Mesh Curving Techniques for High Order Parallel Simulations on Unstructured Meshes

A thesis accepted by the Faculty
of Aerospace Engineering and Geodesy of the University of Stuttgart
in partial fulfillment of the requirements for the degree of
Doctor of Engineering Sciences (Dr.-Ing.)

by

Florian Hindenlang

born in Stuttgart

Main referee:	Prof. Dr. Claus-Dieter Munz
Co referee:	Prof. David A. Kopriva
Co referee:	Prof. Gregor J. Gassner
Date of defence:	September 19, 2014

Institute of Aerodynamics and Gas Dynamics
University of Stuttgart

2014

Für Mariona, Myriam, Simone und Ulrich.

Preface

This thesis was developed during my work as academic employee at the Institute of Aerodynamics and Gas Dynamics (IAG) of the University of Stuttgart.

I thank my doctoral supervisor Prof. Dr. Claus-Dieter Munz for the excellent working conditions in his research group and I am thankful for his unconditional support during the last six and a half years, especially encouraging the participation to many international conferences. Furthermore, I thank Prof. David Kopriva for being my co-referee, for taking the long journey from Tallahassee to Stuttgart. Many thanks also to my second co-referee Prof. Gregor Gassner, who always has been a mentor for me, never tired to answer my continuing questions and always open for discussions. In addition, I appreciated the detailed corrections of this work from all referees.

I thank all the colleagues at the institute, especially the numerics research group, for the excellent working atmosphere. Furthermore I thank Thomas Bolemann, not only for his great help in all hardware issues, but especially for sharing his programming skills and tricks and his large efforts in the software development. Moreover, I thank Hannes Frank for his comments after reviewing the first version of this work and Andrea Beck for long and fruitful discussions.

The projects related to this work were supported by the ‘Bundesministerium für Bildung und Forschung’, the ‘Robert Bosch GmbH’ and the ‘SimTech cluster of excellence’ and the supercomputing center HLRS in Stuttgart.

Last but not least I am profoundly grateful for the love, the patience and the constant support of my parents Simone and Ulrich Hindenlang and my girlfriend Mariona Massana, who always believed in me.

Stuttgart, September 30, 2014

Florian Hindenlang

Contents

Preface v

Contents vii

Symbols and Abbreviations xi

Kurzfassung xiii

Abstract xv

1. Introduction 1

2. Numerics 7
- 2.1. The Element Mapping 7
  - 2.1.1. Derivatives under Coordinate Transformation 9
- 2.2. Mapping the Equations 12
- 2.3. Discontinuous Galerkin Formulation 13
  - 2.3.1. Weak and Strong Form 14
  - 2.3.2. Second Order Equations 16

3. Discretization 19
- 3.1. Polynomial Approximation 21
- 3.2. Nodal DG Scheme for General Elements 27
  - 3.2.1. Nodal DG with Flux Interpolation 30
  - 3.2.2. Discrete Equivalence of the Weak and Strong Form 32
- 3.3. DG-SEM for Hexahedral Elements 33
  - 3.3.1. Interpolation and Discrete Projection 34
  - 3.3.2. Approximation of the Solution and the Fluxes 35
  - 3.3.3. Discrete Metric Terms and Free-stream Preservation 36
  - 3.3.4. Integral of the Time Derivative 37
  - 3.3.5. Volume Integral 38
  - 3.3.6. Surface Integral 40

3.3.7. Semi-discrete Formulation 41
3.3.8. Lifting for Second Order Equations with DG-SEM . . . 43

4. Curved Mesh Generation 47
4.1. Overview 48
4.2. Surface Reconstruction by Normal Vectors 51
4.2.1. CAD Interface 54
4.2.2. Volume Mapping from Blending Curved Edges 55
4.3. Surface Curving with Refined Surface Data 58
4.4. Volume Curving using Block-Structured Meshes 61
4.4.1. Quality of the Interpolation of Stretched Meshes 63
4.5. Volume Curving by Mesh Deformation 67
4.6. Comparison of Surface Curving Techniques 78
4.6.1. Interpolation of a Circular Arc 78
4.6.2. Interpolation of a NACA0012 Airfoil 83
4.6.3. Spherical Quadrilateral 89
4.6.4. Influence of Mesh Generation Errors 92
4.7. Applications 99
4.7.1. Coaxial Waveguide 99
4.7.2. Complex Three-dimensional Hybrid Curved Mesh of the DLR-F6 102
4.8. Evaluation of the Curving Strategies 106

5. Parallelization Aspects 109
5.1. Preprocessing for Parallel Simulations 109
5.1.1. Sorting of Elements along the Space-Filling Curve . . . 110
5.1.2. Domain Decomposition with the Space-Filling Curve . . 112
5.2. Parallelization Concept 115
5.3. Parallel Performance Analysis 120
5.3.1. Hardware and Memory Consumption 121
5.3.2. Strong Scaling 122
5.3.3. Performance Index 125
5.3.4. Weak Scaling 130
5.3.5. Sphere Flow Simulation 131

6. Conclusion and Prospects 137
6.1. Conclusions 137
6.2. Prospects 139

A. Orthogonal Basis Functions on Triangles, Tetrahedra, Prisms and Pyramids 141
A.1. Jacobi Polynomials 141
A.2. Orthonormal Basis of a Triangle 142
A.3. Orthonormal Basis of a Tetrahedron 144
A.4. Orthonormal Basis of a Prism 147
A.5. Orthonormal Basis of a Pyramid 149

B. General Gauss-Type Quadrature Rules 153

C. Element Mappings 155
C.1. Jacobian of Straight-Edge Tensor-Product Elements 159

D. Blending Faces to Elements 161

E. Multiple Node Detection with Space-Filling Curves 165

Bibliography 169

List of Tables 179

List of Figures 181

Lebenslauf 187

Symbols and Abbreviations

Symbols

E	Reference element
∂E	Reference element boundary
$\mathbb{L}_2$	Space of square-integrable functions
N	Polynomial degree
N_g	Polynomial degree of the element mapping
$\boldsymbol{n}$	Normal vector in physical space
$\boldsymbol{N}$	Normal vector in reference space
n_e	Total number of elements
$n_{e/c}$	Number of elements per core (Parallelization)
n_{IP}	Number of interpolation points per element
n_{IP}^s	Number of interpolation points per element side
$\mathcal{Q}$	Physical element
$Q^d = \frac{\partial U}{\partial x^d}$	Gradient of conservative variables in x^d direction
t	Time
U	Vector of conservative variables
$\phi(\boldsymbol{\xi})$	Multivariate test function
$\psi(\boldsymbol{\xi})$	Multivariate Lagrange polynomial basis function
$\ell(\xi)$	One-dimensional Lagrange polynomial basis function
$\varphi(\boldsymbol{\xi})$	Multi-variate orthonormal polynomial basis function
Ω	Computational domain
$\boldsymbol{X} = (X_1, X_2, X_3)^T$	Coordinates in 3D physical space, also $(x, y, z)^T$
$\boldsymbol{\xi} = (\xi^1, \xi^2, \xi^3)^T$	Coordinates in 3D reference space, also $(\xi, \eta, \zeta)^T$

Abbreviations

1D, 2D, 3D	one, two, three dimensions / -dimensional
CG	Continuous Galerkin
CL	Chebychev-Lobatto (point distribution)
CAD	Computer-Aided Design
CFD	Computational Fluid Dynamics
CGNS	CFD Generation Notation System
DG	Discontinuous Galerkin
DG-SEM	Discontinuous Galerkin Spectral Element Method
DOF	Degree(s) of Freedom
DOF/c	Degree(s) of Freedom per core
FV	Finite Volume
FD	Finite Differences
HOPR	High Order Preprocessor
HPC	High Performance Computing
LGL	Legendre-Gauss-Lobatto
MPI	Message-Passing-Interface (Parallelization)
ODE	Ordinary Differential Equation
PID	Performance Index (Parallelization, Eq. (5.9))
SEM	Spectral Element Method
STEP	STandard for the Exchange of Product model data

Kurzfassung

Diese Arbeit befasst sich mit der Erstellung von dreidimensionalen hybriden Gittern hoher Ordnung und deren Verwendung. Die Standardelemente heutiger Vernetzungssoftware besitzen fast ausschließlich lineare Elementkanten. Bei industriellen Anwendung sind die zu vernetzenden Geometrien sehr komplex und weisen meist gekrümmte Gebietsgrenzen vor. Bei der Verwendung von Verfahren hoher Ordnung ist es im Unterschied zu klassischen Verfahren niedriger Ordnung notwendig, die Geometrie ebenfalls mit hoher Ordnung darzustellen. Die Diskretisierung der gekrümmten Randflächen erfolgt also durch eine höherwertige Abbildung der Elemente. Es wird die Grundidee verfolgt, dass vorhandene lineare Vernetzungssoftware weiterhin genutzt werden kann und Elemente hoher Ordnung mithilfe zusätzlich bereitgestellter Informationen generiert werden sollen.

Innerhalb der Verfahren hoher Ordnung ist das *discontinuous Galerkin* (DG) Verfahren ein vielversprechender Kandidat für zukünftige Strömungslöser. Es handelt sich um ein lokal konservatives Verfahren. Die Lösung wird innerhalb des Elements als stetiges Polynom dargestellt und ist über die Elementgrenzen hinweg unstetig. Die Elemente koppeln nur mit direkten Nachbarelementen und aufgrund der Unstetigkeit an der Elementgrenze werden hierfür numerische Flüsse verwendet.

Da ein Hauptaugenmerk dieser Arbeit auf der Behandlung von gekrümmten Elementen liegt, werden die unterschiedlichen Formulierungen und mögliche Implementierungen des DG Verfahrens auf Elementen mit nicht-linearen Abbildungen im Detail diskutiert. Insbesondere wird die Discontinuous Galerkin Spektrale Element Methode (DG-SEM) vorgestellt, eine besonders effiziente Implementierung für Hexaederelemente.

Der Schwerpunkt dieser Arbeit liegt auf der Generierung von Gittern hoher Ordnung. Es werden unterschiedliche Techniken zur Erstellung von Elementen hoher Ordnung beschrieben und deren Anwendbarkeit auf komplexe Geometrien demonstriert. Ausgehend von einem linearen Netz erfolgt in einem ersten Schritt die Krümmung der Elementflächen, die an der gekrümmten Randbedingung anliegen.

Hierbei wird zwischen zwei Ansätzen unterschieden, der erste basierend auf

Normalenvektoren an den Oberflächenpunkten, der zweite auf Interpolation von zusätzlich generierten Oberflächenpunkten. Die volumetrische Abbildung des Elements wird dann durch eine Linearkombination der gekrümmten Elementflächen gebildet.

Im Fall von Grenzschichtnetzen kann die Linearkombination zu ungültigen Elementabbildungen führen. Ein gültiges Gitter kann durch eine zusätzliche Gitterverformung generiert werden. Die gesamte Vorgehensweise wird anhand einem Grenzschichtnetz von einem Flügelprofil erläutert und validiert. Hiervon unabhängig wird ein weiterer Ansatz beschrieben, bei dem man die Volumenabbildung direkt durch Agglomeration block-strukturierter Gitter erhält.

Einer der Gründe für die Attraktivität von DG Verfahren zur Simulation von Strömungen ist deren parallele Effizienz [60]. Da zukünftige Anwendungen in der Strömungsmechanik die Auflösung von instationären dreidimensionalen Effekten umfassen und eine wachsende Komplexität aufweisen, ergibt sich ein steigender Bedarf an Rechenressourcen, und die schwache und starke Skalierbarkeit des numerischen Verfahrens spielt eine entscheidende Rolle.

Daher wird im letzten Teil dieser Arbeit das Parallelisierungskonzept des DG-SEM Lösers *Flexi* vorgestellt. Es wird eine neue Strategie zur Gebietszerlegung erläutert, die auf raum-füllenden Kurven basiert und daher besonders einfach und flexibel ist. Eine ausführliche Analyse der parallelen Performance bestätigt, dass die gesamte Implementierung perfekt skaliert und einen idealen Speed-up bis auf ein Element pro Kern für hohe Polynomgrade aufweist. Da das DG Verfahren nur mit den direkten Nachbarelementen kommunizieren muss, konnte gezeigt werden, dass die parallele Performance unabhängig davon ist, ob ein kartesisches oder voll unstrukturiertes Gitter verwendet wird. Die Ergebnisse zeigen, dass das hier vorgestellte Discontinuous Galerkin Verfahren ein großes Potential für hochaufgelöste Simulationen auf heutigen und zukünftigen Supercomputern aufweist.

Abstract

In this work, the generation of high order curved three-dimensional hybrid meshes and its application are presented. Meshes with linear edges are the standard of today's state-of-the-art meshing software. Industrial applications typically imply geometrically complex domains, mostly described by curved domain boundaries. To apply high order methods in this context, the geometry – in contrast to classical low order methods – has to be represented with a high order approximation, too. Therefore, a high order element mapping has to be used for the discretization of curved domain boundaries. The main idea here is to rely on existing linear mesh generators and provide additional information to produce high order curved elements.

A very promising candidate for future numerical solvers in computational fluid dynamics is the family of high order *discontinuous Galerkin* (DG) schemes. They are locally conservative schemes, with a continuous polynomial representation within each element and allow a discontinuous solution across element faces. Elements couple only to direct face neighbors, and the discontinuity is resolved via numerical flux functions. As the main focus of this work are curved elements, the different formulations and possible implementations of the DG scheme with non-linear element mappings are discussed in detail. Especially, a highly efficient variant of the DG scheme for hexahedra, namely the discontinuous Galerkin spectral element method (DG-SEM), is presented.

The main focus of this thesis is the generation of high order meshes. Several techniques to generate curved elements are described and their applicability to complex geometries is demonstrated. Starting from a linear mesh, the first step curves the element faces representing the curved geometry. Two approaches are presented, the first based on continuity conditions using surface normal vectors and the second based on interpolation of additionally generated surface points. The high order mapping of the volumetric element is computed as a blending of the curved element faces.

In the case of boundary layer meshes, the blending may lead to inverted elements. As a remedy to this problem, an additional mesh deformation approach is proposed and validated. Independent thereof, another approach is presented, allowing one to directly generate curved volume mappings from the agglomer-

ation of block-structured meshes.
One of the reasons making high order DG schemes attractive for the simulation of fluid dynamics is their parallel efficiency [60]. As future applications in fluid dynamics comprise the resolution of three-dimensional unsteady effects and are increasingly complex, the simulations require more and more computing resources, and weak and strong scalability of the numerical method becomes extremely important. In the last part of this thesis, the parallelization concept of the DG-SEM code *Flexi* is described in detail. A new domain decomposition strategy based on space filling curves is introduced, and is shown to be simple and flexible. A thorough parallel performance analysis confirms that the overall implementation scales perfectly. Ideal speed-up is maintained for high polynomial degrees, up to the limit of one element per core. As the DG scheme only communicates with direct neighbors, the same parallel efficiency is found on both cartesian meshes as well as fully unstructured meshes. The findings underline that the proposed Discontinuous Galerkin scheme exhibit a great potential for highly resolved simulations on current and future large scale parallel computer systems.

1. Introduction

The simulation of three-dimensional flows is a very important tool in the design phase of many industrial products, for example the design of aircrafts, turbines, cars or engines.

The first key point in all simulations is the discretization of the physical domain by a computational mesh. The choice of the mesh type is normally driven by the numerical scheme. In Computational Fluid Dynamics (CFD), block-structured meshes have been extensively used in the past, on the one hand because of the use of Finite Difference (FD) schemes, on the other hand also because of their easily enforced high mesh quality. Finite Volume (FV) or Finite Element methods (FEM) are designed for unstructured meshes. Whereas block-structured mesh generation is a very time-consuming task, fast automatic mesh generators are commonly available to produce unstructured meshes. This becomes more and more important in industrial applications, where geometries are very complex and unstructured meshes are often the only choice, since they inherit more geometric flexibility. Therefore, most commercial CFD tools rely on unstructured meshes with either FV or FEM schemes.

The second key point is the increasing computational power, which makes high resolutions possible and drives simulations towards large scale massively parallel computations. In this context, the discontinuous Galerkin (DG) method is a promising candidate for future numerical schemes, since it enables high order accuracy on unstructured meshes and provides high parallel scalability due to its element-local formulation.

While discontinuous Galerkin methods were first introduced in the early 1970's in [86], these methods gained popularity in the 1990's by initial works of Cockburn and Shu [29,31–33,35], making DG a powerful computational tool for the solution of systems of conservation laws [8,13]. Bassi and Rebay extended the method to problems of viscous gas dynamics [6,9], which led to related DG formulations [56,66,83] for the compressible Navier-Stokes equations. Many examples and further details are found in the books of Cockburn, Karniadakis and Shu [30] and Hesthaven and Warburton [58].

DG methods rely on a variational formulation where the solution inside the elements is approximated by a high order polynomial expansion. Solutions are discontinuous across element interfaces, which introduces the need for numerical flux functions, which are adopted from FV schemes. From this starting point, further decisions regarding the implementation specifics have to be made. We can use any element shape from hexahedra to prisms, pyramids, tetrahedra or even polyhedra [49] for discretizing the domain. We can represent the polynomials by a nodal or a modal basis and use a set of tensor-product or complete order basis functions. We can choose the type of approximation and integration of volume and surface fluxes. The choice will significantly influence memory requirements, the number of operations, accuracy and robustness. In addition, different decisions may be taken when using an implicit or an explicit time discretization. We are interested in large scale time-dependent non-linear problems, thus a highly scalable code parallelization is crucial. Only explicit time stepping will fully exploit the locality of the DG scheme, meaning that only surface data of neighboring elements have to be communicated. In addition, the ratio of computational effort to communication will increase with increasing polynomial degree and hence results in higher parallel efficiency.

When using high order methods, wall boundary conditions need a high order representation of the wall normal. Bassi and Rebay [7] were the first to show that in the case of an isentropic flow across a cylinder a high order DG discretization with straight-sided 2D elements leads to low order accuracy and even physically wrong results. To overcome this issue, they proposed to use at least elements with parabolic shaped sides on the boundary to represent the shape of the cylinder.

In Chapter 2, we derive the DG method for first and second order partial differential equations of conservation laws and discuss two discretizations, the nodal DG scheme for general elements and the DG-SEM for hexahedra. The derivations will focus on DG discretizations involving the metrics of high order element mappings, which is the most general case, since it includes the linear element mapping as a special case. Implementation issues such as interpolation and numerical integration will be discussed with regards to the errors introduced for curved elements.

A main question to be addressed in this work is how to generate the high order element mappings, since standard mesh generators typically provide only elements with linear edges. In 1998, Bonhaus [21] stated in the conclusions of his PhD thesis about high order Finite Element Methods for viscous compressible flow:

> “The primary difficulty in using the higher-order schemes was the generation of suitable higher-order discretizations of the flow do-

> main when curved boundaries were involved. No general solution was found - each case required its own preprocessor to generate the higher-order finite-element mesh from an existing structured mesh or linear finite-element mesh. Particularly difficult to generate are meshes that are highly stretched to compute viscous flows - simply moving the boundary control points to match the surface creates overlapping elements which are unacceptable to the scheme. These problems are further compounded in three dimensions. To facilitate application of the higher-order schemes in general, grid generation techniques must be adapted to handle high-order discretizations and must be more closely tied to geometry definitions (e.g. CAD systems). This remains as an avenue of future work."

Almost 10 years later in 2007, Wang et al. [97] published an article reviewing several unstructured high order codes for Euler and Navier-Stokes equations, and concluded about high order grid generation:

> "Almost all researchers in high-order methods pointed out the importance of high-order boundary representation. Many demonstrated the inadequacy of piece-wise linear facet representation used in low-order methods. As most grid generation packages were developed for lower-order methods, capabilities should be added to generate coarser grids with high-order (at least quadratic) boundaries. The generation of highly clustered viscous meshes near high-order walls is another challenge, as more elements near the boundary layer must be better than linear to avoid grid lines crossing into each other."

In 2009, similar conclusions were drawn in the European project ADIGMA [74], which focused in the industrialization of high order methods:

> "Finally, as high-order methods target same accuracy levels with coarser and coarser meshes as the approximation order increases, the quality of the mesh generation becomes more important. Care must be placed into generating meshes where higher-order elements with high aspect ratio have valid curved geometries. Achieving this on arbitrarily complex geometries is a challenge for the future."

The process of three-dimensional mesh generation is the crucial part before running a simulation. Before the mesh is actually generated, one has to start with geometry cleanup, since CAD geometry definitions are often ill-suited, either due to insufficient accuracy, for example non water-tight geometries, or due to excessive detail not essential for the analysis. The geometry cleanup also consists of splitting and merging the CAD surface patches to facilitate the mesh generation. The mesh resolution has to be adapted not only to geometric features, but also to the type of problem and the numerical method considered. For example, boundary layer meshes are only needed for viscous simulations and FV schemes will require meshes with a smaller element size than high order DG schemes for the same resolution. We want to stress that the mesh itself represents the parametrization needed for the simulation, whereas the CAD only defines the position of the boundaries. The CAD surfaces patches, typically defined by non-uniform rational B-Splines (NURBS) have a completely different parametrization. Thus, the mesh generator represents the link between the NURBS definition of CAD surfaces and the simulation-driven mesh parametrization, and guarantees that the surface points lie on the CAD definition.

Due to the described complexity of the mesh generation process, we have to rely on existing (commercial) mesh generation software. The difficulty about high order mesh generation is that nearly all existing meshing software generates meshes where the element edges are straight lines, at most having an additional mid-point. Once the grid is exported, the association of mesh points and the CAD geometry is basically lost.

The open-source project GMSH [50, 51] is an exception, as it generates linear meshes for a given CAD model and is able to create additional high order points for each element and project them onto the CAD geometry to generate high order grids. However, CAD handling and meshing capability for complex geometries and unstructured three-dimensional grids has not yet reached the level of commercial grid generators. A completely novel approach was introduced by Hughes et al. [62], the so-called isogeometric analysis, using the NURBS basis functions directly from the CAD model definition for the approximation of the solution, thus geometric features can be represented exactly. However, since CAD definitions and parametrization do not match the simulation requirements for complex configurations, special techniques to re-parametrize the CAD definitions would be needed, resembling the generation of a blocking for block-structured meshes.

A main focus of this work is the development of a framework for the curving of three-dimensional unstructured meshes for high order simulations, but fully relying on existing linear mesh generators. In Chapter 4 we propose several strategies to curve the surface mesh, for example by using point-normals from CAD definitions or surface meshes created by subdivision. We also present a mesh deformation approach to propagate the surface curving into the volume, to resolve the problem of inverted boundary layer elements. A distinct strategy is proposed for block-structured meshes, providing a tensor-product volume parametrization to directly derive three-dimensional curved hexahedra. All these strategies are implemented in the standalone high order preprocessor tool *HOPR*. It fills the gap between the linear grid generation and the simulation with the high order solver. We show the applicability of the surface curving for a complex hybrid mesh of a full aircraft, we analyze in detail the accuracy of the polynomial approximations for simple geometries and study the sensibility of the high order approximation to errors arising from tolerances of the mesh generation process.

The key requirement to perform large scale three-dimensional simulations is an efficient parallelization of the code optimized for high-performance computing (HPC) architectures. Since today's processor hardware already makes a transition from multi- to many-core CPUs, efficient parallelization for very large numbers of cores is necessary. The first step of a parallel simulation is the domain decomposition of the mesh, assigning one domain to each core. The global mesh connectivity information is needed to set up the communication pattern between the cores. In Chapter 5 we present a novel approach for the domain decomposition, using a unique element sorting along a space-filling curve. This reduces the task of the domain decomposition to a splitting of a one-dimensional list of elements into pieces with the same weight, for a given but arbitrary number of cores. The sorting is done once, and is incorporated to *HOPR*. Furthermore, we analyze the parallel performance of the DG-SEM code *Flexi*, focusing on the strong scaling up to one element per core. We further investigate the scaling for different polynomial degrees and show the applicability and scalability of the code along with the domain decomposition for the simulation of a flow past a sphere on an unstructured mesh, with up to 4096 cores.

Overall, the preprocessor *HOPR* and the DG solver *Flexi* form a tool-chain for the parallel simulation of complex flows on unstructured curved meshes.

2. Numerics

In this chapter, we discuss the Discontinuous Galerkin (DG) scheme and the implementation, with special attention to curved elements. First, we introduce the element mapping and its derivatives, the metric terms. The element mapping is then employed to transform the equations from the physical to the reference space. The DG scheme is formulated analytically on the transformed equations, where we also describe the treatment of second order equations. In the first part of the section on discretization, we present the nodal and several modal polynomial basis functions and assess their properties. In the following, we derive the discretized nodal DG scheme on general elements. Finally, the DG-SEM discretization is presented, where the use of tensor-product nodal basis functions results in a very efficient DG scheme. We conclude this chapter with the DG-SEM implementation of the lifting for second order equations.

2.1. The Element Mapping

We can express the mapping of an element from reference space $\boldsymbol{\xi}$ to physical space $\boldsymbol{x}$ by

$$\boldsymbol{X}(\boldsymbol{\xi})\,, \tag{2.1}$$

with

$$\begin{aligned} \boldsymbol{X} &= (X_1, X_2, X_3)^T &&= (x, y, z)^T\,,\\ \boldsymbol{\xi} &= (\xi^1, \xi^2, \xi^3)^T &&= (\xi, \eta, \zeta)^T\,. \end{aligned} \tag{2.2}$$

In practice, we represent the mapping by an interpolation polynomial

$$\boldsymbol{X}(\boldsymbol{\xi}) = \sum_{l=1}^{n_{\mathrm{IP}}} \hat{\boldsymbol{x}}_l \psi_l(\boldsymbol{\xi})\,, \tag{2.3}$$

using a set of n_{IP} interpolation nodes, with physical coordinates $\{\hat{\boldsymbol{x}}_l\}_{l=1}^{n_{\mathrm{IP}}}$ and corresponding reference coordinates $\{\hat{\boldsymbol{\xi}}_k\}_{k=1}^{n_{\mathrm{IP}}}$, and the Lagrange basis functions $\psi_l(\boldsymbol{\xi})$ of polynomial degree N_g, having the cardinality property

$$\psi_l(\hat{\boldsymbol{\xi}}_k) = \delta_{kl}\,. \tag{2.4}$$

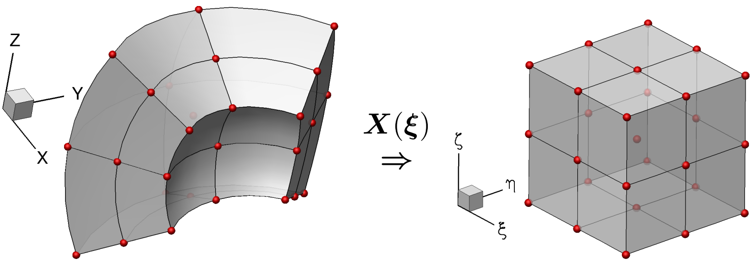


Figure 2.1.: Mapping of a hexahedral element from physical to reference space

The choice of the reference node positions is free. However, it is common to include the boundaries of the element, so that the basic linear mapping ($N_g = 1$), using only the elements corner nodes, is included. If we use an equidistant spacing of the nodes in reference space, we guarantee that the transformation remains linear if the interpolation nodes are equidistant in physical space. In addition, these node positions are very easy to implement for all standard element types. However, equidistant nodes can only be used up to a polynomial degree of $N_g \lessapprox 8$, since they become oscillatory for high order interpolation (Runge's phenomenon).
In Fig. 2.1, the quadratic mapping ($N_g = 2$) of a curved hexahedral element is shown. The element has $n_{\mathrm{IP}} = (N_g + 1)^3$ interpolation nodes and the reference domain is $\boldsymbol{\xi} \in [-1, 1]^3$. The reference node positions are equidistant

$$\hat{\boldsymbol{\xi}}_{ijk} = (\hat{\xi}_i, \hat{\xi}_j, \hat{\xi}_k)^T \quad \hat{\xi}_i = 2\left(\frac{i}{N_g}\right) - 1, \quad i, j, k = 0, \ldots, N_g \,, \tag{2.5}$$

and the hexahedral element mapping reads as

$$\boldsymbol{X}(\boldsymbol{\xi})_{(\mathrm{Hex})} = \sum_{ijk=0}^{N_g} \hat{\boldsymbol{x}}_{ijk} \ell_i(\xi) \ell_j(\eta) \ell_k(\zeta) \,, \tag{2.6}$$

with a tensor product polynomial basis consisting of one-dimensional Lagrange basis functions $\ell_i(\xi)$, defined as

$$\ell_j\left(\xi\right) = \prod_{\substack{i=0\\ i\neq j}}^{N} \frac{\left(\xi-\hat{\xi}_i\right)}{\left(\hat{\xi}_j-\hat{\xi}_i\right)}\,, \quad j=0,\dots,N\,, \tag{2.7}$$

with the cardinality property

$$\ell_j(\hat{\xi}_i) = \delta_{ij}\,, \quad i,j=0,\dots,N\,. \tag{2.8}$$

2.1.1. Derivatives under Coordinate Transformation

Before we transform the equation, we present some basics of coordinate transformations. First, we introduce the gradients in physical and reference space

$$\boldsymbol{\nabla}_{\boldsymbol{X}} = \begin{pmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} \end{pmatrix}, \quad \boldsymbol{\nabla}_{\boldsymbol{\xi}} = \begin{pmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \zeta} \end{pmatrix}. \tag{2.9}$$

We apply the chain rule for the derivatives of a general function $g(\boldsymbol{X})$ in reference space

$$\begin{aligned} \frac{\partial g}{\partial \xi} &= \frac{\partial x}{\partial \xi}\frac{\partial g}{\partial x} + \frac{\partial y}{\partial \xi}\frac{\partial g}{\partial y} + \frac{\partial z}{\partial \xi}\frac{\partial g}{\partial z} \\ \frac{\partial g}{\partial \eta} &= \frac{\partial x}{\partial \eta}\frac{\partial g}{\partial x} + \frac{\partial y}{\partial \eta}\frac{\partial g}{\partial y} + \frac{\partial z}{\partial \eta}\frac{\partial g}{\partial z} \\ \frac{\partial g}{\partial \zeta} &= \frac{\partial x}{\partial \zeta}\frac{\partial g}{\partial x} + \frac{\partial y}{\partial \zeta}\frac{\partial g}{\partial y} + \frac{\partial z}{\partial \zeta}\frac{\partial g}{\partial z}\,. \end{aligned} \tag{2.10}$$

