.656e4	1.923
linear	6	0.266e4	1.191

quotients, which are up to 2.3 times higher. This indicates a less optimal load distribution across the processes for the curved runs. This means, that processes are being earlier ready with their calculations than other processes. But due to the processes' need to communicate in each time step, this leads to processes waiting for each other and having downtime, which has an increase in run times as result.

This is probably caused by the partitioning of the elements across the processes. Currently, the elements are distributed evenly across the processes, so that one process can only have a maximum of one element more or less than another process. But this partitioning does not consider how much computational effort is necessary to compute the geometry of that element. This can lead to one process having only elements of a cell with multiple linked edges and faces and another process having only elements of a linear cell.

5 Conclusion

After performing the evaluation runs with our two application scenarios we can answer our research question:

How does the performance and accuracy of the hexahedral curved domain AMR algorithm compare to linear AMR when solving the advection equation with the linear FV method?

As we have seen in the previous chapter, the evaluations of the application scenarios had vastly different results. While we were not able to fully attribute the differences in terms of accuracy in the sphere shell scenario, we found significant differences in the rotating cylinder scenario. This means we have to answer the research question as differentiated as the results are.

The profit of curved hexahedral AMR hugely depends on the application scenario and has to be assessed for each scenario separately. Curved hexahedral AMR can have an influence in boundary-critical scenarios, like the scenarios we tested here. If the boundary is not that relevant or is not even curved, curved hexahedral AMR will produce no significant differences to linear AMR.

But in general, we cannot deduct specific guidelines for its usage. We have proven the existence of a profit in using a geometrically more accurate mesh and the possible profits from curved, high-order elements remain untested. But in some cases, this geometrical accuracy can also be achieved by a finer cmesh resolution, like in our rotating cylinder scenario.

But this stands in contrast with one of the top priorities of tree-based AMR: Memory efficiency. When dealing with large meshes it is often not possible to use cmeshes with a high geometry resolution and this is, where curved hexahedral AMR could reach its true potential. It is built on the principles of tree-based AMR and can use its memory efficiency.

But it is to say, that the not yet working cmesh partitioning leads to huge memory inefficiencies and the performance of the current implementation is far too slow to be practical under the tested circumstances. The run times of the rotating cylinder scenario with the curved geometry mapping were between 3.8 to 4.6 times slower than the run times of the linear run with the highest geometry resolution and comparable accuracy. But the implementation of the curved geometry mapping is still designed as a proof of concept; there was no focus on execution speed during the implementation. This means, that there are multiple places, where we can optimize the execution speed.

The first possible optimization is the geometry evaluation itself. Currently, we repeat the process of retrieving OCCT curve parameters, interpolating between them, and using them to evaluate an OCCT curve for each linked edge and each neighboring linked face. This means that it is possible to do this process 36 times, even if our cell has only 12 edges that can be linked.

Another possible optimization is the partitioning of the elements across processes, as shown in the previous chapter. Currently, the elements are distributed evenly across the processes. But with the curved geometry evaluation, the computational load of two distinctive elements can be vastly different. Therefore, a weighted distribution of elements across the processes would be more optimal. The weight of an element could then be influenced by the number of geometries linked to the ancestral cell of that element.

With these and possibly some more optimizations, this technique can become essential in the toolbox for large-scale numerical simulations and can even find an industry-relevant application. But until then, this technique remains a work in progress and can be improved by further works presented in the next chapter.

6 Outlook

After the evaluation of curved hexahedral AMR and the proof, that it has a measurable effect on the accuracy of certain scenarios, we are convinced to continue our work and improve this approach. Hence we present some subsequent works, which can advance this approach. We can distinguish between optimizations of the curved AMR in `t8code` itself and works, which serve a more broadened spectrum outside of `t8code`.

Improvements of curved AMR in `t8code`

Some of these works were already mentioned in the previous chapter: The cmesh partitioning, the mapping speed, and load balancing. These three not yet optimal algorithms are the main obstacles, which have to be addressed to make the new approach practical.

The non `OCCT` specific data types, like the node parameters and the information, which `OCCT` geometry is linked to which part of a cell, are already saved along with the cells. This means, that the cmesh partitioning algorithm can already partition this data. But due to the many dependencies between the `OCCT` datatypes themselves, it was not yet possible, to save them with the cells. To implement this, a deeper knowledge of the data structures and inner workings of `OCCT` is needed.

In contrast, the increase in the performance of the mapping speed should be fairly easy to achieve. It is a matter of performance engineering and the restructuring of code, so that operations, which are now performed multiple times, only have to be performed once. This caching of interim results would result in higher memory usage, but only during the geometry evaluation itself. After the geometry evaluation, all interim results would be freed. Therefore no data has to be stored for a long period of time and this caching could be made optional, for the usage of memory critical applications.