Written in matrix vector notation, this reads as

$$\begin{pmatrix} g_\xi \\ g_\eta \\ g_\zeta \end{pmatrix} = \begin{pmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} & \frac{\partial z}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} & \frac{\partial z}{\partial \eta} \\ \frac{\partial x}{\partial \zeta} & \frac{\partial y}{\partial \zeta} & \frac{\partial z}{\partial \zeta} \end{pmatrix} \begin{pmatrix} g_x \\ g_y \\ g_z \end{pmatrix}, \tag{2.11}$$

$$\boldsymbol{\nabla}_{\boldsymbol{\xi}}\, g = \boldsymbol{\mathcal{J}}\,\boldsymbol{\nabla}_{\boldsymbol{X}}\, g\,, \tag{2.12}$$

where $\boldsymbol{\mathcal{J}}$ is the Jacobian matrix of the mapping $\boldsymbol{X}(\boldsymbol{\xi})$.
Since we are interested in expressing the derivatives in physical space with derivatives in reference space, we need the inverse of the Jacobi matrix

$$\boldsymbol{\nabla}_{\boldsymbol{X}}\, g = \boldsymbol{\mathcal{J}}^{-1}\boldsymbol{\nabla}_{\boldsymbol{\xi}}\, g\,. \tag{2.13}$$

We want to express the inverse of the Jacobian matrix with the derivatives of the mapping $\boldsymbol{X}(\boldsymbol{\xi})$. Following [67,71], we start by writing the *co-variant* basis vectors

$$\boldsymbol{a}_i = \frac{\partial \boldsymbol{X}}{\partial \xi^i}\,, \quad i = 1,2,3\,, \quad \boldsymbol{\mathcal{J}} = (\boldsymbol{a}_1,\, \boldsymbol{a}_2,\, \boldsymbol{a}_3)^T\,. \tag{2.14}$$

The inverse of a 3 × 3 matrix is given as

$$\boldsymbol{\mathcal{J}}^{-1} = \frac{1}{J}\begin{pmatrix} (\boldsymbol{a}_2 \times \boldsymbol{a}_3)^T \\ (\boldsymbol{a}_3 \times \boldsymbol{a}_1)^T \\ (\boldsymbol{a}_1 \times \boldsymbol{a}_2)^T \end{pmatrix} := \left(\boldsymbol{a}^1,\, \boldsymbol{a}^2,\, \boldsymbol{a}^3\right)^T\,, \tag{2.15}$$

with the determinant of the Jacobian matrix or the *Jacobian*

$$J = \boldsymbol{a}_1 \cdot (\boldsymbol{a}_2 \times \boldsymbol{a}_3)\,, \tag{2.16}$$

and the *contra-variant* basis vectors $\boldsymbol{a}^i$. From these, the so called metric terms are derived as

$$(J\boldsymbol{a})^i = \boldsymbol{a}_j \times \boldsymbol{a}_k\,, \quad (i,j,k)\,\text{cyclic}\,, \tag{2.17}$$

Thus, a gradient in physical space can be transformed to reference space by

$$\boldsymbol{\nabla}_{\boldsymbol{X}}\, g = \frac{1}{J}\sum_{i=1}^{3}(J\boldsymbol{a})^i \frac{\partial g}{\partial \xi^i} \tag{2.18}$$

and the divergence of a vector $\boldsymbol{g}$ is transformed by

$$\boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{g} = \sum_{n=1}^{3} \frac{\partial g_n}{\partial X_n} = \frac{1}{J}\sum_{i=1}^{3}(J\boldsymbol{a})^i \cdot \frac{\partial \boldsymbol{g}}{\partial \xi^i}\,. \tag{2.19}$$

Following Kopriva [71], this form of the gradient and the divergence are called *non-conservative*, since they are derived directly from the differential form. As shown in [71], the divergence of a flux in a conservation law defined in a infinitesimally small volume ΔV is expressed via the Gauss theorem as an integral over the volume surfaces $\partial\Delta V$ of the flux times the surface normal

$$\boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{g} = \lim_{\Delta V \to 0} \frac{1}{\Delta V} \int\limits_{\partial \Delta V} \boldsymbol{g} \cdot \boldsymbol{n} dS\,, \tag{2.20}$$

and after some manipulations and replacing the discrete by continuous derivatives ($\Delta V \rightarrow 0$), this yields the *conservative form* of the divergence and the gradient

$$\boldsymbol{\nabla}_{\boldsymbol{x}} \cdot \boldsymbol{g} = \frac{1}{J} \sum_{i=1}^{3} \frac{\partial}{\partial \xi^i} \left((J\boldsymbol{a})^i \cdot \boldsymbol{g} \right), \tag{2.21}$$

$$\boldsymbol{\nabla}_{\boldsymbol{x}}\, g = \frac{1}{J} \sum_{i=1}^{3} \frac{\partial}{\partial \xi^i} \left((J\boldsymbol{a})^i g \right). \tag{2.22}$$

If we apply the product rule to the derivatives

$$\boldsymbol{\nabla}_{\boldsymbol{x}} \cdot \boldsymbol{g} = \frac{1}{J} \left\{ \boldsymbol{g} \cdot \sum_{i=1}^{3} \frac{\partial}{\partial \xi^i} (J\boldsymbol{a})^i + \sum_{i=1}^{3} (J\boldsymbol{a})^i \cdot \frac{\partial \boldsymbol{g}}{\partial \xi^i} \right\}, \tag{2.23}$$

$$\boldsymbol{\nabla}_{\boldsymbol{x}}\, g = \frac{1}{J} \left\{ g \sum_{i=1}^{3} \frac{\partial}{\partial \xi^i} (J\boldsymbol{a})^i + \sum_{i=1}^{3} (J\boldsymbol{a})^i \frac{\partial g}{\partial \xi^i} \right\}, \tag{2.24}$$

we observe that the conservative form is equivalent to the non-conservative form if the so-called *metric identities* hold, that is the condition

$$\sum_{i=1}^{3} \frac{\partial (Ja)_n^i}{\partial \xi^i} = 0\,, \quad n = 1, 2, 3\,. \tag{2.25}$$

This property is also often referred as *free-stream preserving*, meaning that the divergence of a constant flux must remain zero under transformation. Satisfying free-stream preservation is not trivial in the discrete three-dimensional case. A solution for the DG-SEM scheme is proposed by Kopriva in [67] and will be introduced in Section 3.3.

Note that the metric terms and the Jacobian are simply products of the covariant vectors. If the mapping is a polynomial of degree N_g, they can be computed directly as derivatives of the mapping. However, the non-linearity of the metric terms is more pronounced with an increasing polynomial degree, since the overall polynomial degree is highly increased by the products. The Jacobian in three dimensions would have a maximum polynomial of degree $(3N_g - 1)$ and $2N_g$ for the metric terms.

In lower dimensions, the increase in polynomial degree is less pronounced. In two dimensions, for example, we find analogously the covariant vectors and the

Jacobian matrix of the mapping

$$a_i^j = \frac{\partial X_j}{\partial \xi^i}\,, \quad i,j=1,2\,, \quad \boldsymbol{\mathcal{J}} = \begin{pmatrix} \boldsymbol{a}_1 \\ \boldsymbol{a}_2 \end{pmatrix}^T, \tag{2.26}$$

with the Jacobian

$$J_{2D} = a_1^1 a_2^2 - a_2^1 a_1^2\,, \tag{2.27}$$

and the metric terms

$$\begin{aligned} (J\boldsymbol{a})_{2D}^1 &= (a_2^2, -a_2^1)^T, \quad (J\boldsymbol{a})_{2D}^2 = (-a_1^2, a_1^1)^T, \\ (J\boldsymbol{a})_{2D}^1 &= (y_\eta, -x_\eta)^T, \quad (J\boldsymbol{a})_{2D}^2 = (-y_\xi, x_\xi)^T. \end{aligned} \tag{2.28}$$

The Jacobian has a maximum degree of $2N_g$ and the metric terms a maximum degree of N_g. Note that the metric identities in two dimensions always hold – given a smooth mapping $\boldsymbol{X}(\boldsymbol{\xi})$ – since the derivatives are exchangeable

$$\frac{\partial}{\partial \xi}\frac{\partial y}{\partial \eta} - \frac{\partial}{\partial \eta}\frac{\partial y}{\partial \xi} = 0\,, \quad -\frac{\partial}{\partial \xi}\frac{\partial x}{\partial \eta} + \frac{\partial}{\partial \eta}\frac{\partial x}{\partial \xi} = 0\,. \tag{2.29}$$

In one dimension, the Jacobian matrix equals the Jacobian, and the derivative is simply transformed by

$$\frac{\partial g}{\partial x} = \frac{1}{J_{1D}}\frac{\partial g}{\partial \xi}\,, \quad J_{1D} = \frac{\partial x}{\partial \xi}\,. \tag{2.30}$$

The one-dimensional Jacobian is a polynomial of degree $(N_g - 1)$.
In the special case of an affine mapping between the physical and reference space, the element mapping remains linear in every coordinate direction. Thus all first derivatives of the mapping are *constant* and accordingly the metric terms and the Jacobian. For non-linear element mappings, the maximum polynomial degree of the element metrics depends not only on the polynomial degree of the mapping, but also on the choice of the basis functions for the specific element type, see Appendix C for a full overview.

2.2. Mapping the Equations

We want to derive the DG scheme on curved elements with a high-order polynomial mapping, since this is the most general and includes the linear element mapping as a special case. We know that we have general Gauss-type integration rules only for reference elements, thus we always need the mapping from the curved physical space to the reference element. In DG, mappings are allowed to be different for each element, under the condition of C^0 continuity at element interfaces. We adopt the idea from spectral element methods [71] to map the *equations* to reference space.

We start from a general system of hyperbolic conservation equations in three-dimensional physical space

$$U_t + F_x(U) + G_y(U) + H_z(U) = U_t + \boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{F}(U) = 0\,, \tag{2.31}$$

where U is the vector of conservative variables $\{U_v\}_{v=1}^{n_{\mathrm{var}}}$, $\{F_v, G_v, H_v\}_{v=1}^{n_{\mathrm{var}}}$ are the vectors of the physical fluxes and $\boldsymbol{F} = (F_1, F_2, F_3)^T = (F, G, H)^T$ is the flux tensor in space $\mathbb{R}^{n_{\mathrm{var}} \times 3}$.

As shown in (2.21), the divergence in physical space is transformed to reference space by

$$\boldsymbol{\nabla}_{\boldsymbol{X}} \cdot \boldsymbol{F} = \frac{1}{J} \sum_{i=1}^{3} \frac{\partial}{\partial \xi^i} \left((J\boldsymbol{a})^i \cdot \boldsymbol{F} \right) := \frac{1}{J} \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{F}}\,, \tag{2.32}$$

yielding the so-called *contravariant fluxes*

$$\mathcal{F}^i = (J\boldsymbol{a})^i \cdot \boldsymbol{F}\,, \qquad i = 1, 2, 3\,. \tag{2.33}$$

Now we formulate (2.31) as *equations in reference space*

$$U_t(t, \boldsymbol{X}(\boldsymbol{\xi})) + \frac{1}{J} \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{F}}(U(t, \boldsymbol{X}(\boldsymbol{\xi}))) = 0\,. \tag{2.34}$$

2.3. Discontinuous Galerkin Formulation

The starting point of the discontinuous Galerkin formulation is a local weak formulation of the problem (2.34). To obtain the weak formulation, we multiply (2.34) by a locally defined test function ϕ and integrate the equation over a single physical element $\mathcal{Q}$

$$\int_{\mathcal{Q}} \left(U_t + \frac{1}{J} \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{F}}(U) \right) \phi \, d\boldsymbol{X} = 0\,. \tag{2.35}$$

This can be seen mathematically as a minimization of the equations' residual in the space spanned by the test function, because the integration of the equation with the test function is a projection onto this test function.

The integral is transformed to reference space E by $\int\limits_{Q}(\dots)\,d\boldsymbol{X}=\int\limits_{E}(\dots)J\,d\boldsymbol{\xi}$, yielding

$$\int\limits_{E}\big(JU_t+\boldsymbol{\nabla}_{\boldsymbol{\xi}}\cdot\boldsymbol{\mathcal{F}}(U)\big)\,\phi\,d\boldsymbol{\xi}=0\,. \tag{2.36}$$

Remember that the solution, the test function as well as the metrics are new functions in reference space, $U=U(\boldsymbol{X}(\boldsymbol{\xi})),\phi=\phi(\boldsymbol{\xi}),J=J(\boldsymbol{\xi})$ and $\boldsymbol{\mathcal{F}}(U)=\boldsymbol{\mathcal{F}}(U,\boldsymbol{X}(\boldsymbol{\xi}))$. Following the Galerkin Ansatz, the number of test functions is chosen to be equal to the number of unknowns of the element.

2.3.1. Weak and Strong Form

We derive the *weak form* by using integration by parts for the flux divergence

$$\int\limits_{E}JU_t\phi\,d\boldsymbol{\xi}-\int\limits_{E}\boldsymbol{\mathcal{F}}(U)\cdot\left(\boldsymbol{\nabla}_{\boldsymbol{\xi}}\phi\right)\,d\boldsymbol{\xi}+\int\limits_{\partial E}(\boldsymbol{\mathcal{F}}\cdot\boldsymbol{N})^{*}\phi(\boldsymbol{\xi})\,d\boldsymbol{\xi}^{\,s}=0\,, \tag{2.37}$$

where $\boldsymbol{N}$ is the outward pointing unit normal vector of the *reference* element faces. At the element surface, the solution is double-valued since discontinuities are allowed. Therefore, we use Riemann solvers to approximate the normal flux on the surface, indicated by the superscript *.

The so-called *strong form* is analytically the same as the weak form, and is found by using integration by parts again for the volume flux term, but with the surface flux only taken from inside, marked with the superscript $(\cdot)^{\mathrm{int}}$

$$-\int\limits_{E}\boldsymbol{\mathcal{F}}(U)\cdot\left(\boldsymbol{\nabla}_{\boldsymbol{\xi}}\phi\right)\,d\boldsymbol{\xi}=\int\limits_{E}\boldsymbol{\nabla}_{\boldsymbol{\xi}}\cdot\boldsymbol{\mathcal{F}}(U)\phi\,d\boldsymbol{\xi}-\int\limits_{\partial E}\left(\boldsymbol{\mathcal{F}}^{\mathrm{int}}\cdot\boldsymbol{N}\right)\phi\,d\boldsymbol{\xi}^{\,s}\,, \tag{2.38}$$

inserting in (2.37) yields the strong form

$$\int\limits_{E}\big(JU_t+\boldsymbol{\nabla}_{\boldsymbol{\xi}}\cdot\boldsymbol{\mathcal{F}}(U)\big)\,\phi\,d\boldsymbol{\xi}+\int\limits_{\partial E}\Big((\boldsymbol{\mathcal{F}}\cdot\boldsymbol{N})^{*}-\left(\boldsymbol{\mathcal{F}}^{\mathrm{int}}\cdot\boldsymbol{N}\right)\Big)\,\phi\,d\boldsymbol{\xi}^{\,s}=0\,. \tag{2.39}$$

The first term in the strong form equals the local residual of the equation, and the second term is called the penalty term, being proportional to the jump of the solution at the element interfaces. The penalty term vanishes when the solution is continuous at the element interfaces.

Being analytically the same, the strong and weak form can be different depending on the discretization of the DG scheme. For example, the weak form guarantees exact local conservation by construction, since for $\phi = 1$, equation (2.37) becomes

$$\int\limits_E JU_t \, d\boldsymbol{\xi} = - \int\limits_{\partial E} (\boldsymbol{\mathcal{F}} \cdot \boldsymbol{N})^* \, d\boldsymbol{\xi}^{\,s} \,, \tag{2.40}$$

In contrast, the strong form is only exactly conservative if equation (2.38) holds for the test function $\phi = 1$,

$$\int\limits_E \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{F}}(U) \, d\boldsymbol{\xi} - \int\limits_{\partial E} \left(\boldsymbol{\mathcal{F}}^{\mathrm{int}} \cdot \boldsymbol{N} \right) \, d\boldsymbol{\xi}^{\,s} = 0 \,, \tag{2.41}$$

thus if volume fluxes and surface fluxes cancel out exactly. Especially for non-linear flux functions and non-linear element mappings, discretization errors would be different for both terms. In the case of a DG-SEM discretization, which will be described in detail in Section 3.3, it was shown by Kopriva and Gassner [68] that the weak and strong form are equivalent on a discrete level, yielding to exact conservation of the discretized strong form.

Surface Normal

We have a closer look at the flux term in the surface integral term in (2.37)

$$\begin{aligned} \boldsymbol{\mathcal{F}} \cdot \boldsymbol{N} &= \sum_{i=1}^{3} \mathcal{F}^i N^i = \sum_{i=1}^{3} \sum_{d=1}^{3} J a_d^i F_d N^i \\ &= \sum_{d=1}^{3} \left(\sum_{i=1}^{3} J a_d^i N^i \right) F_d := \sum_{d=1}^{3} F_d \tilde{n}_d = (\boldsymbol{F} \cdot \boldsymbol{n}) \, |\tilde{\boldsymbol{n}}| \,, \end{aligned} \tag{2.42}$$

where we see the equivalence in physical space using the outward pointing normal vector in physical space $\boldsymbol{n}$. It is defined by the metric terms evaluated at the surface.

The *non-normalized* normal vector components in physical space are computed as

$$\tilde{n}_d(\boldsymbol{X}(\boldsymbol{\xi}^{\,s})) = \sum_{i=1}^{3} J a_d^i(\boldsymbol{\xi}^{\,s}) N^i \,, \quad d = 1, 2, 3 \,, \tag{2.43}$$

leading to the surface element and the unit normal in physical space

$$\Rightarrow \quad \hat{s} = |\tilde{\boldsymbol{n}}| = \sqrt{\sum_{d=1}^{3} (\tilde{n}_d)^2} \quad \Rightarrow \quad n_d = \frac{\tilde{n}_d}{\hat{s}} \,. \tag{2.44}$$

As an example, if the unit normal in reference space would be $\boldsymbol{N} = (1,0,0)^T$, the normal vector in physical space is $\boldsymbol{n} = (J\boldsymbol{a})^1/|(J\boldsymbol{a})^1|$.

Weak and Strong Formulation

Finally, using the physical normal vectors and the numerical flux in the normal direction $G^* = G^*(U^L, U^R, \boldsymbol{n}) = (\boldsymbol{F}\cdot\boldsymbol{n})^*$, with the left and right state U^L, U^R, the weak form reads as

$$\int\limits_E JU_t\phi \, d\boldsymbol{\xi} - \int\limits_E \boldsymbol{\mathcal{F}}(U)\cdot\left(\nabla_{\boldsymbol{\xi}}\phi\right)\, d\boldsymbol{\xi} + \int\limits_{\partial E} G^*\,\hat{s}\;\phi(\boldsymbol{\xi})\, d\boldsymbol{\xi}^{\,s} = 0\,, \tag{2.45}$$

and the strong form as

$$\int\limits_E JU_t\phi \, d\boldsymbol{\xi} + \int\limits_E \nabla_{\boldsymbol{\xi}}\cdot\boldsymbol{\mathcal{F}}(U)\phi \, d\boldsymbol{\xi} + \int\limits_{\partial E} \left(G^* - \left(\boldsymbol{F}^{\text{int}}\cdot\boldsymbol{n}\right)\right)\hat{s}\;\phi(\boldsymbol{\xi})\, d\boldsymbol{\xi}^{\,s} = 0\,. \tag{2.46}$$

2.3.2. Second Order Equations

For second order terms, as for example found in the Poisson equation or the Navier-Stokes equations, the flux includes the gradient of the solution

$$U_t + \nabla_{\boldsymbol{x}}\cdot\boldsymbol{F}(U,\nabla_{\boldsymbol{x}}U) = 0\,, \quad \boldsymbol{F}(U,\nabla_{\boldsymbol{x}}U) = \boldsymbol{F}^a(U) + \boldsymbol{F}^v(U,\nabla_{\boldsymbol{x}}U)\,. \tag{2.47}$$

The flux can be separated into purely hyperbolic fluxes $\boldsymbol{F}^a$ and viscous fluxes $\boldsymbol{F}^v$. It was shown by Bassi and Rebay [6] that a discontinuous approximation of a purely elliptic problem is unstable if only local polynomial derivatives inside the elements are used. To stabilize the scheme, a so-called *lifting* of the gradients is needed. Many different lifting methods exist for the DG scheme, the most prominent candidates are BR1 and BR2 [6, 10], local DG [34] and compact DG [83] as well as interior penalty methods [3, 4]. In [48, 76], it was shown that an optimal stable jump penalization parameter can be derived from a generalized Riemann problem, locally defined at the grid cell interface.

Most lifting methods introduce additional equations for the gradient of the solution in each space direction d

$$Q^d = \frac{\partial U}{\partial X_d} \quad\Leftrightarrow\quad Q^d - \frac{\partial}{\partial X_d}U = 0\,,\;\; d = 1,2,3\,. \tag{2.48}$$

Again, we transform the equations to reference space, using the conservative mapping of the gradient (2.22)

$$Q^d - \frac{1}{J} \sum_{i=1}^{3} \frac{\partial}{\partial \xi^i} \left((Ja)^i_d U \right) = 0\,, \tag{2.49}$$

or

$$Q^d - \frac{1}{J} \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{U}}^d = 0\,, \quad \boldsymbol{\mathcal{U}}^d = \left((Ja)^1_d, (Ja)^2_d, (Ja)^3_d \right)^T U\,. \tag{2.50}$$

The equations are discretized with the same numerical scheme as the conservation equations, in our case the DG formulation. Analogously to Section 2.3, we get the weak DG formulation of the gradient equations as

$$\int\limits_E JQ^d \phi(\boldsymbol{\xi})\, d\boldsymbol{\xi} + \int\limits_E \boldsymbol{\mathcal{U}}^d \cdot \left(\boldsymbol{\nabla}_{\boldsymbol{\xi}} \phi(\boldsymbol{\xi}) \right)\, d\boldsymbol{\xi} - \int\limits_{\partial E} \left(\boldsymbol{\mathcal{U}}^d \cdot \boldsymbol{N} \right)^* \phi(\boldsymbol{\xi})\, d\boldsymbol{\xi}^{\,s} = 0\,, \tag{2.51}$$

where the surface flux can be rewritten with the definition of physical normal vector (2.44) as

$$\left(\boldsymbol{\mathcal{U}}^d \cdot \boldsymbol{N} \right) = (U n_d)\, \hat{s}\,. \tag{2.52}$$

The solution at the surface is again double-valued and the flux will be approximated by a numerical flux $(U n_d)^* = U^* n_d$, which depends on the lifting method. The weak form reads as

$$\int\limits_E JQ^d \phi(\boldsymbol{\xi})\, d\boldsymbol{\xi} + \int\limits_E \boldsymbol{\mathcal{U}}^d \cdot \left(\boldsymbol{\nabla}_{\boldsymbol{\xi}} \phi(\boldsymbol{\xi}) \right)\, d\boldsymbol{\xi} - \int\limits_{\partial E} U^* n_d \hat{s} \phi(\boldsymbol{\xi})\, d\boldsymbol{\xi}^{\,s} = 0\,, \tag{2.53}$$

and the strong form as

$$\int\limits_E \left(JQ^d - \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{U}}^d \right) \phi(\boldsymbol{\xi})\, d\boldsymbol{\xi} - \int\limits_{\partial E} (U^* - U^{\text{int}}) n_d \hat{s} \phi(\boldsymbol{\xi})\, d\boldsymbol{\xi}^{\,s} = 0\,, \tag{2.54}$$

where we recover the residual of the gradient equation in the volume integral, and the surface integral is referred to as the *lifting term*, which is proportional to the jumps at the cell interfaces and vanishes for a continuous solution at the interface.

3. Discretization

The fundamental constraints of the DG scheme are that all quantities are approximated by high order polynomials and that the unknowns are determined by the DG formulation of Section 2.3. From this starting point, there are many choices to be made for the discretization and implementation, each leading to a different variant of the DG scheme.

- **Choice of the element shape:**
  Due to its local element-wise operator structure and the coupling over element faces only, the DG method is well suited for unstructured meshes and all element shapes. However, each element shape requires different polynomial basis functions and integration rules. Arbitrary element shapes can be used as long as integration rules for the element volume and its faces are given. In [49] even polygons were used. General element types are triangles and quadrilaterals in 2D and tetrahedra, prisms, pyramids and hexahedra in 3D, for which standard integration rules exist. (See Appendix B)

- **Choice of the basis function:**
  The solution inside an element is approximated by a polynomial. However, the same polynomial can be represented in different ways by choosing different basis functions. We can classify basis functions into modal or nodal. The coefficients of modal basis functions are hierarchical. For example, the mean value is the first coefficient when using an orthogonal basis function. Nodal basis functions, also called Lagrange basis functions, represent interpolation polynomials. The Lagrange basis functions are only defined by the set of interpolation points and the coefficients are the solution values at the interpolation point position. We discuss the different basis functions for different element types in detail in Section 3.1. Nodal basis functions are very attractive from an implementation point of view, since the coefficients are directly solution values, and one can choose sub-sets of points for cheap interpolation, e.g. 2D face nodes for the interpolation at element faces.

- **Number of integration points:**
  In the implementation of the DG scheme, the volume and surface integrals of the DG formulation are replaced by numerical integration rules. In general, Gauss-Type integration rules are used, where Gauss point positions and weights differ between element types. In 1D, n_{GP} Gauss points would integrate a polynomial of degree $(2n_{GP} - 1)$ exactly. If the integrand is a product of polynomials, an exact integration is possible, but not always feasible.

  The appropriate number of integration points depends on the non-linearity of the integrand, thus the polynomial degree of the solution, of the element mapping (see Section 2.1) and of the flux function.

- **Approximation of the flux function:**
  For linear equations, e.g. Maxwell equations, the polynomial degree of the co-variant flux function equals the degree of the solution. When using a nodal basis and a linear element mapping, the flux function would be exactly represented by interpolation. Thus the flux is evaluated using directly the solution values at the interpolation points and integrals can be evaluated exactly. However, non-linear contra-variant flux functions, either from a non-linear element mapping or from non-linear equation systems, e.g. of the compressible Navier-Stokes equations, a higher number of integration points is needed for the flux integrals to minimize aliasing errors.

In the following sections, we will describe two choices for the design of a DG method:

- The first DG scheme will be designed for hybrid meshes. The hybrid DG scheme will use nodal basis functions, where element face interpolation points are reused for volume interpolation points, and a Gauss-type integration with the number of integration points depending on the element shape and flux function.We will highlight the difference between the integration and interpolation of the flux.

- The second variant is an optimized DG scheme called Discontinuous Galerkin Spectral Element Method (DG-SEM), introduced by Black [15] and described in detail by Kopriva [71], which is restricted to hexahedra (and quadrangles in 2D). It has tensor-product nodal basis functions either defined by a one-dimensional set of Gauss-Legendre or Legendre-Gauss-Lobatto nodes, where the solution and flux interpolation points are chosen to be equal to the integration nodes. This choice is often referred to as ‘collocation’.

3.1. Polynomial Approximation

The computational domain Ω is discretized by a set of n_e non-overlapping elements $\mathcal{Q}_i$, such that

$$\Omega_h = \bigcup_{i=1}^{n_e} \mathcal{Q}_i \,, \quad \mathcal{Q}_i \cap \mathcal{Q}_j = \emptyset \quad \text{if} \quad i \neq j \,, 1 \leq i, j \leq n_e \,. \tag{3.1}$$

Corresponding to the element type, each element $\mathcal{Q}_i$ is associated to its reference element E by the element mapping introduced in Section 2.1

$$\begin{aligned} \boldsymbol{X} : E &\mapsto \mathcal{Q}_i \\ \boldsymbol{\xi} &\mapsto \boldsymbol{X}(\boldsymbol{\xi}) \,. \end{aligned} \tag{3.2}$$

The different reference elements are shown in Fig. 3.1 and the number of interpolation nodes is listed in Table 3.1.

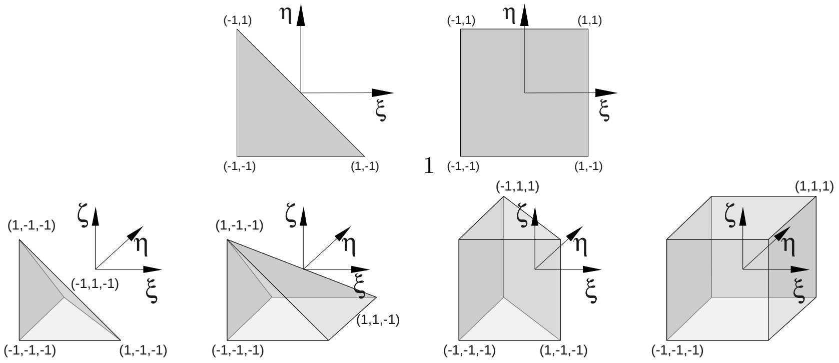


Figure 3.1.: Reference elements for triangles, quadrilaterals, tetrahedra, pyramids, prisms and hexahedra

Inside each element, the solution is approximated as a polynomial in the local reference space. The simplest polynomial is an interpolation polynomial, being

Triangle	$\frac{1}{2}(N+1)(N+2)$	Pyramid	$\frac{1}{6}(N+1)(N+2)(2N+3)$
Quadrilateral	$(N+1)^2$	Prism	$\frac{1}{2}(N+1)^2(N+2)$
Tetrahedra	$\frac{1}{6}(N+1)(N+2)(N+3)$	Hexahedra	$(N+1)^3$

Table 3.1.: Number of interpolation points n_{IP} for six standard element types.

defined by an evaluation of the solution at discrete points $\left\{\hat{\boldsymbol{\xi}}_m\right\}_{m=1}^{n_{\mathrm{IP}}}$. The corresponding points in physical space are found by evaluating the volume mapping

$$\hat{\boldsymbol{x}}_m = X(\hat{\boldsymbol{\xi}}_m)\,. \tag{3.3}$$

The number of interpolation points depend on the polynomial degree N, the type of basis and the element type. The number of interpolation points should be equal or greater than the number of basis functions M. For a complete order basis in 3D, the number of interpolation is

$$n_{\mathrm{IP}} \geq M = (N+1)(N+2)(N+3)/6\,, \tag{3.4}$$

and for a tensor product basis

$$n_{\mathrm{IP}} \geq M = (N+1)^3\,. \tag{3.5}$$

We write the polynomial in reference space using three-dimensional Lagrange basis functions $\psi_j(\boldsymbol{\xi})$, which are defined by the Lagrange property

$$\psi_\ell(\hat{\boldsymbol{\xi}}_m) = \delta_{\ell m} = \left\{ \begin{array}{ll} 1\,, & \text{if} \quad \ell = m \\ 0\,, & \text{if} \quad \ell \neq m \end{array} \right. , \tag{3.6}$$

where $\hat{\boldsymbol{\xi}}_m$ belongs to a set of nodes $\left\{\hat{\boldsymbol{\xi}}_m\right\}_{m=1}^{n_{\mathrm{IP}}}$.
The solution, for example, is then approximated by

$$U(\boldsymbol{X}(\boldsymbol{\xi})) \approx U(\boldsymbol{\xi}) = \sum_{\ell=1}^{n_{\mathrm{IP}}} \hat{U}_\ell \psi_\ell(\boldsymbol{\xi})\,, \qquad \hat{U}_\ell = U(\boldsymbol{X}(\hat{\boldsymbol{\xi}}_\ell))\,. \tag{3.7}$$

The Lagrange basis functions, also referred as *nodal basis functions*, do not have a closed form representation in multiple dimensions. The only exception are basis functions for quadrilaterals and hexahedra.