The last, but also critical improvement, the load balancing across processes, can be optimized using a weight system for the partitioning of the elements. With this system, an element, which is influenced by a lot of `OCCT` geometries, can get a higher weight than an element, which is not influenced by any `OCCT` geometry. But this weight is highly dependent on the numerical application. Not only the geometry evaluation run time has an influence on the overall calculation

run time. It is also influenced by the run time of the solver itself, the number of geometry evaluations the solver needs, the memory latency and speed of the hardware, and so on.

Hence, defining weights for an element is not trivial and involves a lot of testing and even after testing, the weights are only valid for this specific use case. This is why we propose an adaptive weighting algorithm, which determines a weighting function in the first time step and then applies this function in the next time steps. This way only the first time step has an imperfect load balancing, but the subsequent time step would have the optimal load balancing for this use case. And additionally, the weighting function has to be set up only once in the whole computation.

Moving on, we take a look at more optional works to improve curved AMR. The first work is the extension of the geometry mapping to the other supported cell geometries of `t8code`; lines, triangles, quadrilaterals, tetrahedra, prisms, and pyramids. To do this, we do not have to start over, many of the approaches used in the curved hexahedral mapping can also be used in the other element shapes. For example, the concept of the coordinate and parameter error corrections defined in Sec. 3.2, the parametric *msh* import and the storage of the geometry information can be reused. Mainly the actual mapping of the coordinate error corrections inside the volume of the cell has to be developed.

The last improvement of curved AMR we want to propose here, is the in Sec. 3.3 mentioned implementation of a mesh untangling algorithm. At best, this algorithm is also able to quantify the mesh quality, another relevant metric for accurate simulations. The mesh untangling algorithm will be even more necessary with the introduction of triangular and tetrahedral elements, due to their lower sum of interior angles and their capability to form unstructured meshes. With these elements, an intersection between element edges (faces) or near tangent edges (faces) becomes much more common. And when constructing an unstructured mesh, less attention is paid to the shape and position of the mesh elements themselves.

More general concepts for future works

We can also take a broader look at this topic and even depict some works in combination with third-party software. The first work is the linkage of the `t8code` cell geometry and the `t8dg` solver. The necessary steps for this linkage were the already mentioned: The implementation of the Jacobian for the cell geometry and the orientation of the cells to align the polynomials on the element edges and faces. With this coupling, we will be able to expand the evaluation we did in this work to high-order meshes and we can measure if curved AMR leads to higher performance and if so, in which application scenarios.

Last, we want to look at is a whole framework or even suite, which combines a complete simulation chain in a robust and automated manner. Starting with the creation of geometries,

which is followed by the meshing process and ending with the actual simulation with mesh management and a high-order solver. Every chain link is already developed in its own way and efforts to link these links together have already started.

Due to the usage of `OCCT` for the geometry handling, it would be advantageous, if the other links, which use geometries, at least support `OCCT` as well. The creation of the geometries can therefore be handled by `TiGL` [49], a computational geometry library for parametric aircraft design which also uses the `OCCT` kernel and the `CPACS` [50] format for generating geometries.

A look at the automated meshing of these geometries was already taken by Paul Putin in [51]. Here, `Gmsh` was used to mesh geometries. The last link in the chain, besides `t8code`, are the high-order solvers implemented in `Trixi` [52, 53]. The linkage of these solvers is already in planning and will be performed in the Ph.D. thesis of David Knapp.

It is safe to say, that this evaluation is only the beginning of a series of subsequent works to advance this topic and may end in a whole software package for the parametric design of aircraft.

Bibliography

[1] Gianmarco Mengaldo et al. “Industry-Relevant Implicit Large-Eddy Simulation of a High-Performance Road Car via Spectral/hp Element Methods”. In: *CoRR* abs/2009.10178 (2020). DOI: 10.48550/arXiv.2009.10178.

[2] Luca Silvestri. “CFD modeling in Industry 4.0: New perspectives for smart factories”. In: *Procedia Computer Science*. Proceedings of the 2nd International Conference on Industry 4.0 and Smart Manufacturing (ISM 2020) 180 (Jan. 2021), pp. 381–387. DOI: 10.1016/j.procs.2021.01.359.

[3] P. Jöckel et al. “Development cycle 2 of the Modular Earth Submodel System (MESSy2)”. In: *Geoscientific Model Development* 3.2 (Dec. 2010), pp. 717–752. DOI: 10.5194/gmd-3-717-2010.

[4] Simon Rosanka et al. *Organic pollutants from tropical peatland fires: regional influences and its impact on lower stratospheric ozone*. Tech. rep. FZJ-2021-04142. EGU, 2020. DOI: 10.5194/acp-2020-1130.

[5] Raphael Egan et al. “Direct numerical simulation of incompressible flows on parallel Octree grids”. In: *Journal of Computational Physics* 428 (Mar. 2021). DOI: 10.1016/j.jcp.2020.110084.

[6] 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. DOI: 10.1109/MCSE.2014.75.

[7] Z.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. DOI: 10.1002/fld.3767.

[8] Per-Olof Persson and Jaime Peraire. “Curved Mesh Generation and Mesh Refinement using Lagrangian Solid Mechanics”. In: *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*. Aerospace Sciences Meetings. American Institute of Aeronautics and Astronautics, Jan. 2009. DOI: 10.2514/6.2009-949.

[9] 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).

[10] Sandro Elsweijer et al. "Constructing a Volume Geometry Map for Hexahedra with Curved Boundary Geometries". In: *SIAM International Meshing Roundtable Workshop 2022*. Accepted for publication. Feb. 2022. URL: https://elib.dlr.de/185781/ (visited on 03/25/2022).

[11] 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. DOI: 10.1137/100791634.

[12] Johannes Holke et al. *t8code: Parallel AMR on hybrid non-conforming meshes - Curved Geometry Fork*. 2022. URL: https://github.com/sandro-elsweijer/t8code/tree/feature-brep_geometry_2 (visited on 07/19/2021).

[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] Carsten Burstedde and Johannes Holke. "A tetrahedral space-filling curve for nonconforming adaptive meshes". In: *SIAM Journal on Scientific Computing* 38 (2016), pp. C471–C503. DOI: 10.1137/15M1040049.

[15] Lukas Dreyer and Johannes Holke. *t8dg*. 2022. URL: https://github.com/lukasdreyer/t8dg (visited on 01/24/2022).

[16] Lukas Dreyer. "The local discontinuous galerkin method for the advection-diffusion equation on adaptive meshes". MA thesis. Rheinische Friedrich-Wilhems-Universität Bonn, Feb. 2021. URL: https://elib.dlr.de/143969/ (visited on 01/24/2022).

[17] Christophe Geuzaine and Jean-Francois Remacle. "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities". In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331. DOI: 10.1002/nme.2579.

[18] Johannes Holke et al. *t8code: Parallel AMR on hybrid non-conforming meshes*. 2022. URL: https://github.com/holke/t8code (visited on 07/19/2021).

[19] *Open CASCADE Technology*. URL: https://www.opencascade.com/open-cascade-technology/ (visited on 01/14/2022).

[20] Jens-Dominik Mueller and Michael Giles. "Solution adaptive mesh refinement using adjoint error analysis". In: *15th AIAA Computational Fluid Dynamics Conference*. Fluid Dynamics and Co-located Conferences. American Institute of Aeronautics and Astronautics, June 2001. DOI: 10.2514/6.2001-2550.

[21] Ralf Hartmann et al. *Error estimation and adaptive mesh refinement for aerodynamic flows*. Ed. by Norbert Kroll et al. 2010. URL: https://elib.dlr.de/67435/ (visited on 03/25/2022).

[22] H. G. Weller et al. "A tensorial approach to computational continuum mechanics using object-oriented techniques". In: *Computers in Physics* 12.6 (Nov. 1998). Publisher: American Institute of Physics, pp. 620–631. DOI: 10.1063/1.168744.

[23] Rainald Löhner and Paresh Parikh. "Generation of three-dimensional unstructured grids by the advancing-front method". In: *International Journal for Numerical Methods in Fluids* 8.10 (1988), pp. 1135–1149. DOI: 10.1002/fld.1650081003.

[24] Joachim Schöberl. "NETGEN An advancing front 2D/3D-mesh generator based on abstract rules". In: *Computing and Visualization in Science* 1.1 (July 1997), pp. 41–52. DOI: 10.1007/s007910050004.

[25] Nicolas Barral, Edward Luke, and Frédéric Alauzet. "Two Mesh Deformation Methods Coupled with a Changing-connectivity Moving Mesh Method for CFD Applications". In: *Procedia Engineering*. 23rd International Meshing Roundtable (IMR23) 82 (Jan. 2014), pp. 213–227. DOI: 10.1016/j.proeng.2014.10.385.

[26] A. De Boer, M. S. Van der Schoot, and H. Bijl. "New Method for Mesh Moving Based on Radial Basis Function Interpolation". In: *ECCOMAS CFD 2006: Proceedings of the European Conference on Computational Fluid Dynamics, Egmond aan Zee, The Netherlands, September 5-8, 2006* (2006). URL: https://repository.tudelft.nl/islandora/object/uuid%3A8db01fd1-94d0-40d7-8e99-9385bd63b92e (visited on 03/23/2022).

[27] Carsten Burstedde and Johannes Holke. "Coarse Mesh Partitioning for Tree-Based AMR". In: *SIAM Journal on Scientific Computing* Vol. 39 (2017), pp. C364–C392. DOI: 10.1137/16M1103518.

[28] Frank Günther et al. "A Cache-Aware Algorithm for PDEs on Hierarchical Data Structures Based on Space-Filling Curves". In: *SIAM Journal on Scientific Computing* 28.5 (2006), pp. 1634–1650. DOI: 10.1137/040604078.