For quadrilaterals and hexahedra, we use a set of one-dimensional points and its associated one-dimensional Lagrange basis $\ell_i(\xi)$ from (2.7). We extend the interpolation nodes as well as the basis to multiple dimensions by applying the tensor-product

$$\begin{gathered}\hat{\boldsymbol{\xi}}_l = (\hat{\xi}_i, \hat{\xi}_j, \hat{\xi}_k)^T\,, \quad \psi_l(\boldsymbol{\xi}) = \ell_i(\xi)\ell_j(\eta)\ell_k(\zeta)\,, \\ i,j,k = 0,\dots,N\,, \quad l = 1,\dots,(N+1)^3\,,\end{gathered} \tag{3.8}$$

and with an index mapping $l \mapsto (i,j,k)$ between the three-dimensional and the one-dimensional Lagrange basis.
We can also express the solution polynomial with a set of M orthonormal basis functions $\varphi_j(\boldsymbol{\xi})$, which are known analytically,

$$U(\boldsymbol{X}(\boldsymbol{\xi})) \approx U(\boldsymbol{\xi}) = \sum_{j=1}^{M} \tilde{U}_j \varphi_j(\boldsymbol{\xi})\,. \tag{3.9}$$

The basis functions are called orthonormal if the inner product with respect to the reference element space yields

$$\int\limits_E \varphi_i(\boldsymbol{\xi})\varphi_j(\boldsymbol{\xi})\,d\boldsymbol{\xi} = \delta_{ij}\,, \quad i,j = 1,\dots,M\,. \tag{3.10}$$

The Lagrange basis functions for other element types are found by expressing the interpolation polynomial with the known orthonormal basis functions

$$U(\boldsymbol{\xi}) = \sum_{l=1}^{n_{\mathrm{IP}}} \hat{U}_l \psi_l(\boldsymbol{\xi}) = \sum_{j=1}^{M} \tilde{U}_j \varphi_j(\boldsymbol{\xi})\,. \tag{3.11}$$

We find the *modal* coefficients of the solution $\tilde{U}_j$ by evaluating (3.11) at each interpolation point.

$$U(\hat{\boldsymbol{\xi}}_i) = \hat{U}_i = \sum_{j=1}^{M} \tilde{U}_j \varphi_j(\hat{\boldsymbol{\xi}}_i) = \sum_{j=1}^{M} V_{ij}\,\tilde{U}_j\,, \quad i = 1,\dots,n_{\mathrm{IP}}\,, \tag{3.12}$$

with the Vandermonde matrix defined as

$$V_{ij} = \varphi_j(\hat{\boldsymbol{\xi}}_i)\,, \quad i = 1,\dots,n_{\mathrm{IP}}\,, \quad j = 1,\dots,M\,. \tag{3.13}$$

The polynomials are equal if the same number of basis functions is used, i.e. $n_{\mathrm{IP}} = M$, and the Vandermonde matrix is invertible. The orthonormal basis

functions satisfying this condition are described in Appendix A for six standard element types.
In matrix-vector notation, we write the evaluation of the vector of Lagrange basis functions $\underline{\psi}(\boldsymbol{\xi}) = (\psi_1(\boldsymbol{\xi}), \ldots, \psi_M(\boldsymbol{\xi}))^T$ and its derivative as

$$\underline{\psi}(\boldsymbol{\xi}) = \underline{\underline{\boldsymbol{V}}}^{-T} \underline{\varphi}(\boldsymbol{\xi}) \,, \quad \frac{\partial \underline{\psi}(\boldsymbol{\xi})}{\partial \xi^i} = \underline{\underline{\boldsymbol{V}}}^{-T} \frac{\partial \underline{\varphi}(\boldsymbol{\xi})}{\partial \xi^i} \,. \tag{3.14}$$

We want to stress that the Vandermonde matrix has to be inverted to evaluate the Lagrange basis functions via the known modal basis functions. Ralston and Rabinowitz show in [85] that the round-off errors of a numerical matrix inversion using Gauss elimination is related to condition number of the matrix, being the ratio of the largest to the smallest eigenvalue. The larger the condition number, the less accurate is the numerical matrix inverse. The condition number of the Vandermonde matrix strongly depends on the choice of the polynomial basis functions and the interpolation node positions. A common measure for the interpolation quality of a point set is the Lebesque constant Λ

$$\Lambda = \max_{\boldsymbol{\xi} \in E} \sum_{l=1}^{n_{\mathrm{IP}}} |\psi_l(\boldsymbol{\xi})| \,. \tag{3.15}$$

The points with the lowest Lebesgue constant known up to now are the so-called Fekete points. In 1D, they coincide with the Legendre-Gauss-Lobatto (LGL) [44,58]. Tensor-product LGL nodes on quadrilaterals and hexahedra are Fekete points, too [22]. Fekete points exist for triangles and tetrahedrons, but their point positions cannot be expressed as closed formulas for arbitrary polynomial degrees. The nodal sets proposed in the book of Hesthaven and Warburton [58] have Lebesgue constants similar to optimized electrostatic nodal sets and closed formulas and the algorithm for triangles and tetrahedra and arbitrary polynomial degrees are given. A recursive construction technique for optimized nodal sets that can be applied to general element types and arbitrary polynomial degrees was introduced by Gassner et al. [49].
A comparison of the condition number of the Vandermonde matrix (3.13) and the Lebesgue constant for the six standard element types and $N = 1, \ldots, 8$ is shown in Fig. 3.2 and in detail for $N = 8$ in Table 3.2. The simple monomial basis functions $m_{ijk}(\boldsymbol{\xi}) = \xi^i \eta^j \zeta^k$ are compared to orthonormal basis functions on a uniformly spaced nodal set, and again compared to an optimized nodal set. We used optimized nodal sets by Hesthaven and Warburton [58] for triangles, tetrahedra and prisms, an optimized nodal set for the pyramid by Gassner et al. [49], and tensor-product LGL points for quadrilaterals and hexahedra.

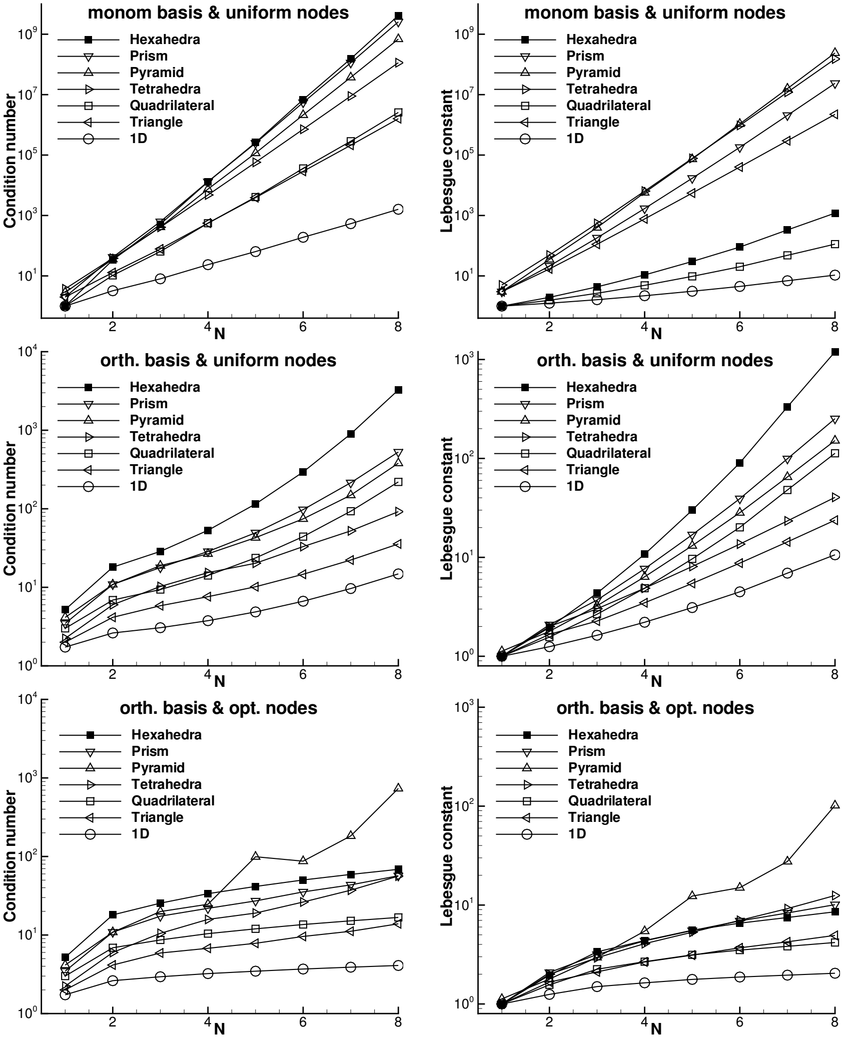


Figure 3.2.: Condition number of the Vandermonde matrix (left) and Lebesgue constant (right) for six standard element types and $N = 1, \ldots, 8$. Three choices of basis functions and interpolation nodes, from top to down: monomial basis and uniformly spaced nodes, orthogonal basis and uniformly spaced nodes and orthogonal basis and optimized nodes.

basis:		monomial	orthonormal	orthonormal	
nodal set:		uniform	uniform	opt. nodes (type)	
1D	$C =$	1.60E+03	15	4.1	(LGL)
	$\Lambda =$	10.6	10.6	2.04	
Quad.	$C =$	2.60E+06	220	16.7	(LGL tensor)
	$\Lambda =$	112	112	4.18	
Hexa	$C =$	4.00E+09	3200	68.4	(LGL tensor)
	$\Lambda =$	1200	1200	8.55	
Triangle	$C =$	1.60E+03	36	14.0	(Hest.& Warb. [58])
	$\Lambda =$	(2.23E+06)	23.8	4.94	
Tetrahedra	$C =$	1.10E+08	92	56.0	(Hest.& Warb. [58])
	$\Lambda =$	(1.51E+08)	40.4	12.5	
Pyramid	$C =$	6.90E+08	380	734	(Gassner et al. [49])
	$\Lambda =$	(2.39E+08)	158	105	
Prism	$C =$	2.60E+09	528	57.0	(Hest.& Warb. + LGL)
	$\Lambda =$	(2.37E+07)	252	10.1	

Table 3.2.: Condition number, C, of the Vandermonde matrix and Lebesque constant, Λ, for $N = 8$ for six standard element types and three choices of basis functions and nodal sets.

The choice of monomial basis functions and a uniformly spaced nodal set leads to very high condition numbers which grow very quickly with the polynomial degree. Note the large scale of the y-axis of 10^{10} for the monomial basis in Fig. 3.2. The best condition numbers are always achieved with an optimized nodal set and orthonormal basis functions, whereas uniformly spaced nodes should be used only for low polynomial degrees. In addition, the growth towards higher polynomial degrees is smaller for the optimized nodes. An exception is the pyramid, even though the optimized nodes reduce the Lebesgue constant, the condition number grows similarly to the uniformly spaced nodal set.

In theory, the Lebesgue constant should only depend on the node position, but as shown in Table 3.2,the Lebesque constant blows up for the uniformly spaced node set if monomial basis functions are used. The error is introduced by the evaluation the Lagrange basis via the numerically inverted Vandermonde matrix. The numerical errors are growing with the condition number of the

matrix. An exception are tensor-product elements like quadrilaterals and hexahedra, where the multidimensional inverse is very sparse and no influence of the basis evaluation for the Lebesque constant is observed for the uniformly spaced nodal set. The growth of the condition number over the polynomial degree is reduced when using orthogonal basis functions. Together with an optimized nodal set, the growth of the condition number can be further reduced and Lebesgue constant are improved, yielding the best choice for high polynomial degrees.
For quadrilaterals and hexahedra, the multidimensional Lagrange basis can be evaluated directly from the one-dimensional Lagrange basis and the tensor-product structure. Therefore, an interpolation to another set of points can be done dimension-by-dimension and *does not involve the inverse* of the Vandermonde matrix. If the modal coefficients of a tensor-product Lagrange basis must be computed, a numerical matrix inversion can still be circumvented. In one dimension, a closed form of the Lagrange basis exists, and therefore the inverse of the Vandermonde can be evaluated exactly by projection

$$\begin{aligned} \ell_j(\xi) &= \sum_{k=0}^{N} \left(V_{1\mathrm{D}}^{-T}\right)_{jk} \varphi_k(\xi)\,, \quad j,k=0,\dots,N\,, \\ \int\limits_{-1}^{1} \ell_j(\xi)\varphi_i(\xi)\,d\xi &= \sum_{k=0}^{N} \left(V_{1\mathrm{D}}^{-1}\right)_{kj} \underbrace{\int\limits_{-1}^{1} \varphi_k(\xi)\varphi_i(\xi)\,d\xi}_{\delta_{ik}}\,, \\ \Rightarrow \quad \left(V_{1\mathrm{D}}^{-1}\right)_{ij} &= \int\limits_{-1}^{1} \ell_j(\xi)\varphi_i(\xi)\,d\xi\,, \end{aligned} \tag{3.16}$$

where the integral is exactly evaluated with $(N+1)$ Gauss points. Hence, we are able to transform a tensor-product Lagrange basis into a Legendre tensor-product basis by applying the exact inverse of the Vandermonde matrix (3.16) in a dimension-by-dimension fashion. This avoids numerical errors being introduced by matrix inversion.

3.2. Nodal DG Scheme for General Elements

In this section, we derive the discrete DG scheme for hybrid meshes, using nodal basis functions and Gauss-type numerical integration. We present two ways to treat the metrics of the element.

We start with the weak formulation of the hyperbolic conservation equation for the element in reference space, (2.45), and use the nodal basis function $\phi = \psi_\ell$ as test function

$$\int_E J U_t \psi_\ell \, d\boldsymbol{\xi} - \int_E \sum_{i=1}^{3} \left(\sum_{n=1}^{3} (Ja)^i_n F_n(U) \right) \frac{\partial \psi_\ell}{\partial \xi^i} \, d\boldsymbol{\xi} + \int_{\partial E} G^* \hat{s} \psi_\ell(\boldsymbol{\xi}) \, d\boldsymbol{\xi}^{\,s} = 0 \,. \tag{3.17}$$

We replace the integrals by numerical Gauss-type quadrature rules (see Appendix B for standard element types), insert the polynomial approximation of the solution (3.7) and write the semi-discrete form of the equation in matrix-vector notation

$$\begin{aligned} \underline{\underline{\tilde{\boldsymbol{M}}}}\, \underline{\hat{U}}_t &= \sum_{n=1}^{3} \underline{\underline{\tilde{\boldsymbol{S}}}}^{\,n} \underline{F}_n - \sum_{s=1}^{n_{\text{sides}}} \underline{\underline{\tilde{\boldsymbol{L}}}}^{\,s} \underline{G^*}, \\ \underline{\hat{U}}_t &= \sum_{n=1}^{3} \left(\underline{\underline{\tilde{\boldsymbol{M}}}}^{\,-1} \underline{\underline{\tilde{\boldsymbol{S}}}}^{\,n} \right) \underline{F}_n - \sum_{s=1}^{n_{\text{sides}}} \left(\underline{\underline{\tilde{\boldsymbol{M}}}}^{\,-1} \underline{\underline{\tilde{\boldsymbol{L}}}}^{\,s} \right) \underline{G^*}. \end{aligned} \tag{3.18}$$

with the vector of the solution values at the interpolation nodes and the vectors of the flux function as well as the numerical flux evaluated at Gauss points

$$\begin{aligned} &\underline{\hat{U}} = \left(\hat{U}_1, \ldots, \hat{U}_{n_{\text{IP}}} \right)^T, \qquad \underline{F} = \left(F(U(\boldsymbol{\chi}_1)), \ldots, F(U(\boldsymbol{\chi}_{n_{\text{GP}}})) \right)^T, \\ &\underline{G^*} = \left(\left. G^*\left(U^L, U^R, \boldsymbol{n}\right) \right|_{\boldsymbol{\chi}^s_1}, \ldots, \left. G^*\left(U^L, U^R, \boldsymbol{n}\right) \right|_{\boldsymbol{\chi}^s_{n^s_{\text{GP}}}} \right)^T, \end{aligned} \tag{3.19}$$

and where $\underline{\underline{\tilde{\boldsymbol{M}}}}$ is the mass matrix, $\underline{\underline{\tilde{\boldsymbol{S}}}}^{\,n}$ the stiffness matrix and $\underline{\underline{\tilde{\boldsymbol{L}}}}^{\,s}$ the lifting matrix for each element side. Note that in this formulation, the discrete operators $\underline{\underline{\tilde{\boldsymbol{M}}}}, \underline{\underline{\tilde{\boldsymbol{S}}}}^{\,n}, \underline{\underline{\tilde{\boldsymbol{L}}}}^{\,s}$ include the element metrics. Therefore, they need to be stored for each element separately.

$$\tilde{M}_{\ell j} = \sum_{k=1}^{n_{\text{GP}}} \omega_k \left[J(\boldsymbol{\xi}) \psi_j(\boldsymbol{\xi}) \psi_\ell(\boldsymbol{\xi}) \right]_{\boldsymbol{\xi} = \boldsymbol{\chi}_k}, \qquad 1 \le \ell \le n_{\text{IP}}, 1 \le j \le n_{\text{IP}}, \tag{3.20}$$

$$(\tilde{S}_{\ell k})^n = \omega_k \sum_{i=1}^{3} \left[(Ja)^i_n(\boldsymbol{\xi}) \frac{\partial \psi_\ell(\boldsymbol{\xi})}{\partial \xi^i} \right]_{\boldsymbol{\xi} = \boldsymbol{\chi}_k}, \quad 1 \le \ell \le n_{\text{IP}}, 1 \le k \le n_{\text{GP}}, \tag{3.21}$$

$$(\tilde{L}_{\ell k})^s = \omega_k \left[\hat{s}^s(\boldsymbol{\xi}) \psi_\ell(\boldsymbol{\xi}) \right]_{\boldsymbol{\xi} = \boldsymbol{\chi}^s_k}, \qquad 1 \le \ell \le n_{\text{IP}}, 1 \le k \le n^s_{\text{GP}}. \tag{3.22}$$

This discretization was also proposed by Hesthaven and Warburton [58] for of curved elements.

In contrast, if we do not include the metrics in the matrices, only one set of operators is needed per element type (tetrahedra, pyramids, prisms or hexahedra). This yields a second variant of the semi-discrete form

$$
\begin{aligned}
\underline{\underline{\boldsymbol{M}}}\left(\underline{J}\,\underline{\hat{U}}_t\right) &= \sum_{i=1}^{3} \underline{\underline{\boldsymbol{S}}}^{\,i}\underline{\mathcal{F}}^i - \sum_{s=1}^{n_{\text{sides}}} \underline{\underline{\boldsymbol{L}}}^{\,s}\left(\underline{G}^*\underline{\hat{s}}\right)\,, \quad \underline{\mathcal{F}}^i = \sum_{n=1}^{3} \underline{(Ja)_n^i}\,\underline{F}_n \\
\underline{J}\,\underline{\hat{U}}_t &= \sum_{i=1}^{3}\left(\underline{\underline{\boldsymbol{M}}}^{-1}\underline{\underline{\boldsymbol{S}}}^{\,i}\right)\underline{\mathcal{F}}^i - \sum_{s=1}^{n_{\text{sides}}}\left(\underline{\underline{\boldsymbol{M}}}^{-1}\underline{\underline{\boldsymbol{L}}}^{\,s}\right)\left(\underline{G}^*\underline{\hat{s}}^s\right)\,.
\end{aligned}
\tag{3.23}
$$

Here, the Jacobian is evaluated only at the interpolation points, and the volume and surface metrics are evaluated at the Gauss points. The discrete operators read as

$$
M_{\ell j} = \sum_{k=1}^{n_{\text{GP}}} \omega_k \left[\psi_j(\boldsymbol{\xi})\psi_\ell(\boldsymbol{\xi})\right]_{\boldsymbol{\xi}=\boldsymbol{\chi}_k}\,, \qquad 1 \le \ell \le n_{\text{IP}}\,, 1 \le j \le n_{\text{IP}}\,, \tag{3.24}
$$

$$
\left(S_{\ell k}\right)^i = \omega_k \sum_{i=1}^{3}\left[\frac{\partial \psi_\ell(\boldsymbol{\xi})}{\partial \xi^i}\right]_{\boldsymbol{\xi}=\boldsymbol{\chi}_k}\,, \qquad 1 \le \ell \le n_{\text{IP}}\,, 1 \le k \le n_{\text{GP}}\,, \tag{3.25}
$$

$$
\left(L_{\ell k}\right)^s = \omega_k \left[\psi_\ell(\boldsymbol{\xi})\right]_{\boldsymbol{\xi}=\boldsymbol{\chi}_k^s}\,, \qquad 1 \le \ell \le n_{\text{IP}}\,, 1 \le k \le n_{\text{GP}}^s\,. \tag{3.26}
$$

Assuming the same Gauss point sets, the discrete volume and surface integral of both formulations are equal, with the relationship between the matrices

$$
\tilde{S}_{\ell k}^n = \sum_{i=1}^{3} (Ja)_n^i(\boldsymbol{\chi}_k) S_{\ell k}\,, \quad \tilde{L}_{\ell k} = \hat{s}^s(\boldsymbol{\chi}_k^s) L_{\ell k}\,. \tag{3.27}
$$

The difference lies in the mass matrix, which is either integrated including the Jacobian (using enough number of Gauss points) or 'mass-lumped', since the Jacobian is interpolated at the solution points which leads to an approximation of the exact mass matrix. For a linear element mapping with $J = \text{const.}$, both formulations have the same discrete solution, since $\underline{\underline{\tilde{\boldsymbol{M}}}} = J\underline{\underline{\boldsymbol{M}}}$ holds for a constant Jacobian.

The two formulations have both advantages and disadvantages. Clearly, for all elements with linear element mappings, the formulations are equal and one would use the mass-lumped formulation (3.23), having the same computational effort because of the constant metrics and less memory consumption. For elements with non-parallel straight edges and curved elements, the element mapping is non-linear and the two formulations differ. The formulation with the exact mass matrix has slightly fewer operations (since the metrics are included into the operators) but much more memory consumption than the mass-lumped formulation. On the other hand, the mass-lumped formulation would introduce an discretization error due to the approximation of the mass matrix. The collocation of solution and Jacobian also introduces aliasing errors. The errors clearly depend both on the polynomial degree N of the DG solution and the polynomial degree N_g of the element mapping, but also on the element type and the choice of interpolation nodes. Regarding the trend of modern computing hardware, the floating-point operation per second (FLOPS) are growing much faster than the memory bandwidth, so that a low memory consumption becomes more and more important.

3.2.1. Nodal DG with Flux Interpolation

A further simplification of the nodal DG scheme is to interpolate the fluxes. As the physical values of the approximate solution are already available at the interpolation points, it seems attractive to evaluate the flux function and the numerical flux at the interpolation points

$$F_n(U) \approx \sum_{j=1}^{n_{\mathrm{IP}}} \hat{F}_{n,j}\psi_j(\boldsymbol{\xi})\,, \quad \hat{F}_{n,j} = F_n(\hat{U}_j)\,, \quad n = 1,2,3\,, \tag{3.28}$$

$$G^* \approx \sum_{j=1}^{n_{\mathrm{IP}}^s} \hat{G^*}_j\psi_j^s(\boldsymbol{\xi}^s) \quad \hat{G^*}_j = G^*(\hat{U}_j^L, \hat{U}_j^R, \boldsymbol{n}_j) \tag{3.29}$$

instead of using a Gauss quadrature, to avoid the additional cost for the evaluation of the solution to the Gauss points. As long as the flux function only depends linearly on the solution, as for example for linear constant coefficient wave equations, no errors are introduced. Otherwise, aliasing errors are likely to occur. Hesthaven and Warburton [58] point out that stabilization by polynomial filtering must be considered for DG schemes with interpolated non-linear fluxes, but is not unconditionally stable. Inserting the flux interpolation into the weak form (3.17) yields

$$\begin{aligned} \int\limits_E JU_t\psi_\ell\, d\boldsymbol{\xi} - \int\limits_E \sum_{i=1}^{3}\left(\sum_{n=1}^{3}(Ja)_n^i \sum_{j=1}^{n_{\mathrm{IP}}} \hat{F}_{n,j}\psi_j(\boldsymbol{\xi})\right)\frac{\partial\psi_\ell}{\partial\xi^i}\, d\boldsymbol{\xi} \\ + \int\limits_{\partial E} \sum_{j=1}^{n_{\mathrm{IP}}^s} \hat{G^*}_j\psi_j^s(\boldsymbol{\xi}^s)\hat{s}\psi_\ell(\boldsymbol{\xi})\, d\boldsymbol{\xi}^s = 0\,. \end{aligned} \tag{3.30}$$

We recover the same discrete DG scheme in matrix-vector notation as in (3.18), but the flux interpolation changes the flux vectors

$$\begin{aligned}\underline{F} &= \Big(F(\hat{U}_1), \ldots, F(\hat{U}_{n_{\mathrm{IP}}})\Big)^T , \\ \underline{G^*} &= \Big(G^*(\hat{U}_1^L, \hat{U}_1^R, \boldsymbol{n}_1), \ldots, G^*(\hat{U}^L_{n^s_{\mathrm{IP}}}, \hat{U}^R_{n^s_{\mathrm{IP}}}, \boldsymbol{n}_{n^s_{\mathrm{IP}}})\Big)^T ,\end{aligned} \tag{3.31}$$

and the definition of the element stiffness and lifting matrices,

$$(\tilde{S}_{\ell j})^n = \int\limits_E \sum_{i=1}^{3} \Big((Ja)^i_n \psi_j(\boldsymbol{\xi})\Big) \frac{\partial \psi_\ell}{\partial \xi^i}\, d\boldsymbol{\xi}\,, \quad 1 \le \ell \le n_{\mathrm{IP}}\,, 1 \le j \le n_{\mathrm{IP}}\,, \tag{3.32}$$

$$(\tilde{L}_{\ell j})^s = \int\limits_{\partial E} \psi^s_j(\boldsymbol{\xi}^s)\hat{s}^s \psi_\ell(\boldsymbol{\xi})\, d\boldsymbol{\xi}^{\,s}\,, \quad 1 \le \ell \le n_{\mathrm{IP}}\,, 1 \le j \le n^s_{\mathrm{IP}}\,, \tag{3.33}$$

which can be integrated with an appropriate Gauss quadrature. Note that the matrix size is reduced due to the flux interpolation, reducing the cost of the DG operator.

Analogously to the derivations in Section 3.2, if we want to have only one set of stiffness and lifting matrices per element type, we use the mass-lumped DG scheme (3.23), and interpolate the transformed fluxes

$$\mathcal{F}^i(U) \approx \sum_{j=1}^{n_{\mathrm{IP}}} \hat{\mathcal{F}}^i_j \psi_j(\boldsymbol{\xi})\,, \quad \hat{\mathcal{F}}^i_j = \mathcal{F}^i(\hat{U}_j)\,, \quad i = 1,2,3\,, \tag{3.34}$$

$$(G^*\hat{s}) \approx \sum_{j=1}^{n^s_{\mathrm{IP}}} \hat{G^*}_j \hat{s}_j \psi^s_j(\boldsymbol{\xi}^s) \quad \hat{G^*}_j = G^*(\hat{U}^L_j, \hat{U}^R_j, \boldsymbol{n}_j)\,, \tag{3.35}$$

which introduces additional errors for non-linear mappings, even if the flux function is linear. Again, the definition of the stiffness and lifting matrices change to

$$(S_{\ell j})^i = \int\limits_E \psi_j(\boldsymbol{\xi}) \frac{\partial \psi_\ell}{\partial \xi^i}\, d\boldsymbol{\xi}\,, \quad 1 \le \ell \le n_{\mathrm{IP}}\,, 1 \le j \le n_{\mathrm{IP}}\,, \tag{3.36}$$

$$(L_{\ell j})^s = \int\limits_{\partial E} \psi^s_j(\boldsymbol{\xi}^s)\psi_\ell(\boldsymbol{\xi})\, d\boldsymbol{\xi}^{\,s}\,, \quad 1 \le \ell \le n_{\mathrm{IP}}\,, 1 \le j \le n^s_{\mathrm{IP}}\,. \tag{3.37}$$

The nodal DG scheme with flux interpolation produces a cost effective implementation, especially with linear element mappings ($J = \text{const.}$) and linear equations. The latter formulation needs the smallest amount of memory and operations. However, for non-linear mappings and flux functions, aliasing errors will decrease the accuracy of the scheme, and additional mesh refinement and/or filtering may be necessary.

3.2.2. Discrete Equivalence of the Weak and Strong Form

The weak and strong form of the DG formulation (2.45) and (2.46) are equal on an analytical level. In the strong form of the equations, the divergence of the flux remains in the volume integral. Before we can apply a numerical integration, the flux must be approximated so that the derivative can be determined. For Discontinuous Galerkin Spectral Element Methods (DG-SEM), it was shown by Kopriva and Gassner in [68] that the definition of a global flux polynomial inside each element leads to the discrete equivalence of the weak and strong form.

The volume integral of the flux in strong form reads as

$$-\int_E \nabla_{\boldsymbol{\xi}} \cdot \boldsymbol{\mathcal{F}}\, \psi_l \, d\boldsymbol{\xi} = -\sum_{i=1}^{3} \int_E \left(\frac{\partial}{\partial \xi^i} \mathcal{F}^i \right) \psi_l \, d\boldsymbol{\xi} \,. \tag{3.38}$$

If we introduce an element-local flux function, defined by an interpolation polynomial

$$\mathcal{F}^i \approx \sum_{j=1}^{n_{\mathrm{IP}}} \hat{\mathcal{F}}^i \psi_j(\boldsymbol{\xi}) \tag{3.39}$$

the RHS of (3.38) is

$$-\sum_{i=1}^{3} \sum_{j=1}^{n_{\mathrm{IP}}} \hat{\mathcal{F}}^i_j \int_E \frac{\partial \psi_j(\boldsymbol{\xi})}{\partial \xi^i} \; \psi_l(\boldsymbol{\xi}) \, d\boldsymbol{\xi} \,. \tag{3.40}$$

A partial integration of the integral yields

$$-\int_E \left(\frac{\partial}{\partial \xi^i} \psi_j(\boldsymbol{\xi}) \right) \psi_l \, d\boldsymbol{\xi} = \int_E \psi_j \frac{\partial \psi_l(\boldsymbol{\xi})}{\partial \xi^i} \, d\boldsymbol{\xi} + N_i \int_{\partial E} \psi_j(\boldsymbol{\xi}) \psi_l(\boldsymbol{\xi}) \, d\boldsymbol{\xi}^{\,s} \,. \tag{3.41}$$

Inserting the first term into (3.40) yields the volume integral of the weak form

$$\sum_{i=1}^{3} \sum_{j=1}^{n_{\mathrm{IP}}} \hat{\mathcal{F}}^i_j \int_E \psi_j \frac{\partial \psi_l(\boldsymbol{\xi})}{\partial \xi^i} \, d\boldsymbol{\xi} \,. \tag{3.42}$$

Since we use a Lagrange basis where surface nodes are reused as volume nodes, the surface term simplifies to

$$\sum_{i=1}^{3}\sum_{j=1}^{n_{\mathrm{IP}}} \hat{\mathcal{F}}_j^i N_i \int\limits_{\partial E} \psi_j(\xi)\psi_l(\boldsymbol{\xi})\, d\boldsymbol{\xi}^{\,s} = \sum_{j=1}^{n_{\mathrm{IP}}^s} \left(\hat{\boldsymbol{F}}_j^{\mathrm{int}} \cdot \boldsymbol{n}_j \hat{s}_j\right) \int\limits_{\partial E} \psi_j^s(\xi)\psi_l(\boldsymbol{\xi})\, d\boldsymbol{\xi}^{\,s}, \tag{3.43}$$

which cancels out the internal flux evaluation of the strong surface integral and leaves the surface integral of the weak form. This shows that if a global polynomial for the transformed fluxes is used, the weak and the strong form are equivalent on the discrete level. We want to stress that even though the flux polynomial is represented with a Lagrange basis, the coefficients could be found by projection instead of interpolation. To be precise, it is only important that the fluxes of the volume and the surface integrals are evaluated from the same polynomial flux function.

3.3. DG-SEM for Hexahedral Elements

For hexahedra (and quadrilaterals in 2D), one set of implementation choices leads to the spectral collocation form of the discontinuous Galerkin method (DG-SEM), using tensor-product basis functions. They are found in numerous applications [16, 39, 40, 52, 53, 69, 70, 88] and are easy to implement [71]. In the DG-SEM, the solution and the fluxes are approximated by interpolation at the nodes of a Legendre-Gauss-type quadrature and integrals are replaced by either Gauss or Gauss-Lobatto quadratures. An efficient approach often taken is to use the same points for interpolation and integration. Regarding the number of operations, using Gauss-Lobatto nodes seems to be less expensive. However, it was found in [68] that the choice of Gauss nodes is superior in terms of accuracy and efficiency. The dispersion and dissipation properties of DG-SEM can be found in [47].

In [28], DG-SEM reveals a highly reduced computational cost per degree of freedom (DOF), compared to a DG method on tetrahedra. In comparison with continuous Galerkin Spectral Element methods (SEM) where interpolation points have to include the boundary nodes (Gauss-Lobatto points), DG can also use Gauss points, which have a higher integration precision, see [68] for details. In this section, we will derive the DG-SEM in detail and illustrate that the tensor-product basis and the collocation of the integration and interpolation points reduces the amount of work significantly, in comparison to the standard nodal DG formulation of Section 3.2.

3.3.1. Interpolation and Discrete Projection

An important property for collocation methods is that every interpolation can be interpreted as a discrete projection. We show the equivalence of an interpolation and a discrete projection in one dimension, with a set of $(N+1)$ interpolation nodes

$$\left\{\hat{\xi}\right\}_{j=0}^{N}, \quad -1 \leq \xi_j \leq 1 \,. \tag{3.44}$$

An arbitrary function $g(\xi)$ is approximated by Lagrangian interpolation

$$g(\xi) \approx \sum_{j=0}^{N} \hat{g}_j \ell_j(\xi) \,, \quad \hat{g}_j = g(\hat{\xi}_j) \,, \quad \ell_j(\hat{\xi}_i) = \delta_{ij} \,, \tag{3.45}$$

with the Lagrange basis function $\ell_j(\xi)$. A quadrature is an approximation of an integral

$$\int\limits_{-1}^{1} g(x)\, d\xi \approx \sum_{k=0}^{N_q} g(\xi_k)\omega_k \,, \tag{3.46}$$

with (N_q+1) quadrature points and weights

$$\{\xi_k, \omega_k\}_{k=0}^{N_q} \,. \tag{3.47}$$

The quadrature weights are defined by

$$\int\limits_{-1}^{1} g(\xi)\, d\xi \approx \sum_{k=0}^{N^q} g(\xi_k) \int\limits_{-1}^{1} \ell_k^q(\xi)\, d\xi \,, \quad \Rightarrow \quad \omega_k = \int\limits_{-1}^{1} \ell_k^q(\xi)\, d\xi \,, \tag{3.48}$$

because the evaluation of the function at quadrature points is also an interpolation with Lagrange basis functions $\ell_k^q(\xi_i) = \delta_{ik}$ defined by the quadrature points.

The exact projection of an arbitrary function $g(\xi)$ onto a polynomial space is defined by

$$\int\limits_{-1}^{1} g(\xi)\ell_i(\xi)\, d\xi = \sum_{j=0}^{N} \hat{g}_j \int\limits_{-1}^{1} \ell_j(\xi)\ell_i(\xi)\, d\xi \,, \tag{3.49}$$

which we can solve for the polynomial coefficients $\hat{g}_j$. In general $\hat{g}_j \neq g(\hat{\xi}_j)$. In a discrete projection, one replaces the integrals by a quadrature

$$\sum_{k=0}^{N_q} g(\xi_k)\ell_i(\xi_k)\omega_k = \sum_{j=0}^{N} \hat{g}_j \sum_{k=0}^{N_q} \ell_j(\xi_k)\ell_i(\xi_k)\omega_k \,. \tag{3.50}$$

Now, if we use the same points for interpolation and quadrature, we get

$$\begin{aligned}\sum_{k=0}^{N} g(\hat{\xi}_k)\underbrace{\ell_i(\hat{\xi}_k)}_{\delta_{ik}}\omega_k &= \sum_{j=0}^{N}\hat{g}_j\sum_{k=0}^{N_q}\underbrace{\ell_j(\hat{\xi}_k)}_{\delta_{jk}}\underbrace{\ell_i(\hat{\xi}_k)}_{\delta_{ik}}\omega_k\,,\\ g(\hat{\xi}_i)\omega_i &= \hat{g}_i\omega_i\,,\\ g(\hat{\xi}_i) &= \hat{g}_i\,,\end{aligned} \tag{3.51}$$

showing that interpolation can always be seen as a discrete orthogonal projection, being exact only up to the maximum polynomial degree of the integration. In 1D, $(N+1)$ Gauss points would integrate polynomials of degree $N_{\text{int}} = (2N+1)$ exactly and Gauss-Lobatto up to $N_{\text{int}} = (2N-1)$, as opposed to $(N+1)$ equidistant points with $N_{\text{int}} = N$.

The discrete projection property of the one-dimensional Gauss-point interpolation extends to multiple dimensions only for quadrilaterals and hexahedra, where tensor-product Gauss or Gauss-Lobatto nodes can be used as interpolation points. The interpolation points for other element types generally have an integration precision of N, leading to a less accurate discrete projection for polynomial functions. For arbitrary functions it was shown by Trefethen [94] that the Gauss quadrature may not be superior in accuracy than Clenshaw-Curtis quadrature points that have a polynomial integration precision of N.

3.3.2. Approximation of the Solution and the Fluxes

The solution of each hexahedral element is approximated by a polynomial tensor-product basis with degree N in each space direction

$$U(\boldsymbol{X}(\boldsymbol{\xi})) \approx U(\boldsymbol{\xi}) = \sum_{i,j,k=0}^{N}\hat{U}_{ijk}\psi_{ijk}(\boldsymbol{\xi})\,, \qquad \psi_{ijk}(\boldsymbol{\xi}) = \ell_i(\xi^1)\ell_j(\xi^2)\ell_k(\xi^3)\,, \tag{3.52}$$

with the nodal degrees of freedom $\hat{U}_{ijk}$ and the one-dimensional Lagrange interpolating polynomials ℓ_i defined in (2.7). We will use either Gauss or Gauss-Lobatto points for the nodal points $\hat{\xi}_i$. Unless stated otherwise, we use Gauss points, being more efficient according to [68].

Analogously, the three contravariant fluxes are approximated by interpolation at the nodal points

$$\mathcal{F}^m(U(\boldsymbol{\xi})) \approx \sum_{i,j,k=0}^{N}\hat{\mathcal{F}}^m_{ijk}\psi_{ijk}(\boldsymbol{\xi})\,, \quad m = 1,2,3\,, \tag{3.53}$$

and the polynomial coefficients are computed as the transformed physical fluxes

$$\mathcal{F}^m_{ijk} = \mathcal{F}^m(\xi^1_i, \xi^2_j, \xi^3_k) = \sum_{d=1}^{3} (Ja)^m_d|_{\xi^1_i, \xi^2_j, \xi^3_k} \, F_d(\hat{U}_{ijk})\,. \tag{3.54}$$

Since we have non-linear fluxes and possibly non-linear metric terms, this step introduces errors by truncation of the orthogonal polynomial infinite series representation of the function and aliasing of high order terms [27]. However, as stated in the previous section, the interpolation of the transformed fluxes is a discrete projection of degree $N_{\text{int}} = (2N + 1)$ for Gauss points and $N_{\text{int}} = (2N - 1)$ for Gauss-Lobatto.

3.3.3. Discrete Metric Terms and Free-stream Preservation

We already introduced in Section 2.1 the concept of *free-stream preservation*, meaning that the divergence of a constant flux must remain zero under transformation. This was done for the analytical metric terms, leading to the metric identities [71]

$$\sum_{i=1}^{3} \frac{\partial (J\boldsymbol{a})^i}{\partial \xi^i} = \vec{0}\,. \tag{3.55}$$

Two-dimensional metric terms directly satisfy the analytical condition, which is not the case in three dimensions. In the work Kopriva [67], a solution is found investigating the discrete DG-SEM in three dimensions.
Under the assumption of collocation of flux and interpolation points and that normal vectors are continuous at element faces, the strong form of the DG scheme yields the discrete metric identities

$$\sum_{i=1}^{3} \frac{\partial I^N\left((J\boldsymbol{a})^i\right)}{\partial \xi^i} = \vec{0}\,, \tag{3.56}$$

Here, $I^N(.)$ is the interpolation operator of the solution with polynomial degree N. The idea of Kopriva is to formulate the metrics in curl form, since the divergence of a curl is automatically zero. The so-called invariant curl form, introduced by Vinokur and Yee [95] for Finite Difference schemes, reads as

$$(Ja)^i_n = -\frac{1}{2}\hat{x}_i \cdot \left[\boldsymbol{\nabla}_{\boldsymbol{\xi}} \times \left(I^N\left(X_l \boldsymbol{\nabla}_{\boldsymbol{\xi}} X_m\right) - I^N\left(X_m \boldsymbol{\nabla}_{\boldsymbol{\xi}} X_l\right)\right)\right]\,, \quad Ja^i_n \in \mathbb{P}^N\,, \tag{3.57}$$

where $\hat{x}_i$ is the unit vector in physical space in direction i and (n, m, l) are cyclic indices. Note that the interpolation of a product will be approximated as

$$I^N(ab) = I^N(I^N(a)I^N(b))\,. \tag{3.58}$$

The interpolation operator refers to one unique nodal set, thus a product of two interpolated values will *not increase* the degree of the polynomial, since the values are collocated. The nodal set for representing the metric terms should include the boundaries of the element, so we typically choose Chebychev-Lobatto points.

We suppose that exact free-stream preservation on the discrete level by using the invariant curl form is not restricted to hexahedra and spectral element operators, but generalizes to other DG schemes and element types as long as the transformed fluxes are computed via collocation at the interpolation points. However, the accuracy of the flux collocation may be inferior compared to hexahedra, since it strongly depends on the element types and interpolation point position. For example Hesthaven and Warburton [58] switch from nodal to Gauss quadrature for the flux approximation on curved triangles and tetrahedra, yielding only an approximate free-stream preservation, depending on integration accuracy and non-linearity of the mapping.

The discrete metric identities are derived from the strong form and the weak form is always exactly conservative. Strong and weak form are discretely equivalent for DG-SEM, thus the scheme is exactly free-stream preserving [67].

3.3.4. Integral of the Time Derivative

We deal with the first term of the weak formulation of the hyperbolic conservation equation for one element in reference space, (2.45), the time-derivative integral. We insert the approximation of the solution (3.52) and choose the test function as $\phi = \psi_{ijk}$:

$$\frac{\partial}{\partial t}\int\limits_E JU\phi\, d\boldsymbol{\xi} = \frac{\partial}{\partial t}\int\limits_E J(\boldsymbol{\xi})\left(\sum_{lmn=0}^{N} \hat{U}_{lmn}\psi_{lmn}(\boldsymbol{\xi})\right)\psi_{ijk}(\boldsymbol{\xi})\, d\boldsymbol{\xi}\,. \tag{3.59}$$

We split the integral in the coordinate directions and approximate each term by a Gauss-Legendre quadrature,

$$
\begin{aligned}
\int_E JU\phi \, d\boldsymbol{\xi} &= \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} J(\boldsymbol{\xi})U(\boldsymbol{\xi})\psi_{ijk}(\boldsymbol{\xi}) d\xi^1 d\xi^2 d\xi^3 \\
&\approx \sum_{\lambda\mu\nu=0}^{N} J(\boldsymbol{\xi}_{\lambda\mu\nu}) \left(\sum_{lmn=0}^{N} \hat{U}_{lmn} \underbrace{\ell_l(\xi^1_\lambda)}_{=\delta_{l\lambda}} \underbrace{\ell_m(\xi^2_\mu)}_{=\delta_{m\mu}} \underbrace{\ell_n(\xi^3_\nu)}_{=\delta_{n\nu}} \right) \psi_{ijk}(\boldsymbol{\xi}_{\lambda\mu\nu}) \omega_\lambda \omega_\mu \omega_\nu \\
&= \sum_{\lambda\mu\nu=0}^{N} J(\boldsymbol{\xi}_{\lambda\mu\nu}) \hat{U}_{\lambda\mu\nu} \underbrace{\ell_i(\xi^1_\lambda)}_{=\delta_{i\lambda}} \underbrace{\ell_j(\xi^2_\mu)}_{=\delta_{j\mu}} \underbrace{\ell_k(\xi^3_\nu)}_{=\delta_{k\nu}} \omega_\lambda \omega_\mu \omega_\nu \, .
\end{aligned}
\tag{3.60}
$$

This can be reduced further thanks to the collocation of interpolation and integration and the Lagrange property (2.8), yielding

$$
\frac{\partial}{\partial t} \int_E JU\phi \, d\boldsymbol{\xi} \approx J(\boldsymbol{\xi}_{ijk}) \omega_i \omega_j \omega_k \frac{\partial \hat{U}_{ijk}}{\partial t} \, . \tag{3.61}
$$

Note that (3.61) is integrated exactly by the Gauss quadrature, as long as the Jacobian is linear in $\boldsymbol{\xi}$, thus only for linear and *bilinear* hexahedra. For trilinear and non-linear mappings an additional integration error is introduced, already termed *mass-lumping* in Section 3.3. See Section C.1 for details on the Jacobian of bilinear and trilinear element mappings.

If Gauss-Lobatto points are used as collocation points, the mass matrix is always mass-lumped, since the integration precision is only $(2N-1)$. In [47], the dissipation and dispersion relations of both point sets are compared, and the mass-lumping of the Gauss-Lobatto 'under-integration' is shown to act as a filter on the last polynomial mode.

3.3.5. Volume Integral

We can split the volume integral, i.e. the second term in (2.45), into the three contravariant flux directions

$$
\int_E \boldsymbol{\mathcal{F}}(U) \cdot \left(\boldsymbol{\nabla}_{\xi} \phi\right) \, d\boldsymbol{\xi} = \sum_{d=1}^{3} \int_E \mathcal{F}^d(U) \frac{\partial \phi}{\partial \xi^d} \, d\boldsymbol{\xi} \, , \tag{3.62}
$$

and focus on the first component for demonstration. We insert the flux approximation (3.53), set the test function to $\phi = \psi_{ijk}$ and replace the integrals by Gauss quadrature. Again, nearly all sums cancel

$$\begin{aligned}
&\int\limits_E \mathcal{F}^1(U)\frac{\partial\phi}{\partial\xi^1}\,d\boldsymbol{\xi} \\
&= \int\limits_{-1}^{1}\int\limits_{-1}^{1}\int\limits_{-1}^{1} J(\boldsymbol{\xi})U(\boldsymbol{\xi})\psi_{ijk}(\boldsymbol{\xi})\left(\sum_{l,m,n=0}^{N}\hat{\mathcal{F}}^1_{lmn}\psi_{lmn}(\boldsymbol{\xi})\right)\frac{\partial\psi_{ijk}(\boldsymbol{\xi})}{\partial\xi^1}d\xi^1 d\xi^2 d\xi^3 \\
&\approx \sum_{\lambda,\mu,\nu=0}^{N}\left(\sum_{l,m,n=0}^{N}\hat{\mathcal{F}}^1_{lmn}\underbrace{\ell_l(\xi^1_\lambda)}_{=\delta_{l\lambda}}\underbrace{\ell_m(\xi^2_\mu)}_{=\delta_{m\mu}}\underbrace{\ell_n(\xi^3_\nu)}_{=\delta_{n\nu}}\right)\frac{\partial\psi_{ijk}(\boldsymbol{\xi})}{\partial\xi^1}\omega_\lambda\omega_\mu\omega_\nu \\
&= \sum_{\lambda,\mu,\nu=0}^{N}\hat{\mathcal{F}}^1_{\lambda\mu\nu}\left.\frac{d\ell_i(\xi)}{d\xi}\right|_{\xi=\xi^1_\lambda}\underbrace{\ell_j(\xi^2_\mu)}_{=\delta_{j\mu}}\underbrace{\ell_k(\xi^3_\nu)}_{=\delta_{k\nu}}\omega_\lambda\omega_\mu\omega_\nu \\
&= \omega_j\omega_k\sum_{\lambda=0}^{N}\hat{\mathcal{F}}^1_{\lambda jk}\left.\frac{d\ell_i(\xi)}{d\xi}\right|_{\xi=\xi^1_\lambda}\omega_\lambda\,.
\end{aligned} \tag{3.63}$$

Equation (3.63) can be seen as a one-dimensional matrix vector product with the differentiation matrix

$$D_{ij} = \left.\frac{d\ell_j(\xi)}{d\xi}\right|_{\xi=\xi_i}, \quad i,j=0,\ldots,N\,. \tag{3.64}$$

Gathering all components, the volume integral approximates as

$$\begin{aligned}
\int\limits_E \boldsymbol{\mathcal{F}}(U)\cdot\boldsymbol{\nabla}_{\boldsymbol{\xi}}\,\phi\,d\boldsymbol{\xi} \approx\quad & \omega_j\omega_k\sum_{\lambda=0}^{N}D_{\lambda i}\hat{\mathcal{F}}^1_{\lambda jk}\omega_\lambda \\
&+\omega_i\omega_k\sum_{\mu=0}^{N}D_{\mu j}\hat{\mathcal{F}}^2_{i\mu k}\omega_\mu \\
&+\omega_i\omega_j\sum_{\nu=0}^{N}D_{\nu k}\hat{\mathcal{F}}^3_{ij\nu}\omega_\nu\,.
\end{aligned} \tag{3.65}$$

3.3.6. Surface Integral

The surface integral, i.e. the third term in (2.45), is computed as the sum of one-dimensional integrals

$$\int\limits_{\partial E} (G^*\hat{s})\ \phi(\boldsymbol{\xi})\, d\boldsymbol{\xi}^{\,s} = \left[\int\limits_{-1}^{1}\int\limits_{-1}^{1} (G^*\hat{s})\ \phi\, d\xi^2\, d\xi^3\right]_{\xi^1=-1}^{1} + \left[\int\limits_{-1}^{1}\int\limits_{-1}^{1} (G^*\hat{s})\ \phi\, d\xi^3\, d\xi^1\right]_{\xi^2=-1}^{1} + \left[\int\limits_{-1}^{1}\int\limits_{-1}^{1} (G^*\hat{s})\ \phi\, d\xi^1\, d\xi^2\right]_{\xi^3=-1}^{1} . \tag{3.66}$$

We choose the surface integrals in the ξ^1 direction for demonstration. We will approximate the numerical flux by interpolation on the element face Gauss nodes, being aligned to the inner Gauss nodes, see Fig. 3.3, yielding

$$G^*(\pm 1, \xi^2, \xi^3)\,\hat{s}^{\pm} \approx \sum_{j,k=0}^{N} [G^*\hat{s}]_{j,k}^{\pm\xi^1}\ \ell_j(\xi^2)\ell_k(\xi^3)\,, \tag{3.67}$$

where evaluations on the face in positive ξ^1 direction are marked by "$+\xi^1$" and on the face in negative ξ^1 direction by "$-\xi^1$".
We insert the test function $\phi = \psi_{ijk}(\boldsymbol{\xi}) = \ell_i(\xi^1)\ell_j(\xi^2)\ell_k(\xi^3)$ into the surface integrals $\xi^1 = \pm 1$

$$\left[\int\limits_{-1}^{1}\int\limits_{-1}^{1} (G^*\,\hat{s})\ \ell_i(\xi^1)\ell_j(\xi^2)\ell_k(\xi^3)\, d\xi^2\, d\xi^3\right]_{\xi^1=-1}^{1} , \tag{3.68}$$

and approximate again the integrals by Gauss quadrature, leading to

$$\begin{aligned} &\approx \sum_{\mu,\nu=0}^{N}\left(\sum_{mn=0}^{N} [G^*\hat{s}]_{mn}^{+\xi^1}\ \ell_m(\xi^2_\mu)\ell_n(\xi^3_\nu)\right)\ell_i(1)\ell_j(\xi^2_\mu)\ell_k(\xi^3_\nu)\omega_\mu\omega_\nu \\ &+ \sum_{\mu,\nu=0}^{N}\left(\sum_{mn=0}^{N} [G^*\hat{s}]_{mn}^{-\xi^1}\ \ell_m(\xi^2_\mu)\ell_n(\xi^3_\nu)\right)\ell_i(-1)\ell_j(\xi^2_\mu)\ell_k(\xi^3_\nu)\omega_\mu\omega_\nu \\ &= \left([G^*\hat{s}]_{jk}^{+\xi^1}\ \ell_i(1) - [G^*\hat{s}]_{jk}^{-\xi^1}\ \ell_i(-1)\right)\omega_j\omega_k\,, \end{aligned} \tag{3.69}$$

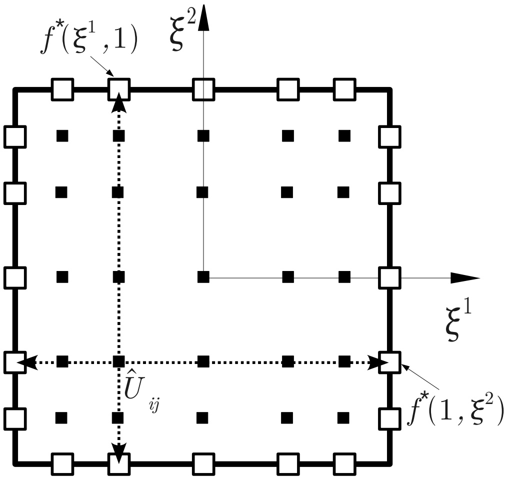


Figure 3.3.: Location of Gauss points ■ and boundary fluxes □ in 2D.

where again the Lagrange property helps to simplify the surface integral significantly. We can apply the surface integral analogously to the ξ^2, ξ^3 directions, yielding the total surface integral contribution

$$\begin{aligned}\int_{\partial E} (G^* \hat{s})\, \phi \, d\boldsymbol{\xi}^{\,s} \approx \\ &\Big([G^*\hat{s}]_{jk}^{+\xi^1}\ \ell_i(1) + [G^*\hat{s}]_{jk}^{-\xi^1}\ \ell_i(-1) \Big)\, \omega_j \omega_k + \\ &\Big([G^*\hat{s}]_{ik}^{+\xi^2}\ \ell_j(1) + [G^*\hat{s}]_{ik}^{-\xi^2}\ \ell_j(-1) \Big)\, \omega_i \omega_k + \\ &\Big([G^*\hat{s}]_{ij}^{+\xi^3}\ \ell_k(1) + [G^*\hat{s}]_{ij}^{-\xi^3}\ \ell_k(-1) \Big)\, \omega_i \omega_j \,. \end{aligned} \tag{3.70}$$

3.3.7. Semi-discrete Formulation

We define one-dimensional precomputed operators as

$$\hat{\ell}_i = \frac{\ell_i}{\omega_i}\,, \quad \hat{D}_{ij} = -D_{ji}\frac{\omega_i}{\omega_j}\,, \qquad i,j = 0,\ldots,N\,. \tag{3.71}$$

We can now assemble the time integral (3.61), the volume integral (3.65) and the surface integral (3.70) from the previous sections in the weak form (2.45) and formulate the time derivative of the DOF on each element

$$
\begin{aligned}
(U_{ijk})_t = -\frac{1}{J_{ijk}} \Bigg[& \left(\sum_{\lambda=0}^{N} \hat{D}_{i\lambda} \hat{\mathcal{F}}^1_{\lambda jk} \right) + \left([G^*\hat{s}]^{+\xi^1}_{jk} \; \hat{\ell}_i(1) + [G^*\hat{s}]^{-\xi^1}_{jk} \; \hat{\ell}_i(-1) \right) \\
& + \left(\sum_{\mu=0}^{N} \hat{D}_{j\mu} \hat{\mathcal{F}}^2_{i\mu k} \right) + \left([G^*\hat{s}]^{+\xi^2}_{ik} \; \hat{\ell}_j(1) + [G^*\hat{s}]^{-\xi^2}_{ik} \; \hat{\ell}_j(-1) \right) \\
& + \left(\sum_{\nu=0}^{N} \hat{D}_{k\nu} \hat{\mathcal{F}}^3_{ij\nu} \right) + \left([G^*\hat{s}]^{+\xi^3}_{ij} \; \hat{\ell}_k(1) + [G^*\hat{s}]^{-\xi^3}_{ij} \; \hat{\ell}_k(-1) \right) \Bigg] .
\end{aligned}
\tag{3.72}
$$

We need to compute the contravariant fluxes $\mathcal{F}^m$ by (2.33) for the volume integral and we have to evaluate the solution, defined on the inner Gauss points, at the element's boundary to compute the Riemann fluxes $G^*(U^L, U^R, \boldsymbol{n})$ for the surface integral. The 'prolongation' to the surface points, as shown in Fig. 3.3, is again only a one-dimensional scalar product [71], for $+\xi^1$ for example

$$
U^{+\xi^1}_{jk} = \sum_{i=0}^{N} U_{ijk} \ell_i(\xi = 1) \,. \tag{3.73}
$$

The surface flux contribution, eg. $[G^*\hat{s}]^{+\xi^1}_{jk}$, is only computed for one element side, since for the neighbor element, this contribution is simply $-[G^*\hat{s}]^{+\xi^1}_{jk}$. In a parallel computation, we send the state on the element face and receive the flux contribution.

Note that the semi-discrete form for three-dimensional equations consists of only one-dimensional DG operators – each row in (3.72) – applied in a dimension-by-dimension fashion, very similar to a Finite Difference scheme. Compared to a standard DG method, where the number of operations per element with $(N+1)^3$ DOF would scale as $\sim (N+1)^6$ (full 3D matrices), the operation count for DG-SEM scales as $\sim 3(N+1)^4$.

It was shown by Kopriva and Gassner [68] that the strong and the weak form of the DG-SEM are identical on the discrete level. In the case of Gauss-Lobatto, the discrete identity of strong and weak forms is also called the *summation by parts* (SBP) property, which links state-of-the-art Finite Difference schemes to DG-SEM, see [46] for details. The semi-discrete strong form is written simply as

$$
\begin{aligned}
(U_{ijk})_t = -\frac{1}{J_{ijk}} \Bigg[&\left(\sum_{\lambda=0}^{N} D_{i\lambda} \hat{\mathcal{F}}^1_{\lambda jk} \right) + \left[(G^* \hat{s}) - \mathcal{F}^1 \right]_{jk}^{\pm\xi^1} \hat{\ell}_i(\pm 1) \\
+ &\left(\sum_{\mu=0}^{N} D_{j\mu} \hat{\mathcal{F}}^2_{i\mu k} \right) + \left[(G^* \hat{s}) - \mathcal{F}^2 \right]_{ik}^{\pm\xi^2} \hat{\ell}_j(\pm 1) \\
+ &\left(\sum_{\nu=0}^{N} D_{k\nu} \hat{\mathcal{F}}^3_{ij\nu} \right) + \left[(G^* \hat{s}) - \mathcal{F}^3 \right]_{ij}^{\pm\xi^3} \hat{\ell}_k(\pm 1) \Bigg] ,
\end{aligned}
\tag{3.74}
$$

where we directly apply the differentiation matrix D and need the volume flux evaluated at the element boundaries, for $+\xi^1$ for example

$$
\left[\mathcal{F}^1 \right]_{jk}^{+\xi^1} = \sum_{i=0}^{N} \mathcal{F}^1_{ijk} \ell_i(\xi = 1) \,. \tag{3.75}
$$

The strong form is therefore computationally more expensive than the weak form, especially when Gauss points are used, because of the additional interpolation of the volume flux. For the Gauss-Lobatto variant, collocation points include the element boundaries, the interpolation can be circumvented by evaluating the flux from the DG solution and the surface metrics at the boundary collocation points

$$
\left[\mathcal{F}^1 \right]_{jk}^{+\xi^1} = \left[(J\boldsymbol{a})^1 \cdot \boldsymbol{F} \right]_{jk}^{+\xi^1} = \boldsymbol{F}([U]_{jk}^{+\xi^1}) \cdot [\boldsymbol{n}\hat{s}]_{jk}^{+\xi^1} \text{ ,for LGL points} \,. \tag{3.76}
$$

3.3.8. Lifting for Second Order Equations with DG-SEM

In this section, we present the discretization of the gradient lifting with DG-SEM, focusing on the difference between the Bassi-Rebay 1 (BR1) [6] and Bassi-Rebay 2 (BR2) scheme [10]. As derived in Section 2.3.2, the flux of the mixed hyperbolic-parabolic system depends on the solution and its gradient. The flux of the volume integral needs the lifted gradient at the volume Gauss points, and the numerical flux function G^* now consists of a hyperbolic part, solved via Riemann solvers, and a parabolic part

$$
G^* = F^*_{\text{adv}}(U^L, U^R, \boldsymbol{n}) + \boldsymbol{n} \cdot \boldsymbol{F}^*_v(U^L, U^R, \boldsymbol{\nabla_x} U^L, \boldsymbol{\nabla_x} U^R) \,. \tag{3.77}
$$

As shown in Section 2.3.2, the flux of the gradient equation is simply the solution U, with the numerical flux U^*. For BR 1 and BR2, they are computed as

$$\begin{aligned} U^* &= \frac{1}{2}(U^L + U^R)\,, \\ \boldsymbol{F}_v^*(U^L, U^R, \boldsymbol{\nabla}_{\boldsymbol{x}} U^L, \boldsymbol{\nabla}_{\boldsymbol{x}} U^R) &= \frac{1}{2}(\boldsymbol{F}_v(U^L, \boldsymbol{\nabla}_{\boldsymbol{x}} U^L) + \boldsymbol{F}_v(U^R, \boldsymbol{\nabla}_{\boldsymbol{x}} U^R))\,, \end{aligned} \tag{3.78}$$

where the parabolic part of the numerical flux is the mean of the left and the right parabolic fluxes. The Bassi-Rebay 1 lifting scheme starts with the weak form of the gradient equation (2.53). Analogously to the hyperbolic equations, we apply the DG-SEM discretization, yielding

$$\begin{aligned} Q^d_{ijk} = \frac{1}{J_{ijk}} \Bigg[& \left(\sum_{\ell=0}^{N} \hat{D}_{i\ell} \mathcal{U}^{d,1}_{\ell jk} + \hat{D}_{j\ell} \mathcal{U}^{d,2}_{i\ell k} + \hat{D}_{k\ell} \mathcal{U}^{d,3}_{ij\ell} \right) \\ & + [U^* n_d \hat{s}]^{+\xi^1}_{jk}\, \hat{\ell}_i(1) + [U^* n_d \hat{s}]^{-\xi^1}_{jk}\, \hat{\ell}_i(-1) \\ & + [U^* n_d \hat{s}]^{+\xi^2}_{ik}\, \hat{\ell}_j(1) + [U^* n_d \hat{s}]^{-\xi^2}_{ik}\, \hat{\ell}_j(-1) \\ & + [U^* n_d \hat{s}]^{+\xi^3}_{ij}\, \hat{\ell}_k(1) + [U^* n_d \hat{s}]^{-\xi^3}_{ij}\, \hat{\ell}_k(-1) \Bigg]. \end{aligned} \tag{3.79}$$

The gradients at the surfaces, needed for the computation of parabolic surface flux $\boldsymbol{F}_v^*$, are then found by interpolation.

$$\left[Q^d\right]^{+\xi^1}_{jk} = \sum_{i=0}^{N} Q^d_{ijk} \ell_i(+1)\,. \tag{3.80}$$

For the sake of simplicity, we only show the derivations for the element face in $+\xi^1$ direction, since all other faces are treated analogously.