[29] James R. Stewart and H. Carter Edwards. "A framework approach for developing parallel adaptive multiphysics applications". In: *Finite Elements in Analysis and Design* 40.12 (2004), pp. 1599–1617. DOI: https://doi.org/10.1016/j.finel.2003.10.006.

[30] W. Bangerth, R. Hartmann, and G. Kanschat. "Deal.II—A General-Purpose Object-Oriented Finite Element Library". In: *ACM Trans. Math. Softw.* 33.4 (Aug. 2007), 24–es. DOI: 10.1145/1268776.1268779.

[31] Ian Stroud, ed. *Boundary Representation Modelling Techniques*. London: Springer, 2006. ISBN: 978-1-84628-616-2.

[32] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer Science & Business Media, Nov. 1996. ISBN: 978-3-540-61545-3.

[33] R. Courant, K. Friedrichs, and H. Lewy. "Über die partiellen Differenzengleichungen der mathematischen Physik". In: *Mathematische Annalen* 100.1 (Dec. 1928), pp. 32–74. DOI: 10.1007/BF01448839.

[34] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*. Kitware, Inc.

[35] W. H. Reed and T. R. Hill. *Triangular mesh methods for the neutron transport equation*. Tech. rep. LA-UR-73-479; CONF-730414-2. Los Alamos Scientific Lab., N.Mex. (USA), Oct. 1973. URL: https://www.osti.gov/biblio/4491151-triangular-mesh-methods-neutron-transport-equation (visited on 03/02/2022).

[36] Bernardo Cockburn and Chi-Wang Shu. "The Runge-Kutta local projection P1-discontinuous-Galerkin finite element method for scalar conservation laws". In: *1st National Fluid Dynamics Conference*. Fluid Dynamics and Co-located Conferences. American Institute of Aeronautics and Astronautics, July 1988. DOI: 10.2514/6.1988-3797.

[37] Bernardo Cockburn and Chi-Wang Shu. "TVB Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws II: General Framework". In: *Mathematics of Computation* 52.186 (1989), pp. 411–435. DOI: 10.2307/2008474.

[38] Bernardo Cockburn, San-Yih Lin, and Chi-Wang Shu. "TVB Runge-Kutta local projection discontinuous Galerkin finite element method for conservation laws III: One-dimensional systems". In: *Journal of Computational Physics* 84.1 (Sept. 1989), pp. 90–113. DOI: 10.1016/0021-9991(89)90183-6.

[39] Bernardo Cockburn, Suchung Hou, and Chi-Wang Shu. "The Runge-Kutta Local Projection Discontinuous Galerkin Finite Element Method for Conservation Laws. IV: The Multidimensional Case". In: *Mathematics of Computation* 54.190 (1990). Publisher: American Mathematical Society, pp. 545–581. DOI: 10.2307/2008501.

[40] Bernardo Cockburn and Chi-Wang Shu. “The Runge–Kutta Discontinuous Galerkin Method for Conservation Laws V: Multidimensional Systems”. In: *Journal of Computational Physics* 141.2 (Apr. 1998), pp. 199–224. DOI: 10.1006/jcph.1998.5892.

[41] Fariba Mohammadi et al. “A Direct High-Order Curvilinear Triangular Mesh Generation Method Using an Advancing Front Technique”. In: *Computational Science – ICCS 2020*. Ed. by Valeria V. Krzhizhanovskaya et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 72–85. DOI: 10.1007/978-3-030-50417-5_6.

[42] Xiao-Juan Luo et al. “Automatic p-version mesh generation for curved domains”. In: *Engineering with Computers* 20.3 (Sept. 2004), pp. 273–285. DOI: 10.1007/s00366-004-0295-1.

[43] O. Sahni et al. “Curved boundary layer meshing for adaptive viscous flow simulations”. In: *Finite Elements in Analysis and Design* 46.1 (2010), pp. 132–139. DOI: 10.1016/j.finel.2009.06.016.

[44] S. J. Sherwin and J. Peiró. “Mesh generation in curvilinear domains using high-order elements”. In: *International Journal for Numerical Methods in Engineering* 53.1 (2002), pp. 207–223. DOI: 10.1002/nme.397.

[45] Zhong Q. Xie et al. “The generation of arbitrary order curved meshes for 3D finite element analysis”. In: *Computational Mechanics* 51.3 (Mar. 2013), pp. 361–374. DOI: 10.1007/s00466-012-0736-4.

[46] S. Dey, R. M. O’Bara, and M. S. Shephard. “Towards curvilinear meshing in 3D: the case of quadratic simplices”. In: *Computer-Aided Design* 33.3 (2001), pp. 199–209. DOI: 10.1016/S0010-4485(00)00120-2.

[47] Thomas Toulorge et al. “Robust untangling of curvilinear meshes”. In: *Journal of Computational Physics* 254 (Dec. 2013), pp. 8–26. DOI: 10.1016/j.jcp.2013.07.022.