It is well known that the BR1 scheme is unstable for purely elliptic equations. The remedy is the BR2 scheme, with so-called *local* lifting operators. Here, the starting point is the strong form, and surface gradients are only lifted with their corresponding element side, with an additional penalty parameter η_{BR2}. Stability was shown for $\eta_{\mathrm{BR2}} >$ number of element faces. The volume flux has to be lifted with all element sides for consistency.

The DG-SEM discretization of the gradient equation in strong form (2.54) yields the lifted gradient at volume Gauss points

$$\begin{aligned} Q^d_{ijk} = \frac{1}{J_{ijk}} \Bigg[& \left(\sum_{\ell=0}^{N} D_{i\ell} \mathcal{U}^{d,1}_{\ell jk} + D_{j\ell} \mathcal{U}^{d,2}_{i\ell k} + D_{k\ell} \mathcal{U}^{d,3}_{ij\ell} \right) \\ & + [H^*]^{+\xi^1}_{jk} \, \hat{\ell}_i(1) + [H^*]^{-\xi^1}_{jk} \, \hat{\ell}_i(-1) \\ & + [H^*]^{+\xi^2}_{ik} \, \hat{\ell}_j(1) + [H^*]^{-\xi^2}_{ik} \, \hat{\ell}_j(-1) \\ & + [H^*]^{+\xi^3}_{ij} \, \hat{\ell}_k(1) + [H^*]^{-\xi^3}_{ij} \, \hat{\ell}_k(-1) \Bigg] . \end{aligned} \tag{3.81}$$

with the abbreviation for the surface flux

$$[H^*]^{+\xi^1}_{jk} = \left[\left(U^* - U^{\text{int}} \right) n_d \hat{s} \right]^{+\xi^1}_{jk} , \tag{3.82}$$

where the jump of the solution is found as the difference of the lifting flux U^* and inner solution evaluated at the surface.

For BR2, the volume contribution must be interpolated to each surface separately, only including the local surface flux

$$\begin{aligned} \left[Q^d \right]^{+\xi^1}_{jk} = \sum_{i=0}^{N} \frac{1}{J_{ijk}} \Bigg[& \left(\sum_{\ell=0}^{N} D_{i\ell} \mathcal{U}^{d,1}_{\ell jk} + D_{j\ell} \mathcal{U}^{d,2}_{i\ell k} + D_{k\ell} \mathcal{U}^{d,3}_{ij\ell} \right) \\ & + \eta_{\text{BR2}} \, [H^*]^{+\xi^1}_{jk} \, \hat{\ell}_i(1) \Bigg] \ell_i(+1) , \end{aligned} \tag{3.83}$$

with a stabilization parameter η_{BR2} introduced for the surface flux.

For general elements, the computational cost of the BR2 scheme is much higher compared to BR1 because the local surface lifting couples face nodes and inner nodes. However, with DG-SEM, we found that an efficient implementation of the BR2 scheme is possible by introducing two steps. The first step ignores the surface contribution. The local volume gradient is computed

$$Q^d_{ijk} = \frac{1}{J_{ijk}} \left(\sum_{\ell=0}^{N} D_{i\ell} \mathcal{U}^d_{1\,\ell jk} + D_{j\ell} \mathcal{U}^d_{2\,i\ell k} + D_{k\ell} \mathcal{U}^d_{3\,ij\ell} \right) \tag{3.84}$$

and is interpolated at the element faces

$$\left[Q^d \right]^{+\xi^1}_{jk} = \sum_{i=0}^{N} Q^d_{ijk} \ell_i(+1) . \tag{3.85}$$

In the second step, for each face, the surface contribution of the lifting is computed locally and added to the volume gradient and surface gradient at the same time:

```
face $(+\xi^1)$ :  for $k, j = 0 \dots N$
    for $i = 0 \dots N$
        $h \leftarrow \frac{1}{J_{ijk}} \left[H^*\right]_{jk}^{+\xi^1} \hat{\ell}_i(1)$
        $Q^d_{ijk} \overset{+}{\leftarrow} h$
        $\left[Q^d\right]_{jk}^{+\xi^1} \overset{+}{\leftarrow} \eta_{\mathrm{BR2}}\, h\, \ell_i(1)\,,$
face $(+\xi^2)$ :  for $k, i = 0 \dots N$
    for $j = 0 \dots N$
        $h \leftarrow \frac{1}{J_{ijk}} \left[H^*\right]_{ik}^{+\xi^2} \hat{\ell}_j(1)$
        $Q^d_{ijk} \overset{+}{\leftarrow} h$                                        (3.86)
        $\left[Q^d\right]_{ik}^{+\xi^2} \overset{+}{\leftarrow} \eta_{\mathrm{BR2}}\, h\, \ell_j(1)\,,$
face $(+\xi^3)$ :  for $j, i = 0 \dots N$
    for $k = 0 \dots N$
        $h \leftarrow \frac{1}{J_{ijk}} \left[H^*\right]_{ij}^{+\xi^3} \hat{\ell}_k(1)$
        $Q^d_{ijk} \overset{+}{\leftarrow} h$
        $\left[Q^d\right]_{ij}^{+\xi^3} \overset{+}{\leftarrow} \eta_{\mathrm{BR2}}\, h\, \ell_k(1)\,,$
... for faces $(-\xi^1, -\xi^2, -\xi^3)$ .
```

The simultaneous lifting of volume and surface gradients minimizes the number of operations.

Note that the discretization and implementation of the local DG scheme [34] is the same as the BR1 scheme, but with different surface fluxes

$$\begin{aligned} U^* &= \beta U^L + (1-\beta) U^R\,, \\ F_v^* &= (1-\beta) F_v(U^L, \boldsymbol{\nabla}_{\boldsymbol{x}} U^L) + \beta F_v(U^R, \boldsymbol{\nabla}_{\boldsymbol{x}} U^R) \end{aligned} \tag{3.87}$$

introducing a switch function $\beta = 0/1$ at the interface, so that either the solution is taken from inside and the gradient from outside, or vice versa. Moreover, the compact DG scheme [83] has the same discretization as the BR2 scheme, but with the fluxes of the local DG scheme (3.87). The local and compact DG scheme do not need the stabilization parameter η_{BR2}.

4. Curved Mesh Generation

In industrial applications, geometries are three-dimensional and typically are comprised of curved surfaces, curved borders and sharp edges. Here meshing itself becomes an issue and therefore, unstructured grids are needed. Most of the unstructured grid generators provide high quality grids consisting of hexahedra, prisms, tetrahedra and pyramids, whereas the element edges are in general straight lines, at most having an additional mid-point (quadratic elements).

Approaches for high order grids should use CAD definitions to guarantee a correct approximation of the geometry. Existing high order grid generators [51] are very promising, however meshing of complex geometries with unstructured hybrid grids has not reached the level of commercial grid generators.

Our approach is to rely on commercial grid generators because of their ability to work on CAD definitions and produce high quality meshes, and to use additional information for the element curving.

Finding the curved element mapping is the key problem of high order grid generation. Generally, two strategies can be distinguished, as sketched in Fig. 4.1. The first is a discrete representation of the surface with points lying on the surface, the second is using continuity conditions, e.g. spline interpolation.

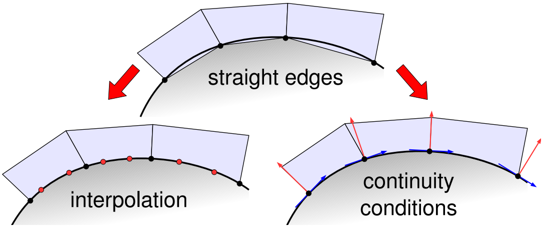


Figure 4.1.: Strategies for the generation of curved element sides.

The first step is to curve boundary faces of elements at curved surfaces. We address both the normal vector and interpolation approaches. Normal vectors of the surface at the element corner nodes are found either from an analytical description of the surface, or by extraction from CAD data via a CAD interface. The interpolation points are found using features of the commercial grid generation software ANSA©and ICEM©.

The curving of the boundary faces of the near wall elements can lead to inverted elements in a boundary layer mesh. To overcome this problem, a mesh deformation strategy is presented in Section 4.5 to propagate the boundary curvature into the mesh volume. Thereof independent is the direct volume curving approach described in Section 4.4 using block-structured meshes transformed to fully curved hexahedral meshes.

All these strategies are implemented in a standalone preprocessing tool called *HOPR* (High Order Preprocessor). It fills the gap between the linear grid generation and the simulation with the high order solver.

This chapter is organized as follows: The first section gives an overview of *HOPR*. The following sections give a detailed description of the curving strategies. Then we compare the approximation properties of the curving techniques for simple geometries in Section 4.6. Furthermore, we show the importance of high order grids on the simple test case of an electro-magnetic wave propagation in a coaxial cable in Section 4.7.1. We also demonstrate the applicability of the proposed approach with an unstructured grid of a complex wing-body-nacelle configuration, which has multiple edges, corners and intersecting sub-surfaces. In Section 4.6.4, we analyze the influence of mesh generation errors on errors of the high order grid. Finally, we will assess the advantages and drawbacks of the different strategies, regarding aspects like accuracy, limits of applicability and complexity.

4.1. Overview

In Fig. 4.2 we show a flowchart of the curved mesh generation process. The starting point is the linear unstructured mesh, created by the grid generator of choice. The CGNS (CFD General Notation System) standard mesh format [91] is commonly used as exchange mesh format. The *point-normals* strategy uses normal vectors at the surface grid points of a given linear mesh. Either they are found by analytical functions, specified by the user, or for complex geometries directly from the CAD model via a CAD interface, providing a list of surface grid points and their associated normal vectors, see Section 4.2.1 for details.

In this work, the commercial grid generators ANSA©and ICEM©were used to generate linear unstructured meshes. The software generates meshes using the original CAD geometry, and also provides CAD modifications, which are often necessary to clean up the geometry. The ANSA©mesh generator has a build-in feature (called *split*) to subdivide surface meshes, and ICEM©is able to extract Chebychev-Lobatto interpolation points along the element edges (called *spectral elements*). The points are again distributed along the original CAD surface, which makes it possible to use these points to construct high order boundary faces by simple interpolation, see Section 4.3.

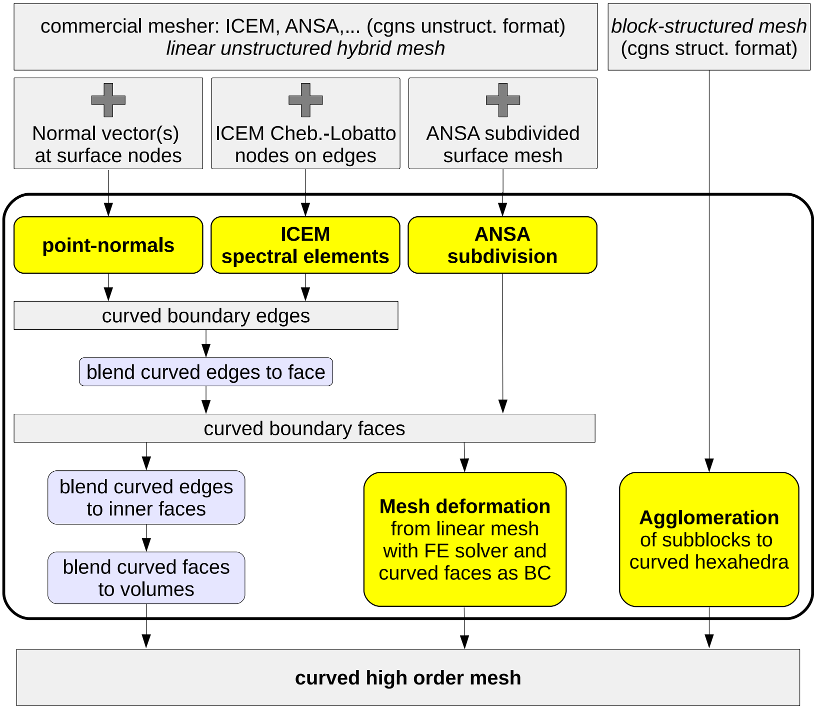

commercial mesher: ICEM, ANSA,... (cgns unstruct. format)
linear unstructured hybrid mesh
block-structured mesh
(cgns struct. format)
Normal vector(s) at surface nodes
ICEM Cheb.-Lobatto nodes on edges
ANSA subdivided surface mesh
point-normals
ICEM spectral elements
ANSA subdivision
curved boundary edges
blend curved edges to face
curved boundary faces
blend curved edges to inner faces
blend curved faces to volumes
Mesh deformation from linear mesh with FE solver and curved faces as BC
Agglomeration of subblocks to curved hexahedra
curved high order mesh

Figure 4.2.: Flowchart of the curved mesh generation process.

The ANSA©subdivision directly produces *curved boundary faces*, whereas the other approaches only produce *curved element edges*, which are then blended to a curved boundary face. The same blending is used for the inner faces sharing edges with the curved boundary face, and finally the volume is again a blending of its faces. The blending techniques are detailed in Section 4.2.2 and Appendix D.

However, surface curving only provides curved elements in the first layer at the boundary, and may lead to inverted element mappings, for example for boundary layer meshes. To make the elements valid, a mesh deformation strategy, first introduced in [84], is applied. Here, the mesh is modeled as a deformable solid, and the curved boundary is imposed as displacement from the linear mesh. A high order Finite Element Method solves for the deformed mesh, leading to a propagation of the boundary curving into the mesh volume. More details are found in Section 4.5.

Finally, a completely separated strategy is a simple agglomeration of block-structured grids, directly leading to fully curved hexahedra, where intermediate structured points are used for interpolation of the element mapping, see Section 4.4.

All strategies generate curved elements with a polynomial element mapping introduced in Section 2.1. Analogously to the curved element mapping (2.3), we introduce the surface mapping for the curved boundary faces

$$\boldsymbol{X}^S(\xi_1,\xi_2) = \sum_{j=1}^{n^s_{\mathrm{IP}}} \hat{\boldsymbol{x}}_j \psi_j(\xi_1,\xi_2)\,, \tag{4.1}$$

being a polynomial representation of the curved triangle or quadrilateral. Here, (ξ_1,ξ_2) are the parameter directions, $\hat{\boldsymbol{x}}_j$ are the coordinates of the interpolation points on the curved surface and $\psi_j(\xi_1,\xi_2)$ are the corresponding Lagrange basis functions. The definition of multivariate Lagrange basis functions is found in Section 3.1.

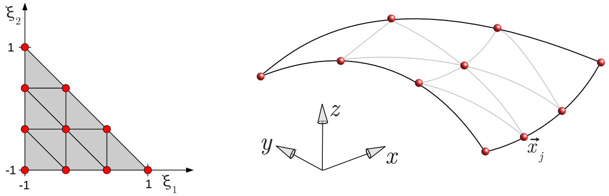


Figure 4.3.: Mapping from parameter space (ξ_1,ξ_2) to physical space using (4.1)

The number of interpolation points for triangles is $n^s_{\mathrm{IP}} = \frac{1}{2}(N_g + 1)(N_g + 2)$ and for quadrilaterals $n^s_{\mathrm{IP}} = (N_g + 1)^2$, where N_g is the polynomial degree of the mapping. In Fig. 4.3 we show the mapping for a triangle and $N_g = 3$. We choose a uniformly spaced node distribution in reference space to represent the polynomial of the mapping. If the interpolation points in physical space are uniformly spaced, a linear mapping is recovered. When using other point distributions or basis functions, like a Bezier basis, polynomial coefficients are easily found by applying a Vandermonde matrix. We want to point out again that this choice is only valid for low order approximations ($N_g \lessapprox 8$), since equidistant tend are oscillate for high polynomial degrees, and there is conditioning problem of the Vandermonde matrix as well. The leading error term of the approximation is the surface data. Unfortunately, except the ICEM©edge data (LGL nodes), all additional surface and volume interpolation points are spaced uniformly in physical space, limiting the maximum polynomial degree of the mapping.

4.2. Surface Reconstruction by Normal Vectors

The basic idea when reconstructing a curve by a set of points with spline interpolation is to enforce continuity at the point positions. The most popular is the cubic spline interpolation with a uniform knot sequence [41]. The curve is found by solving a tridiagonal linear equation system with a size of the number of interpolation points. In two dimensions, it is a simple and robust technique to reconstruct the boundary curves. The only way to extend this technique to three-dimensional boundary surfaces is to use structured point distributions that can be broken down into line-by-line interpolations. However, for unstructured meshes, a three-dimensional spline reconstruction of the surface points would require to solve large equation systems and in particular, a unique parametrization over element edges is difficult to achieve.

Thus a completely local approach is attractive, using only the normal vectors at the element corner nodes. The surface elements are then G^1 continuous at the corner nodes. It is a so-called *point-normal* approach proposed in [41]. In Stiller [92], rational Bezier curves from point-normals are constructed, being able to represent spheres, cylinders and cones exactly. In contrast, we restrict ourselves to non-rational cubic polynomials.

Clearly, the most difficult part is to provide the normal vector. In *HOPR*, analytical expressions for simple geometric shapes are included, and it is easy to add other user-defined functions. We also need to consider sharp edges at the intersections of surfaces, where two point-normals are defined. To find these edges, the user marks each curved boundary patch with a unique index as an additional information to the boundary conditions.

The point-normal assignment for complex geometries is realized via a CAD interface, which is presented inSection 4.2.1. If no CAD model model is available, a reconstruction of point-normals from the surface mesh is used, where the normal at a grid point is an average of the surrounding linear surface element normals. The accuracy of these averaged normals strongly depends on the surface mesh resolution, and the normal averaging degrades at the edges of the boundary surface.

To generate the curved surfaces from the point-normals, we start by constructing the edges of the boundary face as a cubic polynomial curve, defined by its end points $(\boldsymbol{p}_1, \boldsymbol{p}_2)$ and two tangential vectors $(\boldsymbol{t}_1, \boldsymbol{t}_2)$, and reads in Bezier form as

$$\begin{gathered}\boldsymbol{X}_b(t) = \sum_{i=0}^{3} \boldsymbol{b}_i B_i^3(t)\,, \quad B_i^N(t) = \left(\begin{array}{c} N \\ i \end{array}\right)(1-t)^{N-i} t^i\,, \quad t \in [0;1] \\ \boldsymbol{b}_0 = \boldsymbol{p}_1\,, \quad \boldsymbol{b}_1 = (\boldsymbol{p}_1 + \frac{1}{3}\boldsymbol{t}_1)\,, \quad \boldsymbol{b}_2 = (\boldsymbol{p}_2 - \frac{1}{3}\boldsymbol{t}_2)\,, \quad \boldsymbol{b}_3 = \boldsymbol{p}_2\,.\end{gathered} \tag{4.2}$$

The Lagrange interpolation points with uniform point spacing along the curve are simply found by interpolation of the Bezier curve

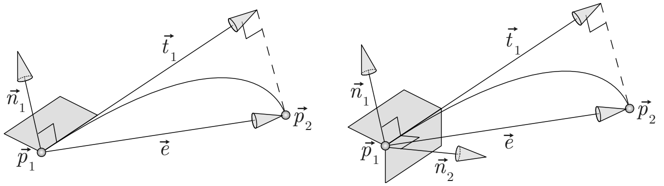
$$\begin{aligned}&\hat{\boldsymbol{x}}_0 = \boldsymbol{p}_1\,, \quad \hat{\boldsymbol{x}}_3 = \boldsymbol{p}_2\,, \\ &\hat{\boldsymbol{x}}_1 = \frac{1}{27}\left(20\boldsymbol{p}_1 + 4\boldsymbol{t}_1 - 2\boldsymbol{t}_2 + 7\boldsymbol{p}_2\right)\,, \quad \hat{\boldsymbol{x}}_2 = \frac{1}{27}\left(7\boldsymbol{p}_1 + 2\boldsymbol{t}_1 - 4\boldsymbol{t}_2 + 20\boldsymbol{p}_2\right)\,.\end{aligned} \tag{4.3}$$

The tangential vectors are constructed from the point-normals. As shown in see Fig. 4.4, the point-normal defines a tangential plane, and the tangential vector is found by projection of the straight edge $\boldsymbol{e}$ connecting the two edge points into the tangential plane. If two point-normals are given, the direction of the tangential vector is found by a cross-product and its length by projection of the straight edge onto the tangential vector.

$$\boldsymbol{e} = \boldsymbol{p}_2 - \boldsymbol{p}_1\,, \tag{4.4}$$

$$\boldsymbol{t}_1 = \boldsymbol{e} - (\boldsymbol{n} \cdot \boldsymbol{e})\boldsymbol{n}\,, \quad \text{for 1 normal,} \tag{4.5}$$

$$\boldsymbol{t}_1 = ((\boldsymbol{n}_1 \times \boldsymbol{n}_2) \cdot \boldsymbol{e})\,(\boldsymbol{n}_1 \times \boldsymbol{n}_2)\,, \quad \text{for 2 normals.} \tag{4.6}$$



(a) 1 point-normal (b) 2 points-normals

Figure 4.4.: Construction of tangential vectors (a) by projection and (a) by cross product.

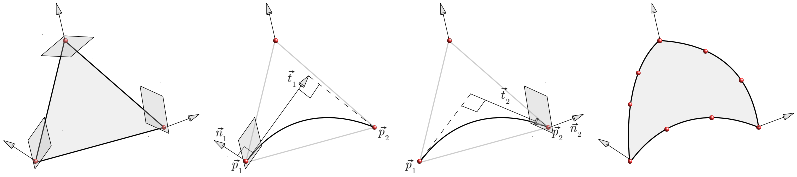


Figure 4.5.: Sequence of constructing curved element edges from point-normals.

As depicted in Fig. 4.5, the tangential vectors are constructed for all element edges and the interpolation points are computed using (4.3).
Goldapp [54] proved that when approximating a circular arc with a cubic polynomial, the optimal length of the tangential vector that minimizes the approximation error is

$$|\boldsymbol{t}\,|_{\text{opt}} = \frac{2}{1+\cos(\alpha)}|\boldsymbol{e}\,| \qquad \cos(\alpha) = \frac{\boldsymbol{t}\cdot\boldsymbol{e}}{|\boldsymbol{t}\,||\boldsymbol{e}\,|} \tag{4.7}$$

where α is the half angle of the circular arc, see Fig. 4.6. A tangential vector given, we derive a correction factor for the length of the tangential vector

$$f_{\text{tcorr}} = \frac{|\boldsymbol{t}\,|_{\text{opt}}}{|\boldsymbol{t}\,|} = \frac{2|\boldsymbol{e}\,|^2}{|\boldsymbol{t}\,||\boldsymbol{e}\,| + \boldsymbol{e}\cdot\boldsymbol{t}}\ . \tag{4.8}$$

The effect of the correction factor is investigated in Section 4.6.

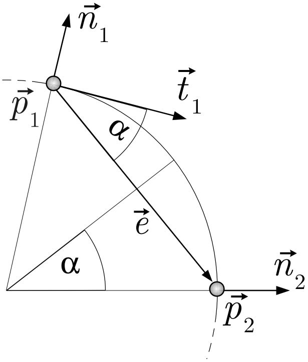


Figure 4.6.: Special case of a circular arc, where optimal tangential vectors can be derived.

4.2.1. CAD Interface

For a complex three-dimensional geometry, we want to use the CAD definitions to find the exact normal vector at the grid points of the linear surface mesh. The CAD interface was developed by Thomas Bolemann in his diploma thesis [19] and is written in Visual Basic 8 and uses the Microsoft .Net 2.0 framework. It is connected to the CAD software CATIA via a scripting interface. Commands to load a model, do geometric operations and extract CAD definitions are thus made directly accessible for the CAD tool.

The work-flow to assign the point-normals obtained from the CAD model is shown in Fig. 4.7. The CAD geometry is typically provided as a STEP file ('.stp'), a widely used standard exchange format [18, 63]. In general, grid generators are able to import the STEP file and provide a mesh with straight-edged elements and boundary conditions (BCs).

In a preliminary step, the CAD tool reads the 3D grid and extracts all grid points lying on boundaries, and even more important, the connectivity of the surface grid. Then the STEP file is loaded and the topology is analyzed, and for each CAD surface, a bounding box is created. For each CAD surface, the distance between the grid points and the CAD surface and its edges is measured to decide whether or not the grid point lies on the surface or edge. To minimize the computational effort, only grid points inside the bounding

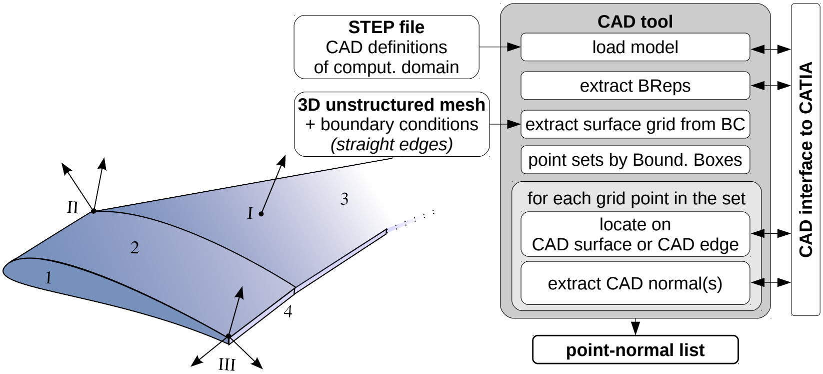


Figure 4.7.: Flowchart of the CAD tool to assign normal vectors from CAD.

box are considered. If the grid point lies on the surface (case I in Fig. 4.7), one normal vector is evaluated at the grid point. Multiple normals are found for CAD edges (case II) or corners (case III). A unique faceID is attached to every CAD surface, and each normal vector carries the faceID as a tag. The connectivity of the surface grid is important, because it helps to define local tolerances for the decision, whether the point lies on a surface, edge or corner, but also helps to find compound CAD faces, which were merged during mesh generation.

Finally, the output is a point-normal list, which includes the grid point index, the position, the number of normals at this point and the faceID for each normal. The data structure of *HOPR* matches this format, where a flexible list of normal vectors can be allocated for each grid point. The faceIDs are very important, since they facilitate curved edge construction, especially at sharp edges and corners.

4.2.2. Volume Mapping from Blending Curved Edges

The volume mapping is defined, following Fig. 4.8, by including the surface mapping of all element sides. The surface mapping in turn is defined by curved element edges. To construct a high order grid from curved edges and guarantee C^0 continuity of the high order grid, the following steps are important:

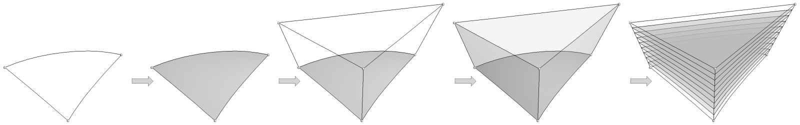


Figure 4.8.: Construction of mappings: from curved element edges to surface and volume mappings.

1. All element edges lying on the curved boundary are *uniquely* defined by a polynomial curve. The polynomial curve is represented by equally spaced interpolation points and Lagrange basis functions. The inner edges remain straight lines.

2. Triangle and quadrilateral mappings are also defined by Lagrange polynomials. The interpolation points on the edge remain the same, and inner points can be expressed in terms of the edge points using blending functions. This is done for all element sides having at least one curved edge.

3. The volume mapping is a blending of all element sides. It is again a polynomial, resulting in the full volume mapping $\boldsymbol{X}(\boldsymbol{\xi})$. The blending functions for the volume elements are found in Appendix D.

For element sides having at least one curved edge, we use a blending function to derive the curved element side. Blending three or four bounding curves yields to the so called *Coons* patches [41, 42]. The blending technique is shown in Fig. 4.9.

We define the curves $C(u,v)$, representing the boundary curves of the element, evaluated at the parameter value $u, v \in [0;1]$, where the relation to the reference coordinates is

$$u = \frac{1}{2}(\xi^1 + 1)\,, \quad v = \frac{1}{2}(\xi^2 + 1)\,. \tag{4.9}$$

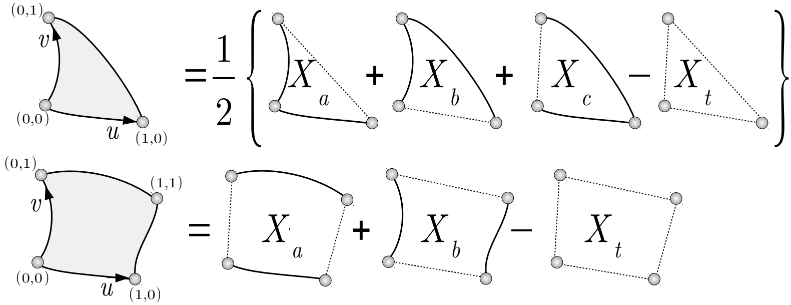


Figure 4.9.: Blending of curved element edges to a curved element side.

The triangular Coons patch according to [41] is defined as

$$\begin{aligned} X(u,v) &= \frac{1}{2}\left(X_a + X_b + X_c - X_t\right)\,, \quad \text{with} \\ X_a &= \frac{u}{u+v} C(u+v,0) + \frac{v}{u+v} C(0,u+v)\,, \\ X_b &= \frac{1-u-v}{1-v} C(0,v) + \frac{u}{1-v} C(1-v,v)\,, \\ X_c &= \frac{1-u-v}{1-u} C(u,0) + \frac{v}{1-u} C(u,1-u)\,, \\ X_t &= (1-u-v)C(0,0) + uC(1,0) + vC(0,1)\,, \end{aligned} \tag{4.10}$$

and the quadrangular Coons patch is a simple bilinear blending with

$$\begin{aligned} X(u,v) &= X_a + X_b - X_t\,, \quad \text{with} \\ X_a &= (1-v)C(u,0) + vC(u,1)\,, \\ X_b &= (1-u)C(0,v) + uC(1,v)\,, \\ X_t &= (1-u)(1-v)C(0,0) + u(1-v)C(1,0) \\ &\quad + (1-u)vC(0,1) + uvC(1,1)\,. \end{aligned} \tag{4.11}$$

Finally, to describe the surface with the polynomial (4.1), the unknown inner interpolation points are found by evaluating the Coons surface $X(u,v)$ at the corresponding position in parameter space. Note that the formula of the triangular coons patch can be evaluated at inner surface points only.

Elements are curved only when elements they are linked – e.g. sharing at least one edge – with the curved wall boundary ('local curving'). If the elements are highly stretched or skewed, additional curving of inner elements would be required ('global curving') to avoid negative or very small Jacobians. We found that using near-wall tetrahedra or pyramids generally leads to much smaller Jacobians than using prismatic extrusions of the surface grid, i.e. prisms and hexahedra.

4.3. Surface Curving with Refined Surface Data

Assuming that we have access to a grid generator to create a mesh of a given CAD model, the mesh and the CAD are internally connected to each other. Once the mesh is exported, this information is typically lost and we are left with a mesh with linear edges. The idea is now to extract additional information using features of the grid generators to generate a refined surface grid and use the additional grid points for interpolation.

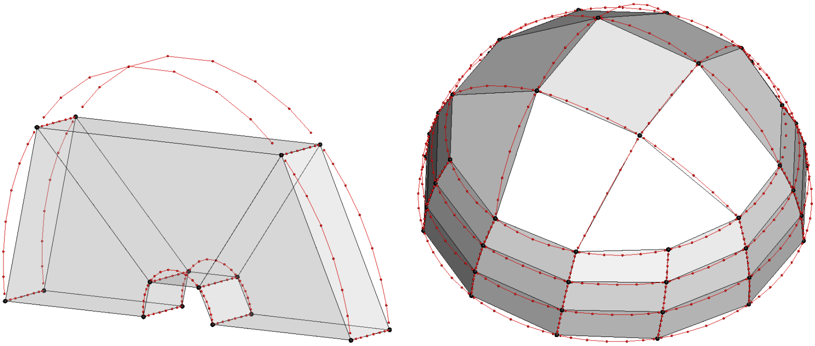


Figure 4.10.: Additional CL points on curved edges for a very coarse mesh of a cylinder and a sphere surface, generated with ICEM©.

In [1], a high order hexahedral grid with Chebychev-Lobatto (CL) point distribution is generated for a spectral element method using ICEM©. They started from a linear base mesh generated in ICEM© and were able to project CL point distributions of edges and surfaces onto the CAD faces via projection libraries of the ICEM© hexa module. Even though the projection libraries are not accessible anymore, a so-called *spectral elements* function is available to the curved boundary faces. One can export a file containing CL points for each edge of the surface mesh. The number of points can be chosen arbitrarily, two examples are shown in Fig. 4.10. The file format provides the edges' end point node IDs and the additional point coordinates. We can therefore associate the linear mesh edges with the curved edges, but, like for the normal vector strategy, we need to blend the curved edges to find the curved element faces as shown in the last section.

Another feature to produce high order surface data is surface mesh refinement. It is naturally supported by the grid generator ANSA©. In each splitting step, surface elements are isotropically refined by a factor of two, and the new grid points are reprojected to the geometry, keeping the original aspect ratio of the surface element. In Fig. 4.11, the original surface mesh and the refined mesh after two splitting steps are shown. Triangles and quadrilaterals are treated equally. Note that original mesh points remain unchanged. This property is important, since *HOPR* applies a point matching routine between the boundary faces of the original mesh and the refined surface mesh, using a fast multiple node detection algorithm explained in Appendix E.

Given the polynomial degree of the surface mapping ($N_g = 2, 4, 8$) corresponding to the number of refinement steps, an algorithm finds the connection between each boundary face and its refined surface elements by making use of the connectivity information of the refined surface mesh.

A sketch of the algorithm is shown in Fig. 4.12. It starts at the corner node of the boundary face and chooses an attached element on the refined surface, then loops over the neighbors in a structured manner to find the next matching corner nodes. We know that there is exactly one set of elements which matches all corner nodes, thus if the corner nodes are not reached, we know that we have to choose another starting element. The algorithm is very fast, there are typically only 3-5 elements attached at one corner node and surface elements are skipped once they are associated with a boundary face. Finally, we can use the additional nodes directly for the interpolation polynomial of the surface mapping defined in (4.1). Hence, in contrast to the ICEM© data, no blending is necessary.

We want to point out that any other grid generator which supports this type of isotropic surface mesh refinement can be used for that strategy. For example, we know that the software CUBIT [17] also provides that feature.

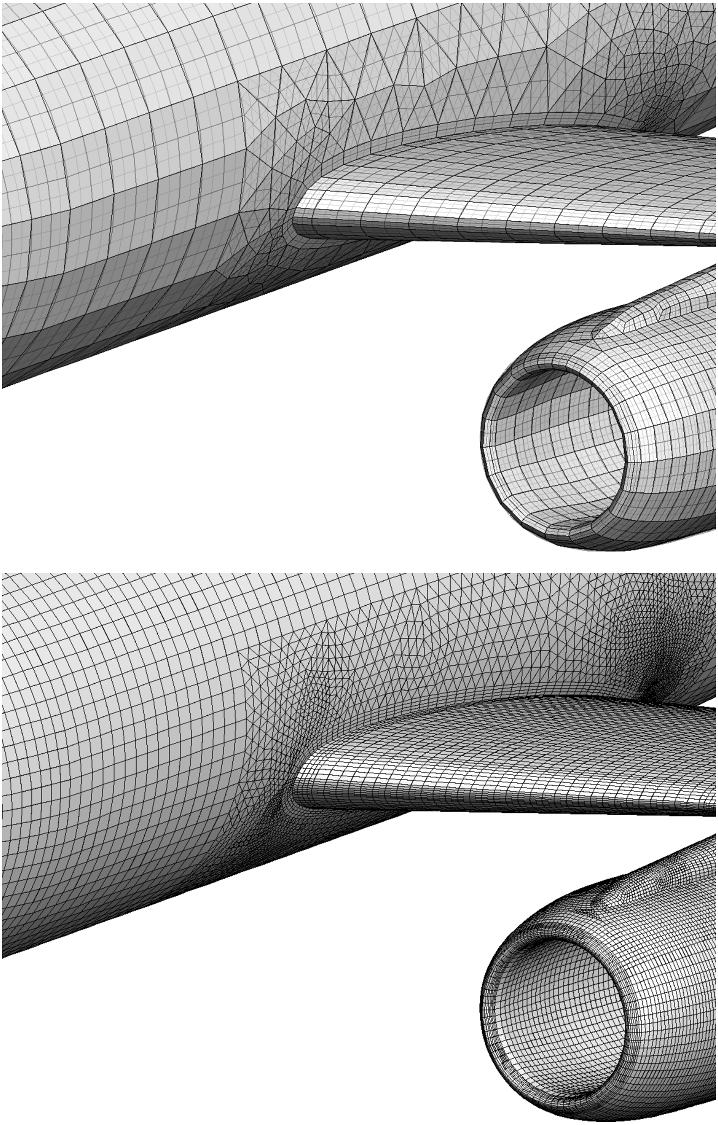


Figure 4.11.: Surface mesh of the DLR-F6 body-wing intersection, after two steps of isotropic refinement.

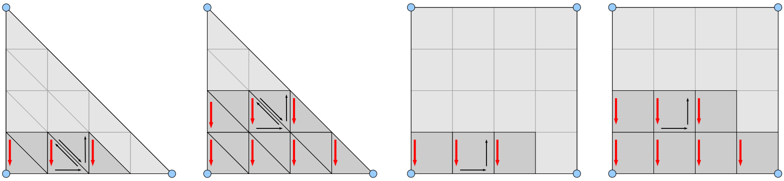


Figure 4.12.: Sketch of two stages of the algorithm to associate refined surface elements and coarse boundary faces, using the surface mesh connectivity, for a triangle and a quadrilateral.

4.4. Volume Curving using Block-Structured Meshes

A simple idea to produce fully curved elements is super-sampling. Like the surface refinement strategy for the surface curving, one could start from a refined volume mesh and *agglomerate* the elements to get the coarse DG mesh. However, the refinement process encounters the same problems as the curved mesh generation, since the meshes have to adapt to curved geometry and inner volume points need to be adjusted. For a general unstructured volume mesh, this is a cumbersome task. In contrast, it is very easy when working with block-structured meshes where the number of points can be simply increased. Assuming a fine 3D structured block of size $(I, J, K) = ([1; E_1], [1; E_2], [1; E_3])$ linear elements, we coarsen the mesh by the factor N_g. The curved hexahedral mapping of an element I, J, K, already introduced in (2.6), is found by interpolation

$$\boldsymbol{X}(\boldsymbol{\xi})^{IJK}_{(\mathrm{Hex})} = \sum_{ijk=0}^{N_g} \hat{\boldsymbol{x}}^{IJK}_{ijk}\, \ell_i(\xi)\ell_j(\eta)\ell_k(\zeta)\,, \tag{4.12}$$

where a uniform point distribution in reference space is chosen (see (2.5)) to guarantee that the mapping remains linear, if the physical node position is uniformly spaced, too. Given the grid points of the block, the mapping to the interpolation points is simply

$$\hat{\boldsymbol{x}}^{IJK}_{ijk} \mapsto \boldsymbol{x}^{\mathrm{block}}_{mno}\,, \quad \begin{array}{ll} m = (I-1)N_g + i\,, & m = 0,\dots,E_1 N_g \\ n = (J-1)N_g + i\,, & n = 0,\dots,E_2 N_g \\ o = (J-1)N_g + i\,, & o = 0,\dots,E_3 N_g \end{array} . \tag{4.13}$$

In Fig. 4.13 and Fig. 4.14, two examples of the curved mesh from agglomeration with a polynomial degree of $N_g = 4$ are depicted.

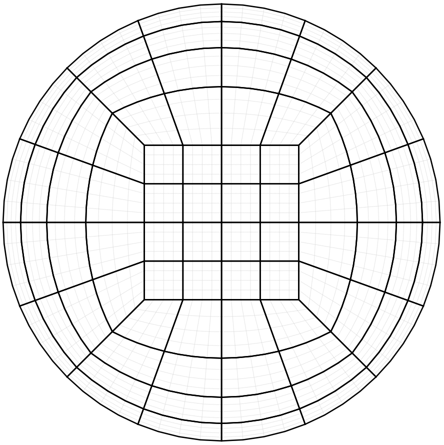


Figure 4.13.: Example of an agglomerated structured grid ($N_g = 4$).

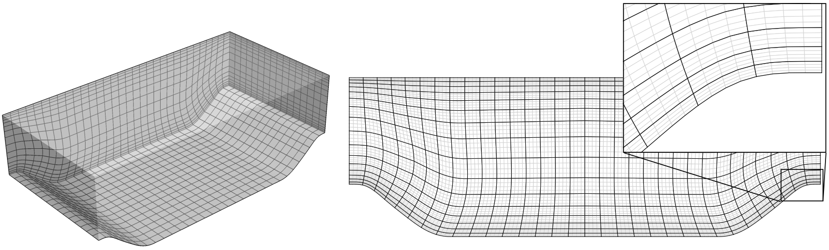


Figure 4.14.: Curved mesh of the periodic hill test case from an agglomerated structured grid ($N_g = 4$), 3D view (left) and front view (right) with initial grid lines shown in gray.

Note that if the mesh refinement is not possible, the structured point data still is valuable information. From the points of each line of a structured block, we can compute a cubic spline interpolation curve to construct additional interpolation points. Sequentially applied in each structured direction to all lines, a fully curved mesh is generated. Two additional points inside each segment are sufficient to exactly represent the cubic spline segment, yielding in a polynomial degree of $N_g = 3$ for the element mapping. The quality of the curved mesh generated from the spline interpolation will not only depend on the resolution of the geometry but also on the block size.

4.4.1. Quality of the Interpolation of Stretched Meshes

In structured volume meshing, all blocks are connected, and therefore refinement or coarsening of the mesh is mainly done by stretching the mesh. The grid for a boundary layer is typically stretched. If we want to apply the agglomeration approach on a stretched grid, we have to be careful of the strength of the stretching, since element mapping may deteriorate by the interpolation. In this section, we quantify the strength of the stretching to yield a high quality mesh. As a model problem for a stretched grid, we investigate the interpolation of an exponentially stretched grid in one dimension, where each element size Δx_i is determined by a constant growth rate f between two consecutive elements

$$f := \frac{\Delta x_{i+1}}{\Delta x_i} = \text{const.} \tag{4.14}$$

Without loss of generality, we fix $x \in [0,1]$, and the smallest elements are found at $x = 0$. The range of elements is $i = 1, \ldots, n_e$. Each grid point can be computed by

$$x_j = x_0 + \sum_{i=1}^{j} \Delta x_i \,, \tag{4.15}$$

with $x_0 = 0$ and $j = 1, ..., n_e$. The grid spacings are found by

$$x_{n_e} = 1 = \sum_{i=1}^{n_e} \Delta x_i = \Delta x_1 \sum_{i=1}^{n_e} f^{i-1} = \Delta x_1 \frac{f^{n_e} - 1}{f - 1} \quad \Rightarrow \quad \Delta x_1 = \frac{f-1}{f^{n_e} - 1} \tag{4.16}$$

where we used the relation for a finite geometric series. Thus, the grid point coordinate is described by

$$x_j = \Delta x_1 \sum_{i=1}^{j} f^{i-1} = \frac{f^j - 1}{f^{n_e} - 1} = \frac{c^{\frac{j}{n_e}} - 1}{c - 1} \tag{4.17}$$

with $c = f^{n_e}$. For $(n_e \rightarrow \infty)$, (4.17) can be interpreted as a continuous stretching function

$$x(r) = \frac{c^r - 1}{c - 1}\,, \quad \text{with} \quad c := f^{n_e}\,, \tag{4.18}$$

which maps a uniform point distribution in $r \in [0,1]$ to a stretched physical point distribution in $x \in [0,1]$, with the same boundary points.
Figure 4.15 shows a one-dimensional fine boundary layer grid, where only every fourth grid node $(x_0, x_4, x_8, ...)$ is used to define the DG element boundaries. Together with 3 internal nodes, an element mapping of polynomial degree $N_g = 4$ is defined. The element mapping from (4.12) simplifies in one dimension to

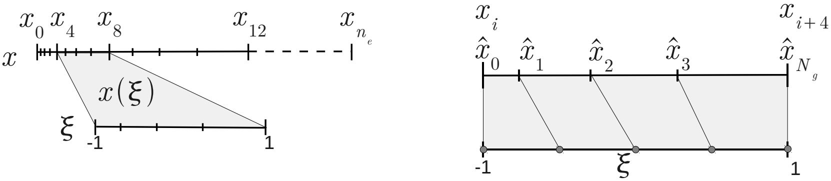


Figure 4.15.: Example of a stretched DG grid derived from a finer grid (left), where the sub-grid is used for the element mapping (right).

$$x(\xi) = \sum_{j=0}^{N_g} \hat{x}_j\, \ell_j(\xi)\,, \tag{4.19}$$

where $\{\hat{x}_k\}_{i=0}^{N_g}$ are physical grid point positions.
From the analytical stretching function (4.18), we can derive the analytical mapping of the high order element with the reference coordinate $\xi \in [-1,1]$. An arbitrary element e starts at a grid position i, and ends at grid position $(i + N_g)$

$$r^{(e)}(\xi) := r_i + (r_{(i+N_g)} - r_i)\frac{1}{2}(\xi + 1) = \frac{i}{n_e} + \frac{N_g}{n_e}\frac{1}{2}(\xi + 1) \tag{4.20}$$

$$x^{(e)}(\xi) = x(r(\xi)) = := \frac{c^{\frac{i}{n_e}}\, c^{N_g \frac{1}{2}(\xi+1)} - 1}{c - 1} = \frac{f^{\frac{i}{n_e}}\, f^{N_g \frac{1}{2}(\xi+1)} - 1}{f^{n_e} - 1}\,. \tag{4.21}$$

Using a linear transformation

$$x^*(\xi) := \frac{x^{(e)}(\xi) - x^{(e)}(-1)}{x^{(e)}(1) - x^{(e)}(-1)} \qquad x^* \in [0,1] \tag{4.22}$$

all element mappings can be generalized, and the internal 'shape' of all elements

$$x_i^*(\xi) := \frac{\left(f^{N_g}\right)^{\frac{1}{2}(\xi+1)} - 1}{\left(f^{N_g}\right) - 1}\,, \quad i = 1, ..., n_e\,, \tag{4.23}$$

depends only on the growth rate f and the polynomial degree of the element. It is clear that the quality of the interpolated element mapping depends strongly on the polynomial degree and the magnitude of the stretching factor. Even though the analytical mapping is monotone, monotonicity is not guaranteed for the interpolated mapping and its derivatives. In the worst case, the approximation of the mapping by a polynomial can lead to negative Jacobians. Thus, we will evaluate the quality and the limits of the polynomial approximation by interpolating the analytical stretching function (4.23) at uniformly spaced points for an increasing growth rate f and different polynomial degrees N_g.

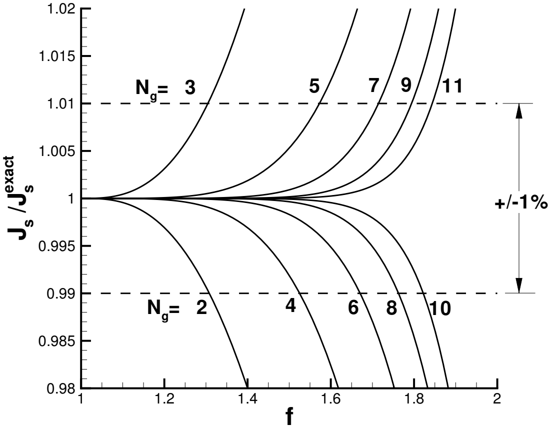


Figure 4.16.: Quality of the interpolated element mapping for polynomial degrees $N_g = 2, ..., 11$, over the fine grid growth rate f_{fine}.

An element mapping is only valid if the Jacobian is positive. This requirement however is not sufficient to describe the quality of the element. To furthermore assess the one-dimensional element Jacobian, the so-called *scaled Jacobian* is introduced

$$J_s := \frac{\min(J(\xi))}{\max(J(\xi))}\,, \tag{4.24}$$

with $J_s = 1$ for a constant Jacobian, i.e. a linear mapping. For the exact mapping function (4.23), the scaled Jacobian is $J_s^{\text{exact}} = f^{-N_g}$. Therefore, to compare the approximation for all stretching parameters, we normalize the scaled Jacobian of the interpolation mapping by the exact scaled Jacobian. We plot the normalized scaled Jacobian in Figure 4.16 for $N_g = 2, ..., 11$ over the fine grid growth rate f.

$\frac{J_s}{J_s^{\text{exact}}}$	$N_g = 2$	3	4	5	6	7	8	9	10	11
	$f^{\max}$									
< 0	3.00	4.79	2.62	3.36	2.46	2.92	2.37	2.70	2.31	2.57
±10%	1.75	1.71	1.91	1.93	1.99	2.00	2.02	2.03	2.04	2.04
±1%	1.31	1.30	1.53	1.57	1.67	1.71	1.76	1.79	1.82	1.84
	$f_{\text{DG}}^{\max}$									
< 0	9.00	109	47.2	430	222	1801	1002	7600	4401	31892
±10%	3.07	5.02	13.4	27.1	61.5	129	279	588	1242	2601
±1%	1.71	2.22	5.41	9.60	21.6	43.4	92.9	192	403	833

Table 4.1.: Maximum growth rate $f^{\max}$ for increasing mapping quality of the scaled Jacobian J_s, normalized by $J_s^{exact} = f^{-N_g}$ and corresponding growth rate between DG elements is f_{DG}.

We can define a quality for the approximation, for example ±1% deviation in the scaled Jacobian, which leads to a limiting growth rate $f^{\max}$, listed in Table 4.1. It is interesting to note that for the grid generation, the underlying fine grid should never exceed a growth rate of ≈ 2 for a good and valid grid for all polynomial degrees. If we look at the final high order mesh, the growth rate from one element to the next is $f_{DG} = f^{N_g}$, also given in Table 4.1. hence, with increasing polynomial degree of the mapping larger growth rates between the resulting DG elements can be realized.

As a final remark, coarsening an initially fine mesh by skipping intermediate points also increases the stretching factor. For example, skipping every second point leads to a stretching factor

$$f_{\text{coarse}} = \frac{\Delta x_{i+2} + \Delta x_{i+3}}{\Delta x_i + \Delta x_{i+1}} = \frac{f^2(\Delta x_i + f\Delta x_i)}{\Delta x_i + f\Delta x_i} = f^2\,. \tag{4.25}$$

Generally, using only every n^{th} point of the initial mesh, the stretching factor on the coarse grid yields $f_{\text{coarse}} = f^n$. As already mentioned, the resulting factor should be $f_{\text{coarse}} \leq 2$ for a valid mapping.

4.5. Volume Curving by Mesh Deformation

Boundary layer meshes typically have very high aspect ratios near a wall, and if the wall is curved, a curved mesh generated from linear meshes and surface curving may lead to inverted elements, see Fig. 4.17. A curved boundary layer mesh is easy to construct with the agglomeration approach from Section 4.4 and should always be considered if a structured mesh generation is possible. Fairly complex geometries are meshed nowadays with block-structured meshes, however structured meshing remains a time-consuming task, and geometric restrictions often lead a large number of additional elements. As the unstructured meshes available have only straight element edges, the surface curving must be propagated into the domain.

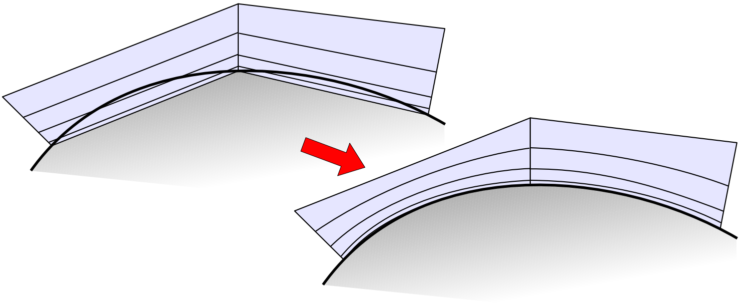


Figure 4.17.: The curved surface intersects the inner edges of a linear boundary layer mesh, and inner elements must be curved for a valid mesh.

Provided that near-wall elements are simple prismatic extrusions of the surface mesh (prisms or hexahedra), a given stack of elements can be found from the mesh connectivity information and curved via a transfinite interpolation between the lower and upper surface. However, complex meshes may involve non-prismatic near-wall elements, so a more general approach is required. The mesh deformation issue is not new.

It is found in applications involving fluid-structure interaction, where volumetric fluid meshes have to adapt to a deformation of the boundary surface to guarantee a valid mesh. For linear unstructured meshes, different approaches are found in literature and can be divided into two main strategies. The first uses radial basis functions [37] or continuous approaches exploiting the internal mesh connectivity, from simple spring models [38] to Laplacian or biharmonic operators [11, 57]. The second considers the mesh as a deformable solid and therefore prevents the elements to invert by construction. The second approach was first applied successfully for curving of high order meshes by Persson and Peraire [84], who formulate a high order FE scheme with a non-linear material law to solve for the inner element deformations. We will adopt the same strategy and describe the details in the remainder of this section.

We start with a second order system of equations

$$\nabla_{\boldsymbol{X}} \cdot \boldsymbol{P}(\nabla_{\boldsymbol{X}} \boldsymbol{u}) = 0\,, \tag{4.26}$$

with the solution vector of the displacement $\boldsymbol{u} = (u_1, u_2, u_3)$ and the second order tensor $\boldsymbol{P} = (P_1, P_2, P_3)^T$ with the components of each vector $P_i = (p_{i1}, p_{i2}, p_{i3})$. The notation of the tensor helps us to keep the same transformation from Chapter 2 to express the equation in reference space. We also introduce the undeformed mesh position $\boldsymbol{X}$, the displacement $\boldsymbol{u}$ and the deformed mesh position $\boldsymbol{x}$

$$\boldsymbol{x} = \boldsymbol{X} + \boldsymbol{u}\,. \tag{4.27}$$

Typically, the undeformed mesh position is the initial mesh with linear edges. We have to solve the system (4.26) with a high order scheme. We limit this investigation to quadrilateral meshes and therefore use the continuous Galerkin spectral element method (CG-SEM), which guarantees C^0 continuity across element interfaces by construction. This property is very important, since it directly produces conformal elements. The implementation is similar to the DG-SEM method, mainly involving local element operations. A detailed description of implementing CG-SEM is found in the book of Kopriva [71]. The derivations in this section are based on that book. We assume that we already applied the surface curving. Hence, we choose Dirichlet boundary conditions to impose the displacement as the difference between the linear and the curved surface

$$\boldsymbol{u}|_{\partial\Omega} = \boldsymbol{x}_{\text{curved}}|_{\partial\Omega} - \boldsymbol{X}|_{\partial\Omega}\,, \tag{4.28}$$

which is applied discretely at the interpolation points, see Fig. 4.18.

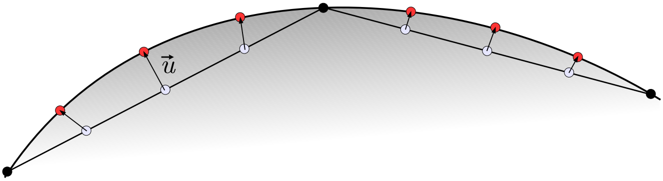


Figure 4.18.: Displacement vector for the Dirichlet boundary condition, being the difference between the linear and the curved boundary interpolation points.

Other non-curved boundaries are clamped by setting $\boldsymbol{u}|_{\partial\Omega} = 0$. We transform the equation to reference space, yielding

$$\begin{aligned} \boldsymbol{\nabla}_{\boldsymbol{\xi}} \cdot \tilde{\boldsymbol{P}} = 0\,, \quad \tilde{\boldsymbol{P}} = (\tilde{P}^1, \tilde{P}^2, \tilde{P}^3)^T\,, \\ \tilde{P}^n = (J\boldsymbol{a})^n \cdot \boldsymbol{P}(\boldsymbol{\nabla}_{\boldsymbol{x}}\boldsymbol{u})\,, \quad \boldsymbol{\nabla}_{\boldsymbol{x}}\boldsymbol{u} = \frac{1}{J}\sum_{n=1}^{3} \frac{\partial}{\partial \xi^n}\left((J\boldsymbol{a})^n \boldsymbol{u}\right)\,. \end{aligned} \tag{4.29}$$

where we used the metric terms (2.17) and the conservative form of the gradient (2.22) and the divergence (2.21). They are equal to the non-conservative form if the metric identities (2.25) hold. We will not explicitly derive the discretization of (4.29) with CG-SEM. The derivations are similar to DG-SEM, see Section 3.3. We use tensor-product Lagrange basis functions with Gauss-Lobatto points for the solution approximation, and we collocate the fluxes and the integration points, leading to a diagonal mass matrix and dimension-by-dimension element operators. More details on the derivation of CG-SEM are found in [71]. The discrete weak form of (4.29) yields the residual for each element (e)

$$\begin{aligned} R^{(e)}_{ijk} = \omega_i\omega_j\omega_k \left(\sum_{l=0}^{N} \hat{D}_{il}\tilde{P}^1_{ljk} + \hat{D}_{jl}\tilde{P}^2_{ilk} + \hat{D}_{kl}\tilde{P}^3_{ijl}\right)\,, \\ \tilde{P}^n_{ijk} = (J\boldsymbol{a})^n_{ijk} \cdot \boldsymbol{P}(\boldsymbol{\nabla}_{\boldsymbol{x}}\boldsymbol{u}^{(e)}_{ijk})\,. \end{aligned} \tag{4.30}$$

In [71], Kopriva develops an attractive way to impose the C^0 continuity at the element boundary nodes. Three routines called *mask*, *unmask* and *globalSum* represent the only global operations involving the element connectivity. The residual is always computed element by element, and the data structure for solution and residual remains element local. During initialization, the mesh connectivity is exploited to define unique master corner nodes, master edges and master faces, and all other nodes on the element boundaries are set to slaves.
The mask routine (local to global) sets the residual of all slave nodes to zero, the unmask routine (global to local) copies the solution from the master to the slaves, and the global sum adds the residual of slave nodes to the master. A residual evaluation involves the following steps:

$$\begin{aligned} \textit{Unmask: } U^{(e)} \Leftrightarrow U^{(e)}\,, &\quad \text{local residual: } R^{(e)}(U^{(e)})\,, \\ \textit{GlobalSum: } R^{(e)} \Leftrightarrow R^{(e)}\,, &\quad \textit{mask: } R^{(e)}\,. \end{aligned}$$

Rearranging all nodes of the domain to a vector, we write the resulting discrete equation system as

$$\underline{R}(\underline{\boldsymbol{u}}) = 0\,. \tag{4.31}$$

This is solved with the Newton method

$$\underline{\boldsymbol{u}}^{k+1} = \underline{\boldsymbol{u}}^{k} + \Delta\underline{\boldsymbol{u}}\,, \tag{4.32}$$

$$\underline{R}(\underline{\boldsymbol{u}}^{k}) + \left(\frac{\partial \underline{R}}{\partial \underline{\boldsymbol{u}}}\right)^{k} \Delta\underline{\boldsymbol{u}} = 0\,. \tag{4.33}$$

In each iteration, we solve the linear equation system (4.33) via the conjugate gradient method. The algorithm does not need the system matrix explicitly, but only the action of the matrix on a vector. If the equation is linear, the Newton iteration is not needed, and the matrix-vector product simplifies to $\underline{R}(\underline{\boldsymbol{u}}^0)$. For non-linear equations, we introduce a standard finite difference linearization of the matrix-vector product [24]

$$\left(\frac{\partial \underline{R}}{\partial \underline{\boldsymbol{u}}}\right)^{k} \Delta\underline{\boldsymbol{u}} \approx \frac{1}{\varepsilon}\left(\underline{R}(\underline{\boldsymbol{u}}^{k} + \varepsilon\Delta\underline{\boldsymbol{u}}) - \underline{R}(\underline{\boldsymbol{u}}^{k})\right)\,, \tag{4.34}$$

which involves only residual evaluations.
In addition, if the material is non-linear, the deformation at the boundary cannot be applied at once but in load increments so that elements remain valid after the initial boundary deformation. We also recommend a transfinite

interpolation of the initial deformation into the first element layer to improve the robustness and the convergence speed.