[48] G. K. Batchelor. *An Introduction to Fluid Dynamics*. Cambridge Mathematical Library. Cambridge University Press, 2000. ISBN: 978-0-511-80095-5.

[49] Martin Siggel et al. “TiGL: An Open Source Computational Geometry Library for Parametric Aircraft Design”. In: *Mathematics in Computer Science* 13.3 (Sept. 2019), pp. 367–389. DOI: 10.1007/s11786-019-00401-y.

[50] Marko Alder et al. “Recent Advances in Establishing a Common Language for Aircraft Design with CPACS”. In: *Aerospace Europe Conference 2020*. 2020. URL: https://elib.dlr.de/134341/ (visited on 03/19/2022).

[51] Paul Putin. “Parametrische CFD-Netzgenerierung im Flugzeugvorentwurf”. bachelor. Hochschule Bonn-Rhein-Sieg, Feb. 2019. URL: https://elib.dlr.de/147200/ (visited on 03/20/2022).

[52] 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.

[53] Michael Schlottke-Lakemper et al. “A purely hyperbolic discontinuous Galerkin approach for self-gravitating gas dynamics”. In: *Journal of Computational Physics* 442 (June 2021). DOI: 10.1016/j.jcp.2021.110467.

A Conversion of parameters

As already mentioned in Sec. 3.1.2, we have to use OCCT's functionality to retrieve the parameters of OCCT vertices on OCCT edges and OCCT faces or to convert OCCT edge parameters into OCCT face parameters.

OCCT provides us with multiple functions to do so and we tried different solutions and yielded different results. The first classes we tried were the GeomAPI_ProjectPointOnCurve[1] and GeomAPI_ProjectPointOnSurf[2] class. Both of these get three-dimensional coordinates and try to find parameters of a point on the OCCT edge or OCCT face, which is an orthogonal projection of the input coordinates.

But due to the fact, that we only try to project boundary points (like the endpoints of an OCCT edge or points on the edges of an OCCT face) onto the OCCT edge or OCCT face, we do not always get a result. We however know, that there is a solution to this because our point does lie on the geometry.

We get better results, if we first use the aforementioned classes and if no parameters were retrieved, we try the same process again with ShapeAnalysis_Curve[3] or ShapeAnalysis_Surface[4]. This does not seem like the intended usage the OCCT developers thought of and it generates some problems with our geometry mapping.

Mainly, these problems occur with OCCT curves and OCCT surfaces, which have periodic parameters, like circles and cylinders. If the period T of a parameter of an OCCT curve is, for example, 2π, and the parameter we are searching for is π, then 3π, 5π and even $-\pi$ are also valid parameters to describe this point. And if our geometry mapping then interpolates between the parameters of nodes, it is relevant, that we have the right parameter. Therefore, we convert all projected parameters into $[0,\ T]$, but we still get deformed cells from the geometry mapping.

The last solution we test and which works with our algorithm is the usage of the BRep_Tool[5] class, which can access the topology information. With this topology information, the class can

[1] https://dev.opencascade.org/doc/refman/html/class_geom_a_p_i___project_point_on_curve.html

[2] https://dev.opencascade.org/doc/refman/html/class_geom_a_p_i___project_point_on_surf.html