The choice of the second order tensor $\boldsymbol{P}$ will result in different models for the mesh deformation. Equation (4.26) yields the Laplace equation when choosing

$$p_{ij} = \frac{\partial u_j}{\partial X_i}\,, \quad \Rightarrow \quad \boldsymbol{\Delta}_{\mathbf{X}} U = 0. \tag{4.35}$$

It is the simplest model and the equation is linear.

In non-linear continuum mechanics, the tensor $\boldsymbol{P}$ refers to the first Piola-Kirchhoff stress tensor. The book by Bonet and Wood [20] describes several non-linear material laws, one of them is the compressible neo-Hookean material, which was used in [84] for the mesh deformation. The tensor is

$$\boldsymbol{P} = \left(\mu(\mathsf{F} - \mathsf{F}^{-T}) + \lambda J_{\mathsf{F}} \mathsf{F}^{-T}\right), \tag{4.36}$$

where F denotes the deformation tensor

$$\mathsf{F}_{ij} = \frac{\partial x_j}{\partial X_i} = \delta_{ij} + \frac{\partial u_j}{\partial X_i}\,, \tag{4.37}$$

and $J_{\mathsf{F}} = \det(\mathsf{F})$ is the determinant, μ is the shear modulus and λ Lamé's first parameter. They are related to Young's modulus E and the Poisson ratio $0 < \nu < 0.5$ by

$$\mu = E\frac{\nu}{(1+\nu)(1-2\nu)}\,, \quad \lambda = E\frac{1}{2(1+\nu)}\,. \tag{4.38}$$

In fact, the only parameter changing the way the boundary deformation is propagated is the Poisson ratio, since the Young's modulus is simply a constant factor. Persson and Peraire [84] choose $\nu = 0.4$. Bonet and Wood [20] have many other material laws for large deformations, but we will restrict ourselves to the linear Laplace equation and the non-linear compressible neo-Hookean law. Even though not investigated here, locally varying material coefficients were used in [45] to improve the mesh quality for large mesh deformations.

To show its applicability, we apply the mesh deformation method to a sequence of two-dimensional boundary layer meshes of the NACA0012 airfoil, a coarse mesh with $n_e = 4$ elements along the chord and two refined levels created by doubling the number of elements in each direction ($n_e = 8, 16$ elements). The total number of elements is $n_{e,\text{total}} = 288, 1152, 4608$, but we will refer to the meshes with the number of elements along the chord.

The initial linear meshes are depicted in Fig. 4.19, together with the curved airfoil geometry, for $N_g = 4$. The zoom into the boundary layer shows that

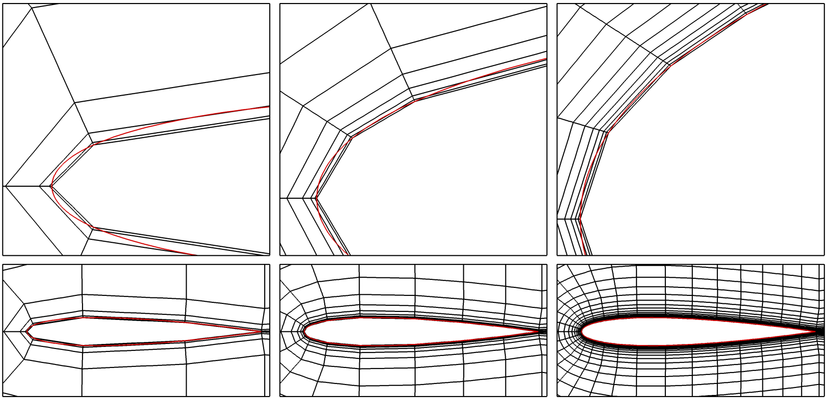


Figure 4.19.: Sequence of linear boundary layer meshes for the NACA0012 airfoil, curved boundary ($N_g = 4$) in red.

the curved surfaces lies within 2-3 boundary layer elements for all mesh levels. As the polynomial degree of the curved surface is $N_g = 4$, it is convenient to choose the same polynomial degree for the solution of the mesh deformation. The Dirichlet boundary condition for the displacement is applied at each interpolation node as the displacement vector between the linear and the curved nodes, see Fig. 4.18.

The Laplace equation is the easiest to solve, since we only need to solve the linear system with the conjugate gradient method and a simple diagonal preconditioning from [71]. In Fig. 4.20, the deformed curved mesh is depicted. The deformation is propagated into the mesh volume and all elements become valid. Here, we did not clamp the element corner nodes, and thus the deformation also pushes the elements away from the boundary. if this is not desired the element corner nodes can be clamped easily, we only need to nullify the residual at corner codes, in the *mask* routine. There are two advantages, first the solver converges much faster, and the deformation propagates further into the domain yielding a better distribution of the mesh curving, see Fig. 4.21. However, as additional constraints are introduced, strong deformation gradients can be produced if the corner nodes are too close or not well aligned with the geometry.

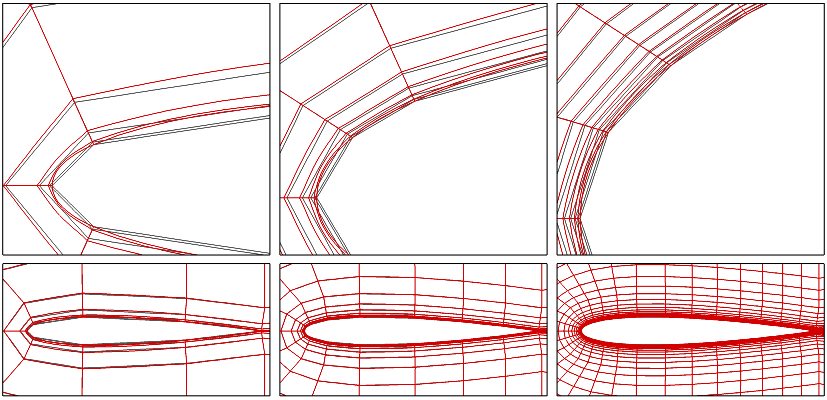


Figure 4.20.: Curved boundary layer meshes (red lines) after deformation with Laplace equation and free corner nodes. Initial linear mesh in gray.

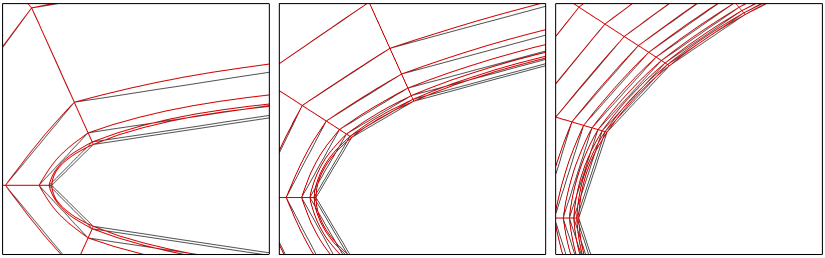


Figure 4.21.: Curved boundary layer meshes (red lines) after deformation with Laplace equation and clamped corner nodes. Linear mesh shown in gray.

In Fig. 4.22, we show the effect of the material model. The non-linear neo-Hookean material (4.36) shows a different propagation of the deformation. We can increase the depth of the deformation propagation by increasing the Poisson ratio $\nu \to 0.5$, and we recover the results of the Laplace equation for $\nu \to 0$.

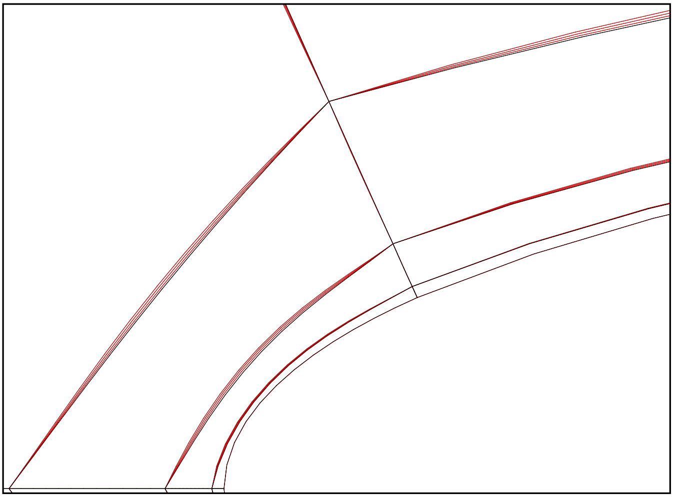


Figure 4.22.: Comparison of the mesh deformation using the Laplace equation (black lines) and the neo-Hookean material law with increasing Poisson ratio $\nu = 0.3, 0.4, 0.45$ (red lines), with clamped corner nodes.

A useful measure for quality and validity of a curved mesh is the normalized or scaled Jacobian [84,87], already introduced in (4.24). It measures the variation of the Jacobian $J(\boldsymbol{\xi})$ inside an element Q

$$J_s = \frac{\min_{\boldsymbol{\xi} \in Q} J(\boldsymbol{\xi})}{\max_{\boldsymbol{\xi} \in Q} |J(\boldsymbol{\xi})|} \, . \tag{4.39}$$

For linear element mappings the Jacobian is constant, so the scaled Jacobian equals 1. The scaled Jacobians range from -1 to 1, but only elements with $J_{scaled} > 0$ are valid, quality increases with $J_{scaled} \to 1$. Note that the criterion applies to the initial linear mesh, too, since Jacobians vary inside quadrangular or hexahedral elements with non-parallel edges. The scaled Jacobian is computed by super-sampling the derivatives of the element mapping. Our definition is slightly different to the one found in [84,87], which relates the deformed element to the one with linear edges. In their definition, the Jacobian of the mapping in (4.39) is replaced with the determinant of the deformation tensor $J_{\mathsf{F}} = \det(\mathbf{F})$, thus equals one for all elements with linear edges.

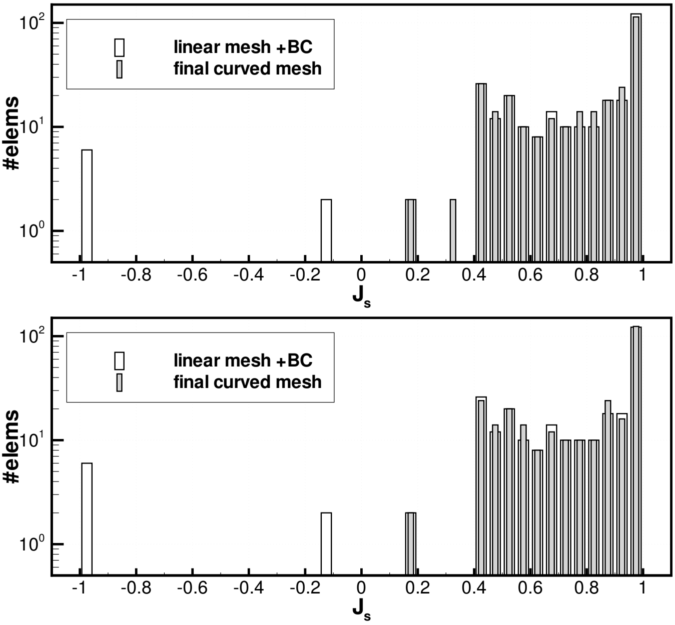


Figure 4.23.: Distribution of the scaled Jacobian, before and after the mesh deformation with clamped corner nodes, using the Laplace equation (top) and the neo-Hookean material with $\nu = 0.45$ (bottom).

We plot distribution of the number of elements belonging to a range of the scaled Jacobians (4.39) in Fig. 4.23. The corner nodes are clamped and we show the results for the coarse mesh $n_e = 4$. The transfinite interpolation of the boundary deformation leaves the 8 boundary elements inverted. The application of the mesh deformation pushes the elements back and yields positive Jacobians of the whole mesh, for both the Laplace equation and the neo-Hookean material. We summarize the results of all meshes and all models in Table 4.2 and look at the difference between the scaled Jacobian distribution of the final curved and the initial linear mesh. The linear elements at the boundary all have $J_s > 0.7$. After the application of the boundary deformation, negative Jacobians are found. The final deformed mesh pushes them back into the range $0.2 < J_s < 0.9$. The quality of the mesh increases for the neo-Hookean material law compared to the Laplace equation. Also a higher Poisson ratio has an overall positive effect and the mesh quality is higher if the element corner nodes are clamped.

The total number of iterations to solve the linear system with the diagonal preconditioned CG method is listed in Table 4.3 for the linear and the non-linear material. In the non-linear case, we need to apply the deformation in load increments and apply Newton's method. We choose 40 load increments for all cases and the Newton solver converges within three steps. The Poisson ratio influences the stiffness of the equation system, and the number of iterations increases for $\nu \to 0.5$. The use of clamped corner nodes always reduces the number of iterations. The number of iterations decreases slightly for the finer meshes. The computational cost for each iteration, however, scales with the number of elements. The CPU time ranges between $1-8$ seconds for the linear and between $5-65$ seconds for the non-linear material. In a full 3D configuration, we can reduce the number of elements by choosing only a certain amount of neighbors of the elements at the curved surfaces, leaving the elements far from boundaries undeformed.

We showed that using the mesh deformation, all models provide a valid curved mesh. The mesh quality slightly increases for the neo-Hookean material, especially for high Poisson ratios, compared to the Laplace equation, but also involves a more expensive non-linear solution procedure. We recommend to clamp the element corner nodes, since clamping improves the mesh quality and speeds up the solution process.

As long as using a material law and the solver converges, negative Jacobians are avoided by construction, and if the solver fails, the mesh remains invalid. In addition, positivity is not a quality measure, and small scaled Jacobians can lead to inferior solution approximation and smaller time steps. To remedy this issue, the GMSH research group around J.F. Remacle recently published in [87] a very promising multiple objective optimization strategy, where the positivity of the Jacobian and the timestep restriction are expressed as functionals of the curved node positions.

$J_s >$	-1	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
$J_s \leq$	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
$\boldsymbol{n_e = 4}$											
linear mesh	0	0	2	0	0	38	30	22	22	28	146
only boundary	**+8**	0	0	0	0	0	0	0	**-2**	0	**-6**
free corner nodes											
Laplace	0	0	0	**+2**	0	**+2**	0	**+2**	0	**+6**	**-12**
neo-H., $\nu = 0.3$	0	0	0	0	**+2**	**+2**	0	0	0	**+8**	**-12**
neo-H., $\nu = 0.4$	0	0	0	0	**+2**	0	**+2**	**-2**	**+2**	**+6**	**-10**
neo-H., $\nu = 0.45$	0	0	0	0	0	**+2**	**+2**	**-2**	-1	+5	**-6**
clamped corner nodes											
Laplace	0	0	0	0	**+2**	**+2**	0	**-2**	**+2**	**+4**	**-8**
neo-H., $\nu = 0.3$	0	0	0	0	**+2**	**+2**	0	**-2**	0	**+4**	**-6**
neo-H., $\nu = 0.4$	0	0	0	0	0	**+2**	**+2**	**-2**	**-2**	**+6**	**-6**
neo-H., $\nu = 0.45$	0	0	0	0	0	0	**+4**	**-2**	**-2**	**+6**	**-6**
$\boldsymbol{n_e = 8}$											
linear mesh	0	0	0	2	0	4	6	142	160	146	692
only boundary	**+12**	**+2**	**+2**	0	0	0	0	0	0	**-4**	**-12**
free corner nodes											
Laplace	0	0	0	0	**+2**	**+2**	0	**+4**	**+6**	**+2**	**-16**
neo-H., $\nu = 0.3$	0	0	0	0	0	**+2**	**+2**	**+4**	**+4**	**+2**	**-14**
neo-H., $\nu = 0.4$	0	0	0	0	0	0	**+2**	**+2**	**+6**	-1	-9
neo-H., $\nu = 0.45$	0	0	0	0	0	0	0	**+4**	**+4**	**-4**	**-4**
clamped corner nodes											
Laplace	0	0	0	0	0	0	**+4**	**+2**	**+1**	**+3**	**-10**
neo-H., $\nu = 0.3$	0	0	0	0	0	0	**+2**	**+2**	0	**+2**	**-6**
neo-H., $\nu = 0.4$	0	0	0	0	0	0	0	**+2**	**+2**	**-4**	0
neo-H., $\nu = 0.45$	0	0	0	0	0	0	0	**-2**	**+4**	**-2**	0
$\boldsymbol{n_e = 16}$											
linear mesh	0	0	0	0	2	2	4	14	130	1138	3318
only boundary	**+20**	0	**+2**	0	**+2**	**+2**	**+6**	0	0	**-2**	**-30**
free corner nodes											
Laplace	0	0	0	0	0	0	0	**+2**	**+8**	**+5**	**-15**
neo-H., $\nu = 0.3$	0	0	0	0	0	0	0	0	**+8**	**+8**	**-16**
neo-H., $\nu = 0.4$	0	0	0	0	0	0	0	0	**+6**	**+8**	**-14**
neo-H., $\nu = 0.45$	0	0	0	0	0	0	0	0	**+2**	**+10**	**-12**
clamped corner nodes											
Laplace	0	0	0	0	0	0	0	0	**+8**	0	**-8**
neo-H., $\nu = 0.3$	0	0	0	0	0	0	0	0	**+6**	**+2**	**-8**
neo-H., $\nu = 0.4$	0	0	0	0	0	0	0	0	**+2**	**+8**	**-10**
neo-H., $\nu = 0.45$	0	0	0	0	0	0	0	0	0	**+4**	**-4**

Table 4.2.: Difference of the scaled Jacobian distributions between the final curved and the initial linear mesh, for different deformation models and three meshes $n_e = 4, 8, 16$.

	$n_e = 4$		$n_e = 8$		$n_e = 16$	
corners:	free	clamped	free	clamped	free	clamped
Laplace	221	51	191	81	104	57
neo-H., $\nu = 0.3$	5015	2374	5287	3389	2740	1995
neo-H., $\nu = 0.4$	6468	2910	6737	4023	3304	2304
neo-H., $\nu = 0.45$	8583	3776	8501	5139	4276	2735

Table 4.3.: Total number of iterations of the CG solver, $||R|| < 1.0E-04$. Nonlinear system solved with 40 load increments and ≤ 3 Newton steps per increment.

4.6. Comparison of Surface Curving Techniques

In this section, we compare the different approaches for the polynomial approximation of curved element faces. First, we investigate the approximation of curves and choose the circular arc and the cross section of a NACA0012 airfoil to compare the continuity approach with point-normals and the interpolation approach with uniform and Chebychev-Lobatto point distributions.

4.6.1. Interpolation of a Circular Arc

We investigate the polynomial approximation of a circular arc, which is defined as a parametric curve

$$\boldsymbol{x}(\xi) = \begin{pmatrix} x(\xi) \\ y(\xi) \end{pmatrix} = \begin{pmatrix} R\cos\left(\frac{\varphi}{2}(\xi+1)\right) \\ R\sin\left(\frac{\varphi}{2}(\xi+1)\right) \end{pmatrix}, \qquad \xi \in [-1;1], \tag{4.40}$$

with angular extent φ and radius R. Note that the arc length increases linearly with the parameter ξ, which is a desired property for a circular parametrization. We introduce a polynomial approximation of the circular arc with a polynomial degree of N_g

$$\boldsymbol{x}(\xi) \approx \boldsymbol{x}_h(\xi) = \sum_{i=0}^{N_g} \hat{\boldsymbol{x}}_i \ell_i(\xi). \tag{4.41}$$

The polynomial coefficients $\hat{\boldsymbol{x}}$ are either defined by a set of (N_g+1) interpolation nodes, where we will compare uniform and Chebychev-Lobatto distributions, or they are derived from a cubic spline ($N_g = 3$) reconstruction, using only the end points and normal vectors of the circular arc. We exactly evaluate the circular arc function (4.40) to compute the interpolation points and normal vectors.

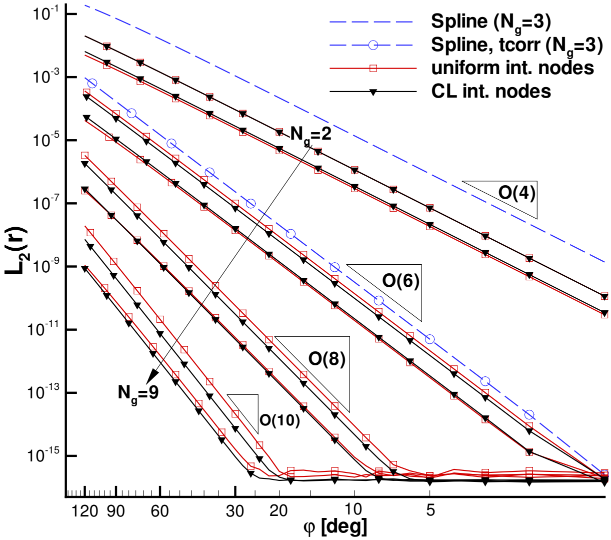


Figure 4.24.: L_2 error norm of the radius for different approximations of a circular arc with angular extent $\varphi \in [1°; 120°]$.

As a measure of the approximation errors, we consider the L_2 error norm, and we approximate the norm by evaluation of the error function e at the point set $\{\xi_i^e\}_{i=1}^{M_e}$,

$$L_2(e) = \sqrt{\int_{\xi=-1}^{1} |e|^2 \, d\xi} \approx \sqrt{\frac{1}{M_e} \sum_{i=1}^{M_e} |e(\xi_i^e)|^2}\,, \tag{4.42}$$

where we choose $M_e = 400$ uniformly spaced points. We compute the error function of the radius as

$$e(r) = \frac{r_h(\xi) - R}{R}\,, \quad r_h(\xi) = \sqrt{x_h(\xi)^2 + y_h(\xi)^2}\,, \tag{4.43}$$

of the normal vector as

$$e(n) = 1 - \boldsymbol{n}_h(\xi) \cdot \boldsymbol{n}_{\mathrm{loc}}(\xi)\,, \tag{4.44}$$

relating the normal from the polynomial derivative to the local normal defined by the node position

$$\boldsymbol{n}_h(\xi) = \frac{1}{\sqrt{\dot{x}_h(\xi)^2 + \dot{y}_h(\xi)^2}} \begin{pmatrix} \dot{y}_h(\xi) \\ -\dot{x}_h(\xi) \end{pmatrix}, \quad \boldsymbol{n}_{\mathrm{loc}}(\xi) = \frac{1}{r_h(\xi)} \begin{pmatrix} x_h(\xi) \\ y_h(\xi) \end{pmatrix}, \tag{4.45}$$

and finally the error function of the curvature

$$e(c) = \frac{c_{\mathrm{ex}} - c_h(\xi)}{c_{\mathrm{ex}}}\,, \quad c_{\mathrm{ex}} = \frac{1}{R}\,, \quad c_h(\xi) = \frac{\dot{x}_h(\xi)\ddot{y}_h(\xi) - \dot{y}_h(\xi)\ddot{x}_h(\xi)}{(\dot{x}_h(\xi)^2 + \dot{y}_h(\xi)^2)^{\frac{3}{2}}}\,. \tag{4.46}$$

The L_2 error norm of the radius for the interpolation with uniform and the Chebychev-Lobatto node distributions as well as the spline reconstruction are shown in Fig. 4.24 for polynomial degrees ranging between $N_g = 2, \ldots, 9$. The error drops with decreasing arc length, from left to right, from $\varphi = 120^\circ$ to $\varphi = 1^\circ$. The Chebychev-Lobatto node distribution has only slightly better interpolation properties compared to a uniform node distribution. The symmetry of the interpolation problem yields an odd-even behavior in the convergence, odd polynomial degrees have on order of convergence $L_2(r) \sim \mathcal{O}(N_g{+}1)$, whereas for even polynomial degrees, the order of convergence is higher $L_2(r) \sim \mathcal{O}(N_g + 2)$. Even though the spline reconstruction results in a cubic polynomial, the error is always larger than the quadratic interpolation approach. The use of the optimal tangent correction from (4.8) by Goldapp [54] clearly increases the accuracy and the convergence rate, since it is optimized for the circular arc. The minimum error is bounded by the double precision rounding error 10^{-16}. Note that this is simply due to a better parametrization of the interpolation polynomial. In Bjøntegaard et al. [14], an optimization approach for the parametrization of the interpolation of an arbitrary curve is presented, revealing huge gains in accuracy and fast spectral convergence.

In Fig. 4.25 we plot the errors of the normal vectors and curvature. The error of the normal vector does only account for the normal vector direction and therefore convergence rates are higher. The spline reconstruction without tangent correction is again less accurate than the quadratic interpolation polynomial. For high polynomial degrees, the error is mainly bounded by the rounding error.

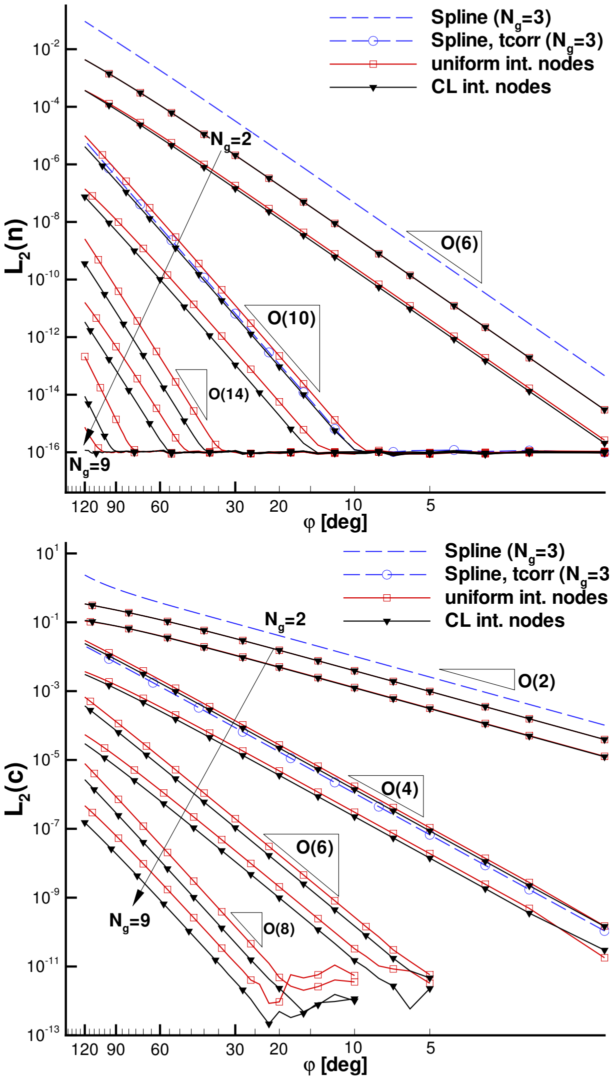


Figure 4.25.: L_2 error norm of the normal vector and curvature for different approximations of a circular arc with angular extent $\varphi \in [1°; 120°]$.

Many grid generators do not provide double precision data. We increase the rounding error of the initial data to single precision by a randomized disturbance of 10^{-8} and plot the mean error of 1000 realizations in Fig. 4.26. For the polynomial degrees of $N_g = 8, 9$, the error is fully dominated by the rounding error, and is slightly higher for uniformly spaced interpolation points. They can be even less accurate than low order approximations in the case of a small angular extent.

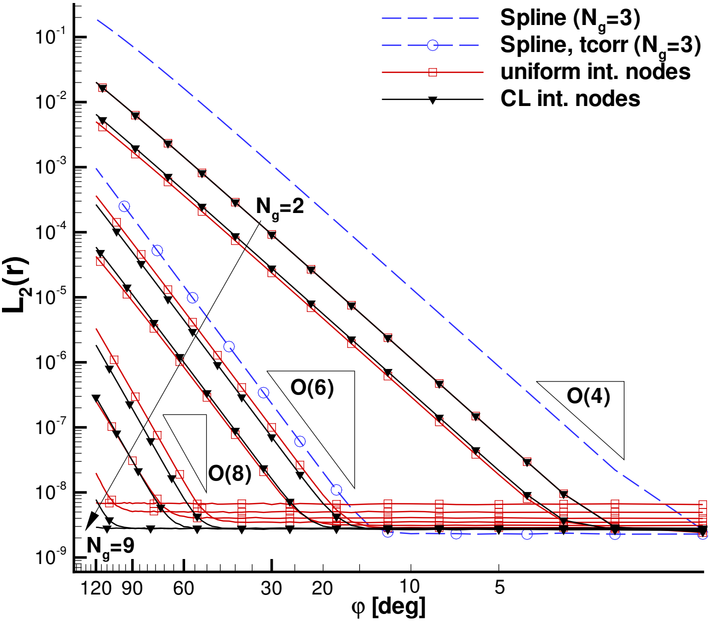


Figure 4.26.: L_2 error norm of the radius with a random noise disturbance of 10^{-8}, for different approximations of a circular arc with angular extent $\varphi \in [1°; 120°]$.

From these plots we can estimate the accuracy of the geometry approximation for a given mesh resolution and a given polynomial degree. It is clear that the point-normal approach is only recommendable for fine meshes, and that the interpolation approach is more flexible, since the polynomial degree can be chosen according to the resolution. Even though Chebychev-Lobatto points produce smaller errors, the difference to uniformly spaced points is marginal in the investigated range of $N_g = 2, \ldots, 9$.

4.6.2. Interpolation of a NACA0012 Airfoil

We consider now the approximation of a NACA0012 airfoil. The cross-section is given by an analytical function defined in [75]

$$\begin{gathered} y_{\mathrm{NACA}}(x) = \frac{t}{0.2}\left(c_0\sqrt{x} + c_1\,x + c_2\,x^2 + c_3\,x^3 + c_4\,x^4\right)\,, \quad x \in [0;1]\,, \\ (c_0;c_1;c_2;c_3;c_4) = (0.2969;-0.1260;-0.3516;0.2843;-0.1015)\,, \end{gathered} \tag{4.47}$$

where the thickness of the NACA0012 airfoil at $x = 0.2$ is given by $t = 0.12$ for a chord length of $c = 1$.

We will only investigate the nose of the airfoil ($x < 0.1$) featuring the largest curvature, see Fig. 4.27.