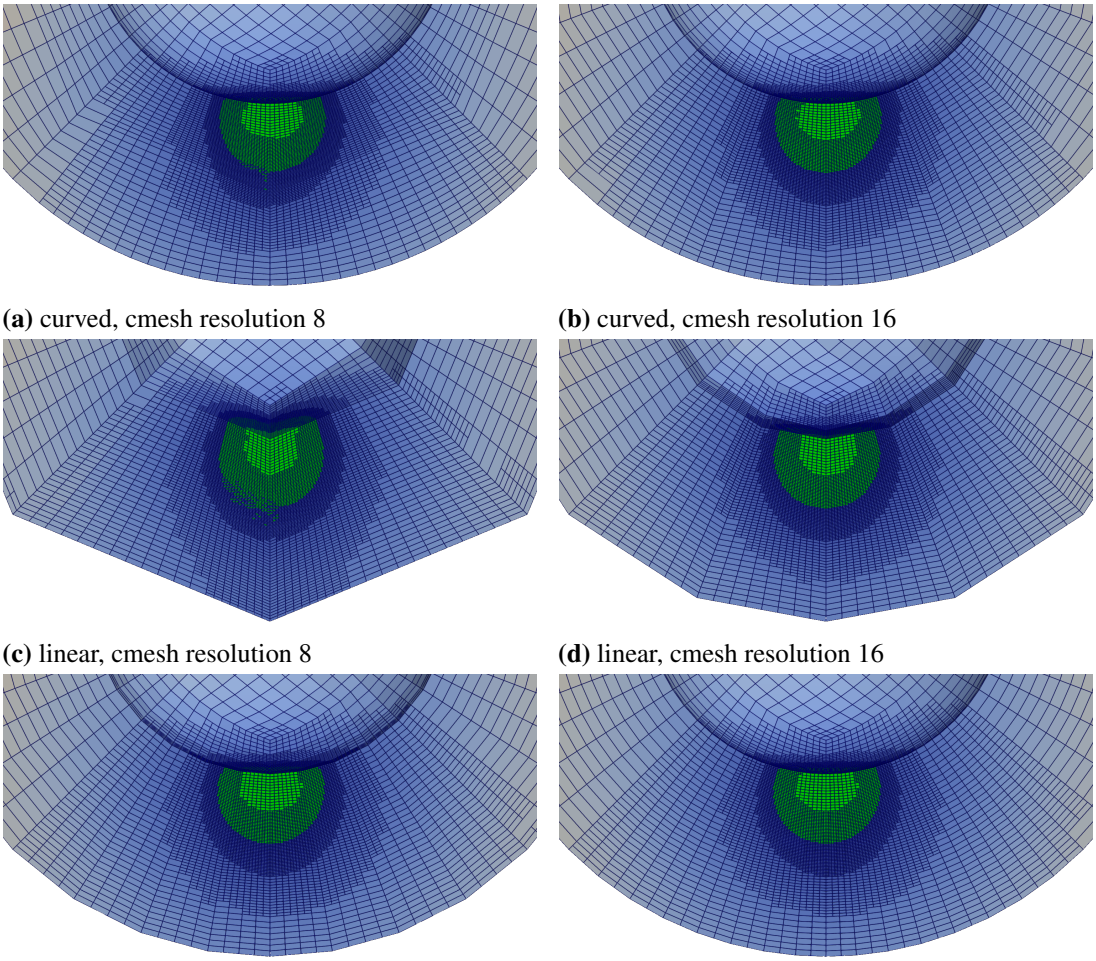
[3] https://dev.opencascade.org/doc/refman/html/class_shape_analysis___curve.html

[4] https://dev.opencascade.org/doc/refman/html/class_shape_analysis___surface.html

[5] https://dev.opencascade.org/doc/refman/html/class_b_rep___tool.html

convert the parameters rather than project points onto the geometry. This also solves the issue with periodic parameter spaces.

B Visualizations of the evaluation runs



(a) curved, cmesh resolution 8

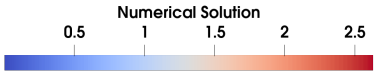
(b) curved, cmesh resolution 16

(c) linear, cmesh resolution 8

(d) linear, cmesh resolution 16

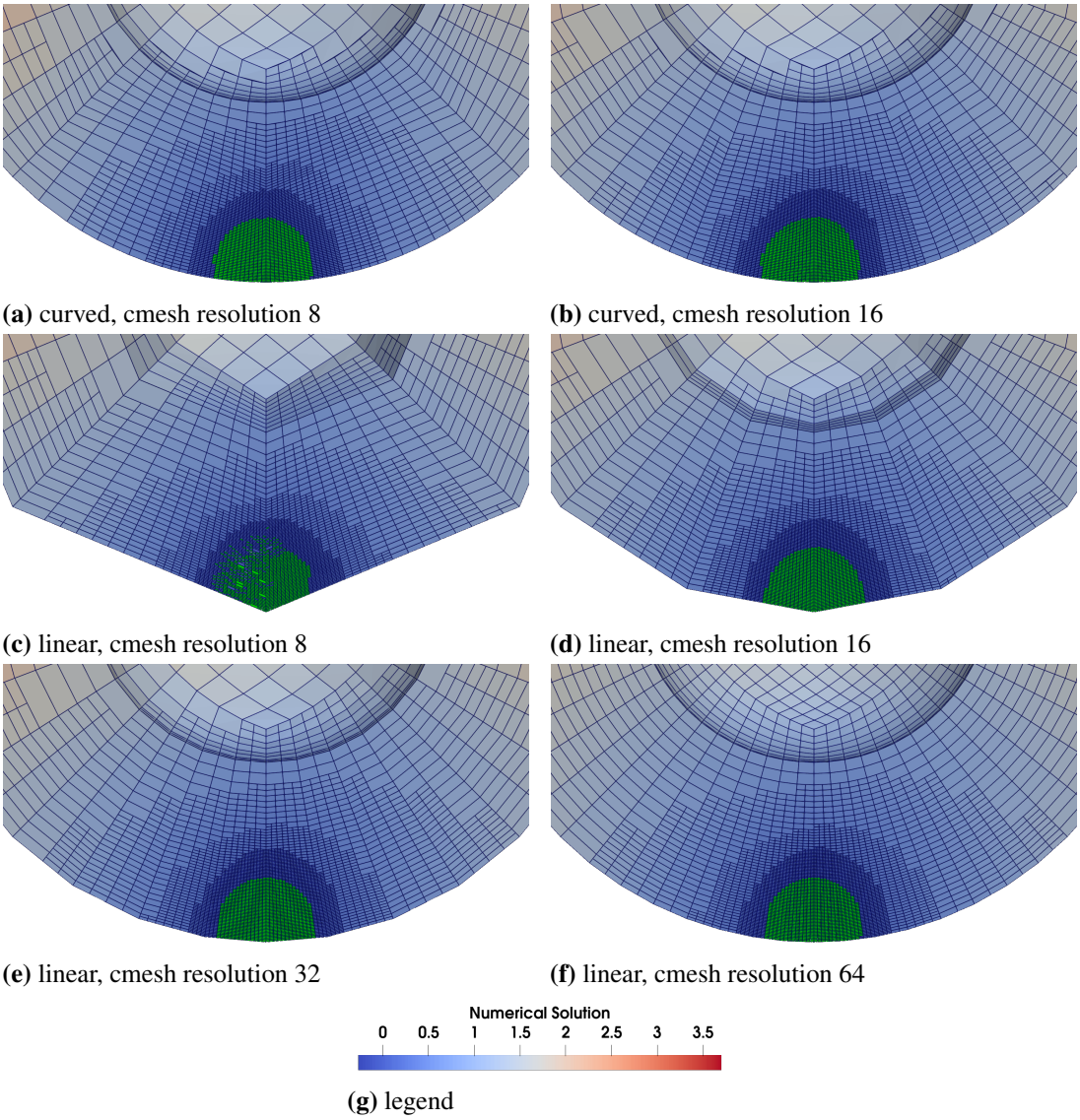
(e) linear, cmesh resolution 32

(f) linear, cmesh resolution 64



(g) legend

Figure B.1: Shown are visualizations of the last time step of each evaluation run where the concentration is advected along the inner, convex boundary of the sphere shell.



(a) curved, cmesh resolution 8

(b) curved, cmesh resolution 16

(c) linear, cmesh resolution 8

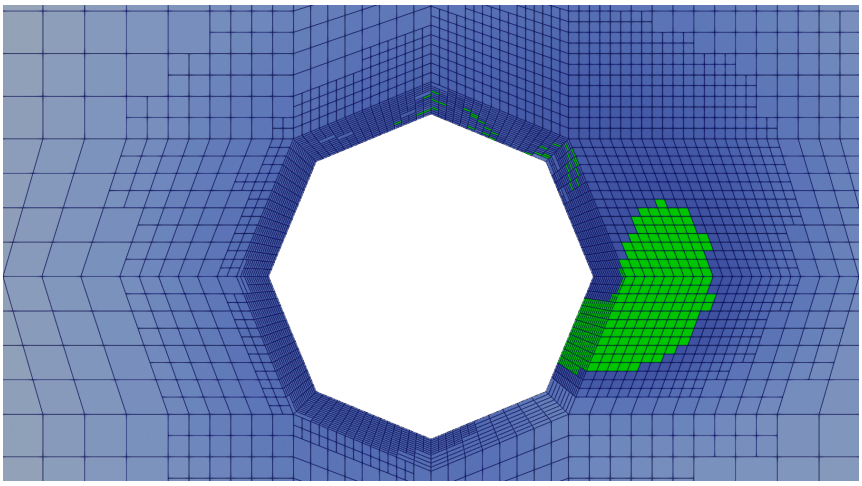
(d) linear, cmesh resolution 16

(e) linear, cmesh resolution 32

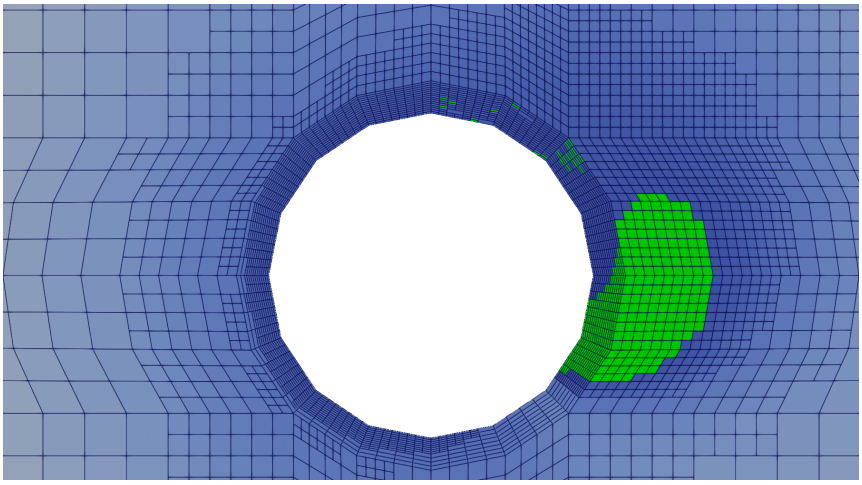
(f) linear, cmesh resolution 64

(g) legend

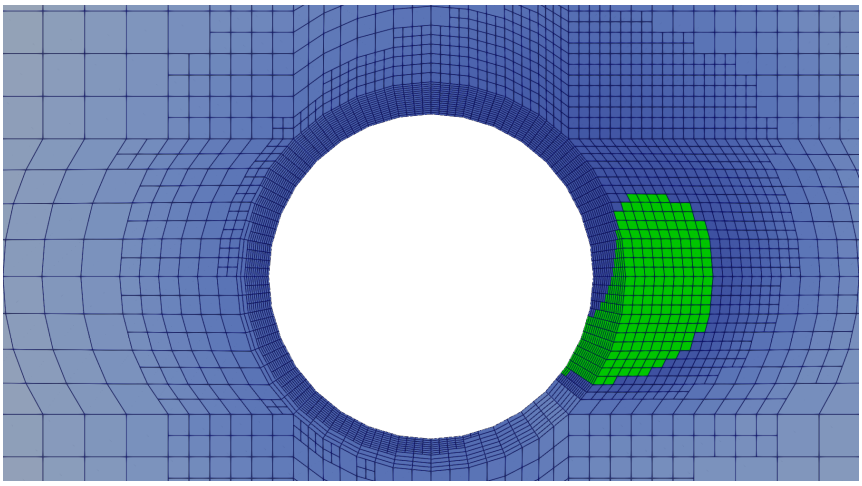
Figure B.2: Shown are visualizations of the last time step of each evaluation run where the concentration is advected along the outer, concave boundary of the sphere shell.



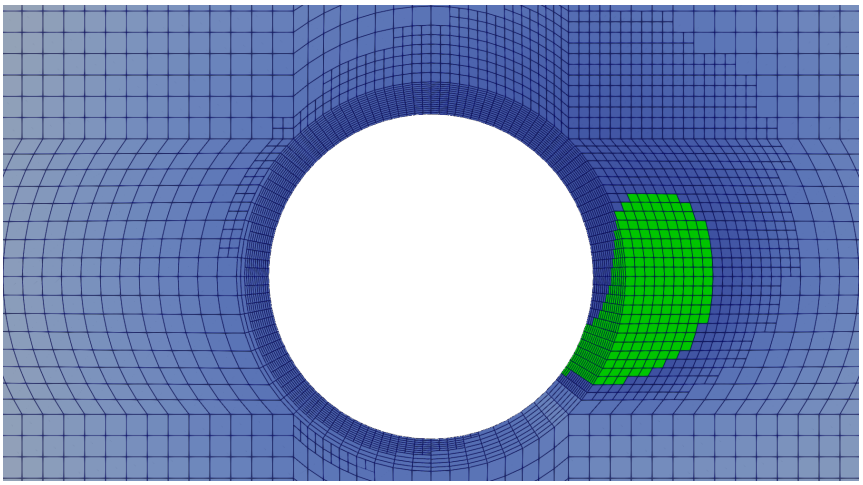
(a) linear, cmesh resolution 8, refinement level 4



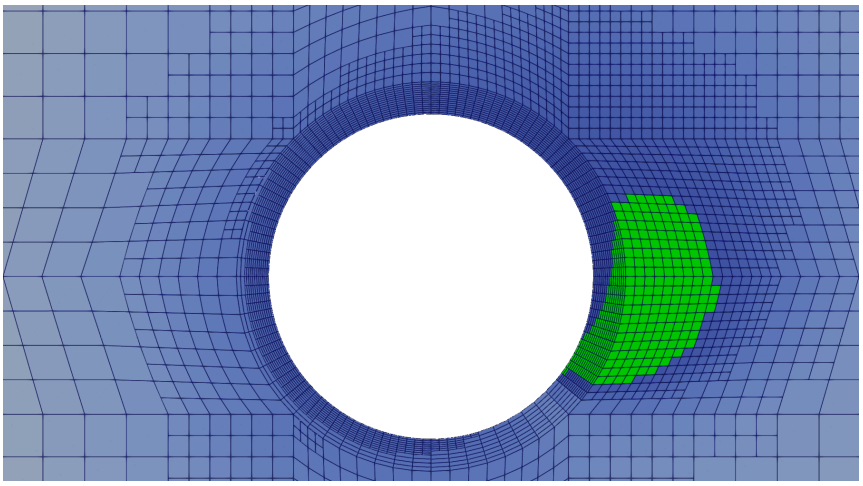
(b) linear, cmesh resolution 16, refinement level 3



(c) linear, cmesh resolution 32, refinement level 2



(d) linear, cmesh resolution 64, refinement level 1



(e) curved, cmesh resolution 8, refinement level 4

(f) legend

Figure B.3: Shown are visualizations of the last time step of each rotating cylinder evaluation run of the first set.