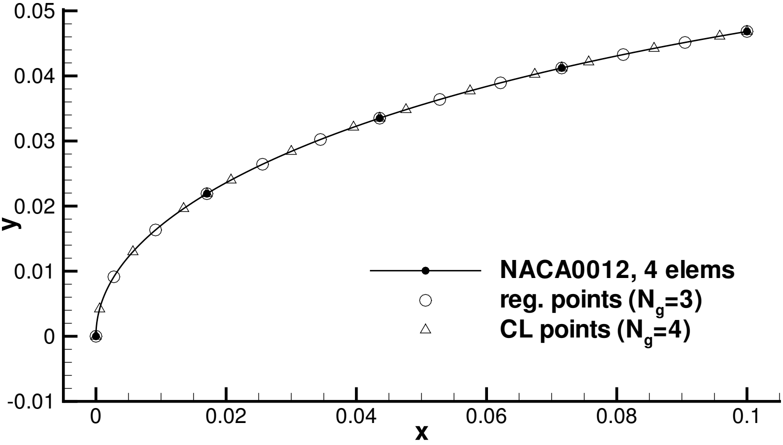


Figure 4.27.: Nose of a NACA0012 airfoil with 4 elements and a uniform and Chebychev-Lobatto point distribution and an arc length parametrization.

If we want to choose a point distribution along the curve, an arc length parametrization of the curve is desirable. Still, superior parameterizations to the arc length parametrization can be produced by optimization [14]. The arc length parametrization is a trivial task for any grid generator. We implemented the following algorithm for the arc length parametrization, enabling us to generate an arbitrary number of uniformly spaced points along any non-parametrized curve. We choose a first parametrization of the airfoil curve as

$$x(t) = t^2, \quad y(t) = y_{\mathrm{NACA}}(t^2) \tag{4.48}$$

and can compute the arc length by

$$s(t) = \int_0^t \sqrt{\dot{x}(\tau)^2 + \dot{y}(\tau)^2}\, d\tau \,. \tag{4.49}$$

The arc length parametrized curve then reads as

$$x(s) = x(t(s))\,, y(s) = y(t(s))\,. \tag{4.50}$$

Here, we need the inverse mapping $t(s) = (s(t))^{-1}$. In [96], a spline approximation of the inverse mapping was chosen, which is justified by the fact that the mapping $s(t)$ is a smooth and monotonically increasing function. One can use a spline approximation of the curve itself, and then integrate (4.49) with Gauss-quadrature. This is done in [96] for fast but approximate solutions. For accurate results, we propose a simple and robust strategy. We sample the curve with a high number points (here we used 10^5 points) and approximately integrate the arc length by a sum of the distance between consecutive points

$$s(t_i) = s(t_{i-1}) + |(\boldsymbol{x}(t_i) - \boldsymbol{x}(t_{i-1}))|\,, i = 0, \dots, n\,, \tag{4.51}$$

leading to a discrete mapping $t_i \mapsto s(t_i)$. We use every 10th point to define the spline of the inverse mapping. It is possible to iteratively increase the accuracy of the arc length integration, by recomputing t_i in each step from the current inverse mapping and a uniform arc length distribution

$$t_i^{(k)} \to s(t_i^{(k)}) \to [t(s)]^{(k)} \to t_i^{(k+1)} = \left[t \left(s = \frac{i}{n} s_{\max} \right) \right]^{(k)} . \tag{4.52}$$

Now we insert the interpolation points at a specific arc length parameter s_{IP} and evaluate the parametric curve (4.48) at $\boldsymbol{x}(t_{IP}) = \boldsymbol{x}(t(s_{IP}))$. The exact normal vector and curvature are computed from the derivatives, following equations (4.45) and (4.46).

We will define the error of the approximation as the distance between the polynomial and the exact geometry. We have to sample the polynomial first at uniformly spaced points in reference space, following (4.42). However, the question is how to compute the distance to the exact geometry. If we compare to a set of uniformly spaced points on the curve, we must be aware of assessing the error of the curve parametrization *and* the error in geometry representation together. If we define the error as the exact distance to the geometry, we have to find the nearest points on the curve. In the case of the circular arc, this was simply the point in radial direction. For the airfoil, this involves a root finding process, where we have to minimize the distance of a given point position (x_p, y_p) to the curve $x(t), y(t)$

$$(x(t) - x_p)^2 + (y(t) - y_p)^2 \to \min \,, \tag{4.53}$$

$$\Rightarrow \quad F(t) = (x(t) - x_p)x_t(t) + (y(t) - y_p)y_t \overset{!}{=} 0 \,, \tag{4.54}$$

and find the parameter t by iteration with Newtons' method

$$t^{(k+1)} = t^{(k)} - \frac{F(t^{(k)})}{F_t(t^{(k)})} \,. \tag{4.55}$$

The iteration is finished for a residual of $|t^{k+1} - t^k| < 10^{-15}$. We will refer to the error of the *minimum distance*, if the root finding process is applied.

We restrict the investigation of error to a uniform mesh where all elements have the same arc length, and we choose again the cubic spline from point-normals as well as the interpolation approximation with uniform or Chebychev-Lobatto point distributions along each element. The resolution ranges from 4 to 256 elements. We plot the L_2 error norm of the distance to a uniformly spaced point set in Fig. 4.28 over the number of elements. Since the curve does not have any symmetries, the convergence behavior for the interpolation approach is simply $\mathcal{O}(L_2(\Delta x)) \sim (N_g+1)$. Not much difference between the uniform and Chebychev-Lobatto point distributions is observed. They are equal for $N_g = 5$, the uniformly spaced points being slightly better for $N_g < 5$, and Chebychev-Lobatto points for $N_g > 5$. The cubic spline from the point-normals is again less accurate than the quadratic interpolation. Even though the curve is not a circle, the tangential correction helps to decrease the approximation error by an order and falls below the quadratic interpolation error. The third order error convergence for the spline is not optimal, since it does not correspond to the polynomial degree of $N_g = 3$.

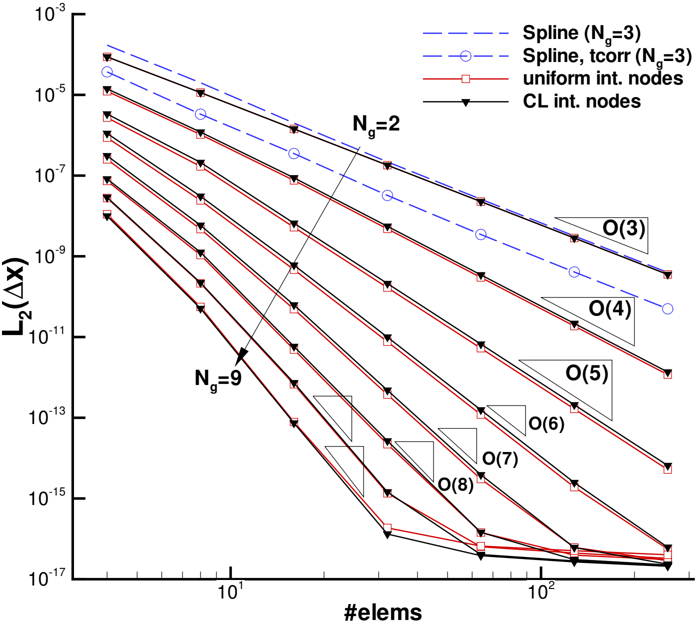


Figure 4.28.: L_2 error norm of the distance to the NACA0012 profile, over the number of elements, for different polynomial approximations.

If we only want to know the geometry error without the parametrization error, we have to consider the error norm computed with the minimum distance, plotted in Fig. 4.29. Note the difference for the spline approach, now converging with fourth order. It permits the reverse conclusion that the spline approximation has a large parameterization error, and now error falls below the quadratic interpolation for a resolution of more than 16 elements. As expected, the error decreases for the interpolation approach and an increasing polynomial degree, but does not show the convergence behavior at low resolutions and high polynomial degrees, in contrast to the error to a uniformly spaced point set.

In Fig. 4.30 the difference between the two errors is shown for the Chebychev-Lobatto interpolation points. The minimum distance error is always smaller, and the difference corresponds to the parametrization error. For example,

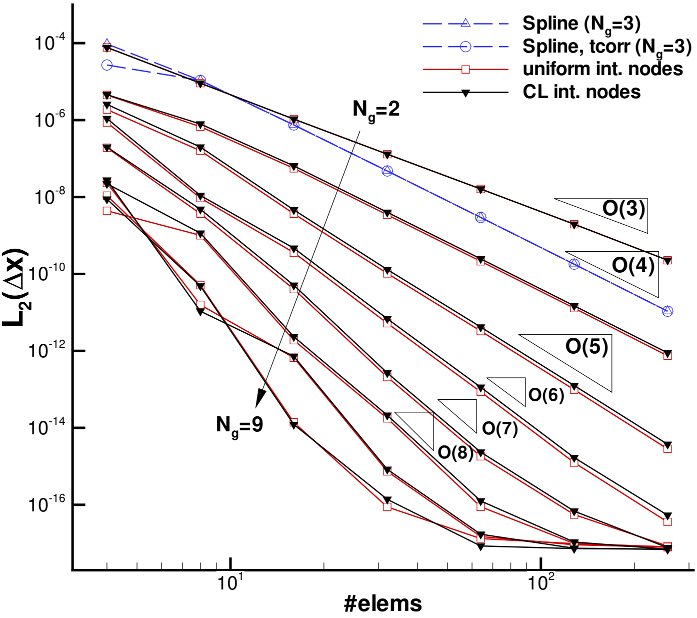


Figure 4.29.: L_2 error norm of the *minimum distance* to the NACA0012 profile, over the number of elements, for different polynomial approximations.

an approximation of $N_g = 8$ and 16 elements, the errors are equal, thus the parametrization error is very small, but at a resolution of 8 elements, the parametrization error is large. For $N_g = 9$, the situation is reversed, and an odd-even behavior is observed again, where $N_g = 7$ behaves like $N_g = 9$ and $N_g = 6$ like $N_g = 8$.
The normal vector and curvature error are less sensitive to the location of the evaluation, we plot in Fig. 4.31 the errors from the points of minimum distance. The error norms behave similarly to the distance error norm, except for a larger difference between the uniform and Chebychev-Lobatto point distribution for interpolation.

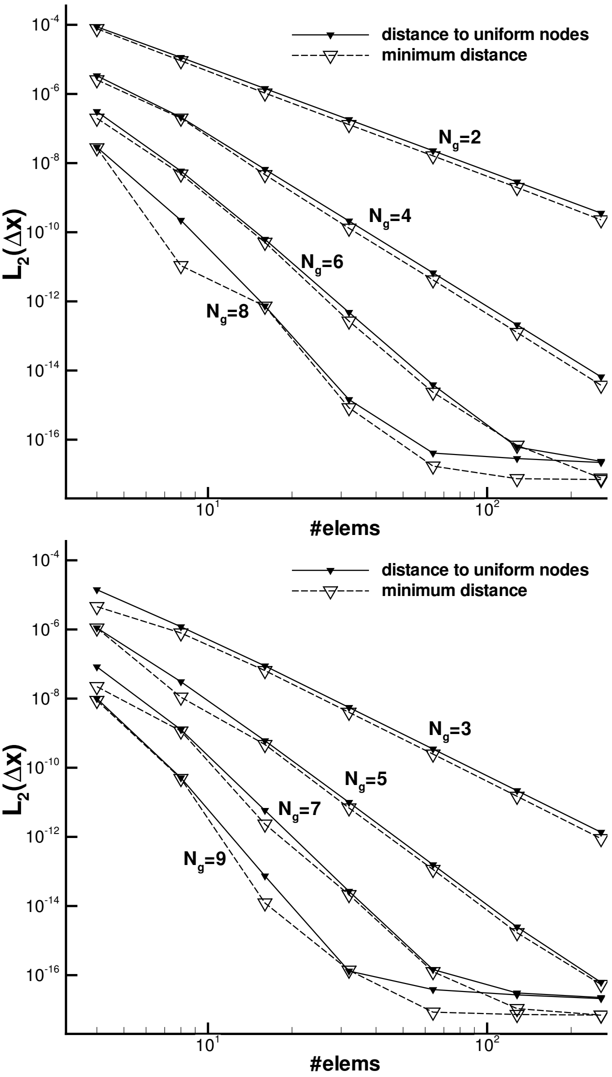


Figure 4.30.: Comparison of the error of the distance to a uniformly spaced point set and the minimum distance, for Chebychev-Lobatto interpolation and even (left) and odd (right) polynomial degrees N_g.

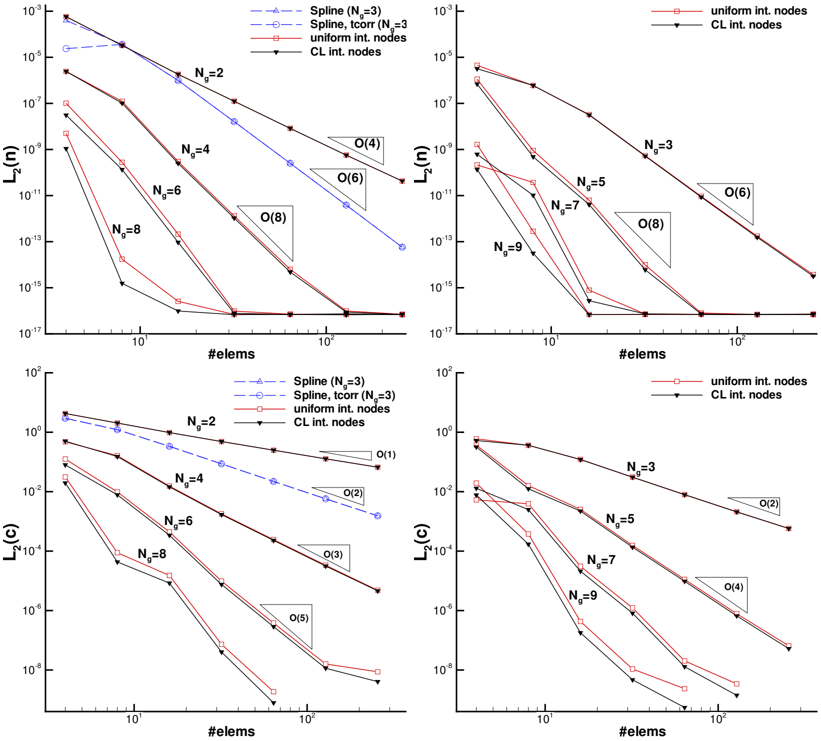


Figure 4.31.: Error norm of the normal vector (top) and curvature (bottom) for even (left) and odd (right) polynomial degrees.

4.6.3. Spherical Quadrilateral

The results of the curve approximation can only be translated to 3D surfaces if the curvature is restricted to one parameter, for example a cylinder or a wing, and if the edges of the elements are aligned with the curvature direction. If the surface mesh is unstructured or the surface has two curvatures, both parameter directions are curved. Here, we have to reassess the approximation properties of the different approaches. The edges of the surface elements are found by interpolation or spline reconstruction. Only the subdivision approach and the agglomeration of structured meshes provide additional interpolation nodes inside the surface area, with a uniform point distribution. The other methods employ a transfinite interpolation of the edges (Coons surface). As a test case to assess the errors, we investigate the approximation of a spherical quadrilateral, shown in Fig. 4.32. Its edges are circles of the sphere radius (geodesics of the sphere surface) and the parametrized surface has no rotational symmetry.

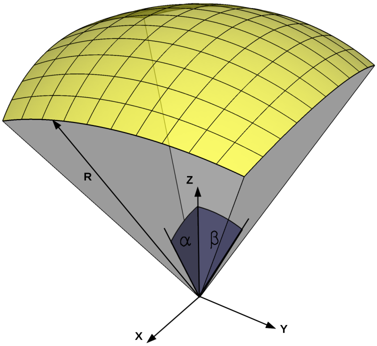


Figure 4.32.: Sketch of the spherical quadrilateral surface with parametric lines.

Each point of the spherical quadrilateral is defined by the intersection of two planes and a sphere. The sphere is defined by $|\vec{x}| = R$ and the first plane is the x-z-plane rotated around the x-axis by an angle α, the second plane is the y-z-plane rotated around the y-axis by an angle β. Then, the spherical quadrilateral reads as

$$\vec{x}(\xi,\eta,R) = \frac{R}{\sqrt{(\cos\alpha)^2 + (\sin\alpha\cos\beta)^2}} \begin{pmatrix} \cos\alpha\sin\beta \\ \sin\alpha\cos\beta \\ \cos\alpha\cos\beta \end{pmatrix}, \tag{4.56}$$

$$\alpha = \xi\alpha^*, \quad \beta = \eta\beta^*, \quad \xi,\eta \in [-1;1]$$

with boundaries defined by (α^*, β^*). We will only investigate the fully symmetric case $\alpha^* = \beta^*$ for a range of $[1, 60]$ degrees.

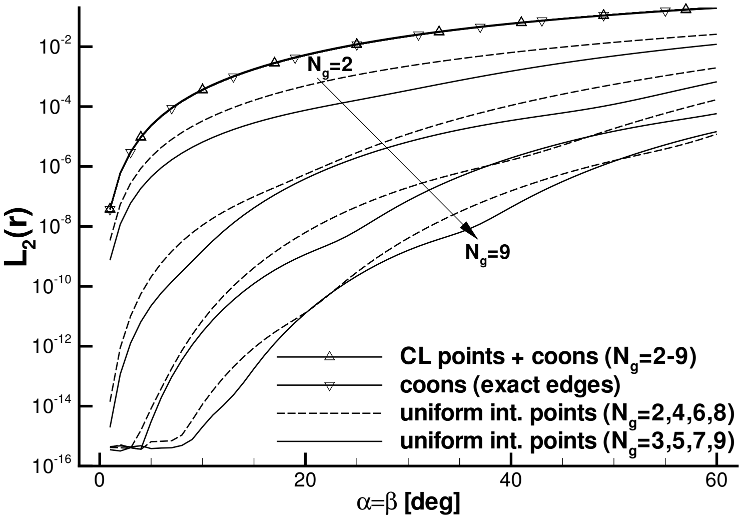


Figure 4.33.: L_2 error norm of the radius for different approximations of the spherical quadrilateral.

In Fig. 4.33 we plot the error norm of the radius, where we have a full set of uniformly spaced interpolation nodes on the surface and compare to a coons surface patch with exact boundary curves and curves defined by Chebychev-Lobatto interpolation points. The error inside the surface when using the coons patch is dominant, since the Coons blending of the interpolated and the exact curves have the same error. All interpolations, even though the point distribution is uniform, have much lower errors than the coons surface patch. An odd even behavior of the error norms can be observed.

An improvement of the classical coons patches is discussed by Stiller et al. [92], employing rational quadratic coons patches to guarantee that surface normals coincide with the boundary curve normals, leading to much better approximations of conics and spheres. We did not further investigated rational coons patches, but one could construct better interpolation points inside the surface patch and then approximate the surface again by polynomial interpolation, combined with the parametrization optimization from [14].

4.6.4. Influence of Mesh Generation Errors

The investigation above relied on analytical descriptions of the geometry and did not include any mesh generation. However, there are several sources of errors during mesh generation process. For an airfoil, the starting point is typically a spline curve, from a discrete set of data points. Thus the number and position of the interpolation points as well as their distribution affects the geometry representation.

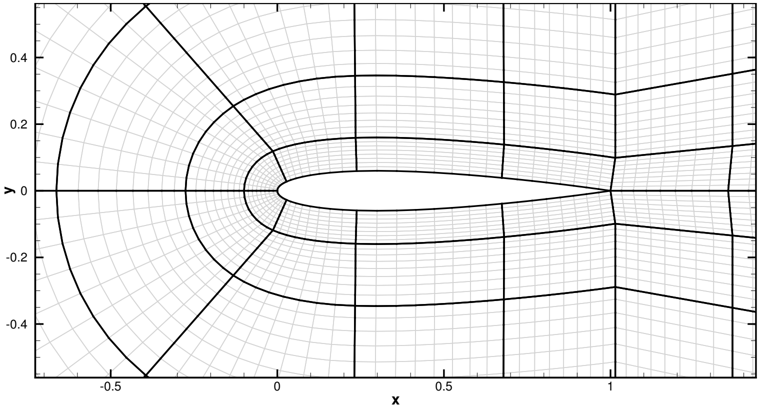


Figure 4.34.: NACA0012 mesh with 4 curved elements ($N_g = 8$) along the chord, generated from a structured base mesh (in gray).

Second, the accuracy of the mesh generation itself, where points are projected onto the geometry, has a certain tolerance. Finally, input/output processes are prone to errors, for example if only single precision data is provided.

In this section, we use a mesh generated with ICEM©and show that the mesh generation error is dominant, especially for coarse high order meshes. We implemented a function in *HOPR* to project the interpolation nodes of the boundary onto the analytical description of the geometry, using the minimum distance iteration procedure from (4.55). This enables us to separate the errors of the mesh generation process from the errors of the polynomial approximation.

The airfoil geometry definition has 200 interpolation points along the chord with a length of $c = 1000$. The spline as well as the CAD definitions for the domain were generated in ANSA©. The CAD data was transferred via the STEP-CAD standard interface to ICEM©to generate the structured mesh. It is a common practice to use dimensions $\gg 1$ in mesh generators to minimize errors from internal tolerances. We scale the mesh in *HOPR* to a chord length of $c = 1$.

The structured base mesh has 32 elements along the chord length of the airfoil and we used the direct volume curving approach, from Section 4.4, to generate different curved meshes. The coarsest mesh consists of 4 elements along the chord and is shown together with the base mesh in Fig. 4.34. We produce polynomial degrees of $N_g = 2, 4, 8$ for the same number of elements by simply skipping intermediate points. We also generate an 8 element mesh with $N_g = 2, 4$ and a 16 element mesh with $N_g = 2$.

For all meshes, the mesh points at the airfoil boundary have a maximum distance of roughly $1E - 04$ relative to the chord length $c = 1$. In Fig. 4.35 and Fig. 4.36 we plot the difference between a curved mesh with points from the mesh generator compared to a mesh with projected interpolation points. The polynomial approximation errors are made visible by magnifying the *distance to the exact geometry* by a factor m

$$\boldsymbol{x}_{\text{vis}} = m(\boldsymbol{x} - \boldsymbol{x}_{\text{ex}}) + \boldsymbol{x}_{\text{ex}}\,, \tag{4.57}$$

with $m = 50$ for $N_g = 4$ and $m = 100$ for $N_g = 8$. The distance of the interpolation points is magnified by the same factor.

The largest errors occur in the first two cells, and point positions are not symmetric. The errors are larger on the lower side of the airfoil. In Fig. 4.35 the error of the interpolation point positions do not induce any oscillations of the polynomial with degree $N_g = 4$, whereas for the higher degree $N_g = 8$ in Fig. 4.36, the additional points lead to more oscillations, of which the magnitude however remains inside the distance of the interpolation points. Note that the plots have different magnification factors and that initial point positions are the same, only that for $N_g = 4$ every second point is used. Finally, the oscillations are highly reduced when projected interpolation points are employed.

We can look at the difference in accuracy un more detail. In Fig. 4.37, we plot the maximum error norm L_∞ of the interpolation points used for the mapping along with the L_∞ and L_2 error norms of the resulting interpolation polynomial. As comparison for the latter, we show the error of the polynomial approximation, using the projection of the interpolation points on the exact geometry (marked with *). In Table 4.4, the error norms are listed together with the normal vector error norms.

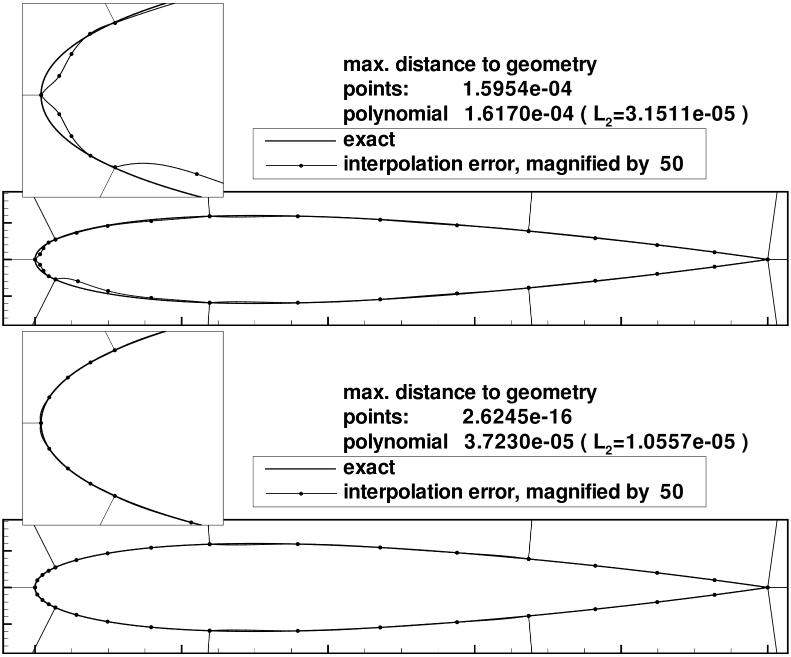


Figure 4.35.: Polynomial approximation for $N_g = 4$, the distance to exact geometry is magnified by factor 50. Initial mesh (top) and using projected interpolation points (bottom).

For the coarsest mesh with 4 elements, the approximation with $N_g = 2$ has the largest errors, which are mainly dominated by the polynomial approximation, since using the projected nodes does not change the error. But if we increase the polynomial degree to $N_g = 4$, the mesh generation error becomes dominant, and further increasing to $N_g = 8$ does not change the distance error anymore. The error in the normal vector is even larger than for $N_g = 4$, see Table 4.4. Comparing the errors to the interpolation with projected points reveals that the polynomial approximation error for $N_g = 4$ and $N_g = 8$ is very low.
After the refinement of the mesh from $n_e = 4$ to $n_e = 8, 16$ for $N_g = 2$ and from $n_e = 4$ to $n_e = 8$ for $N_g = 4$, the error is again bounded by the mesh generation

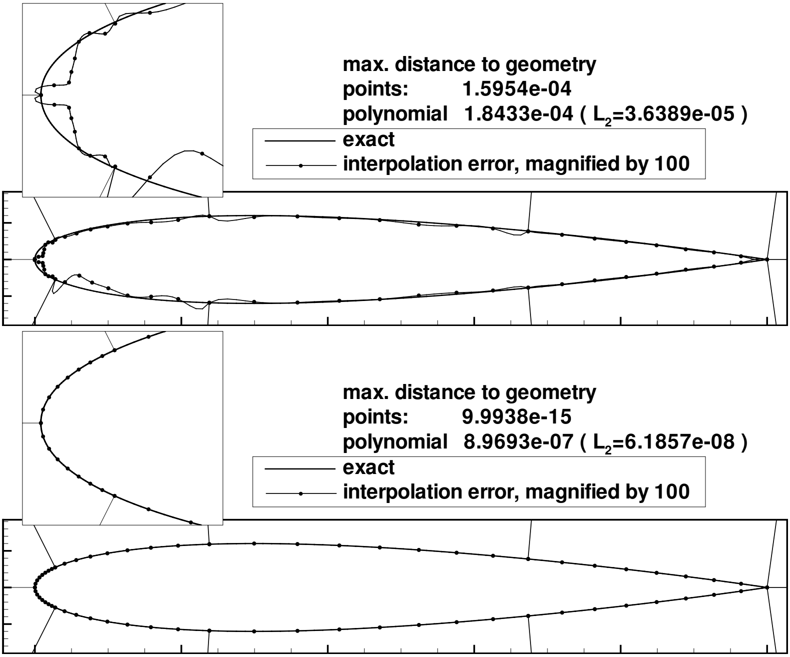


Figure 4.36.: Polynomial approximation for $N_g = 8$, the distance to exact geometry is magnified by factor 50. Initial mesh (top) and using projected interpolation points (bottom).

error. The question remains in which cases an accuracy of $1E-04$ is really needed. To answer, we simulate an inviscid compressible 2D flow at $Ma = 0.3$ with DG-SEM on the coarsest $n_e = 4, N_g = 8$ mesh, with and without the projection of interpolation points. We increase the resolution by increasing the polynomial degree from $N = 4$ to $N = 8$ and $N = 16$. The flow should remain isentropic, and the deviation in entropy is a good measure for geometry errors. We plot the entropy error $|e - e_0|$ as well as the pressure coefficient at the wall $c_p = (p - p_0)/(\frac{1}{2}\rho v_0^2)$ in Fig. 4.38.

When $N = 4$, the high order element mapping is interpolated to $N_g = 4$ for the metrics evaluation, and there is no significant change between the initial mesh

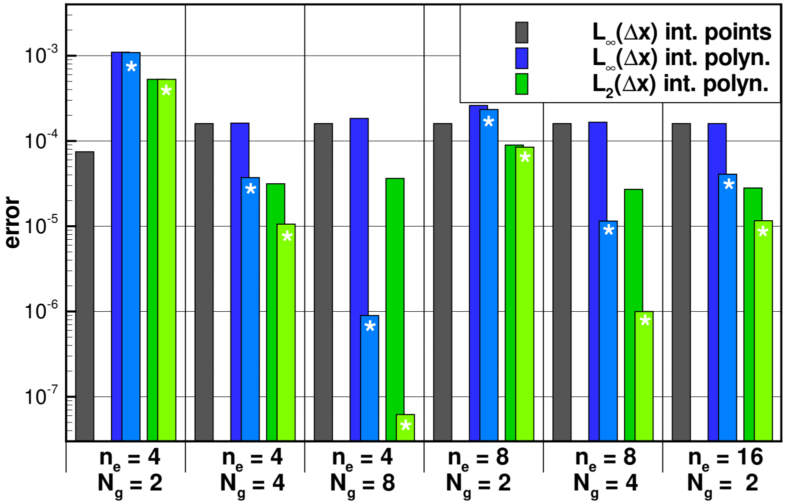


Figure 4.37.: Comparison of minimum distance error norms for NACA0012 mesh, using interpolation points from a mesh generator, the cases with interpolation point projection are marked with *.

and the mesh with projected interpolation points. Increasing the resolution however reveals again a large influence of the mesh generation error. For the initial mesh, the entropy error is reduced by a factor of two from $N = 4$ to $N = 8$ but not anymore for $N = 16$. The pressure distribution shows large jumps at the element interfaces, especially between the first and second element. No improvement is found when changing from $N = 8$ to $N = 16$. Only if the projected interpolation points are used does the increased resolution actually improve the solution and the entropy error is improved by more than an order. We arrive at the conclusion that the discretization error of the numerical scheme can hide the errors in geometry, but with a higher resolution, the error is dominated by the geometry. It reveals that we cannot only improve the polynomial approximation of the geometry only, but we have to to look at every step of the mesh generation process to reduce the overall error. However, this is not always easy to influence as a user of the software, especially when it comes to errors from input/output formats of CAD or mesh data.

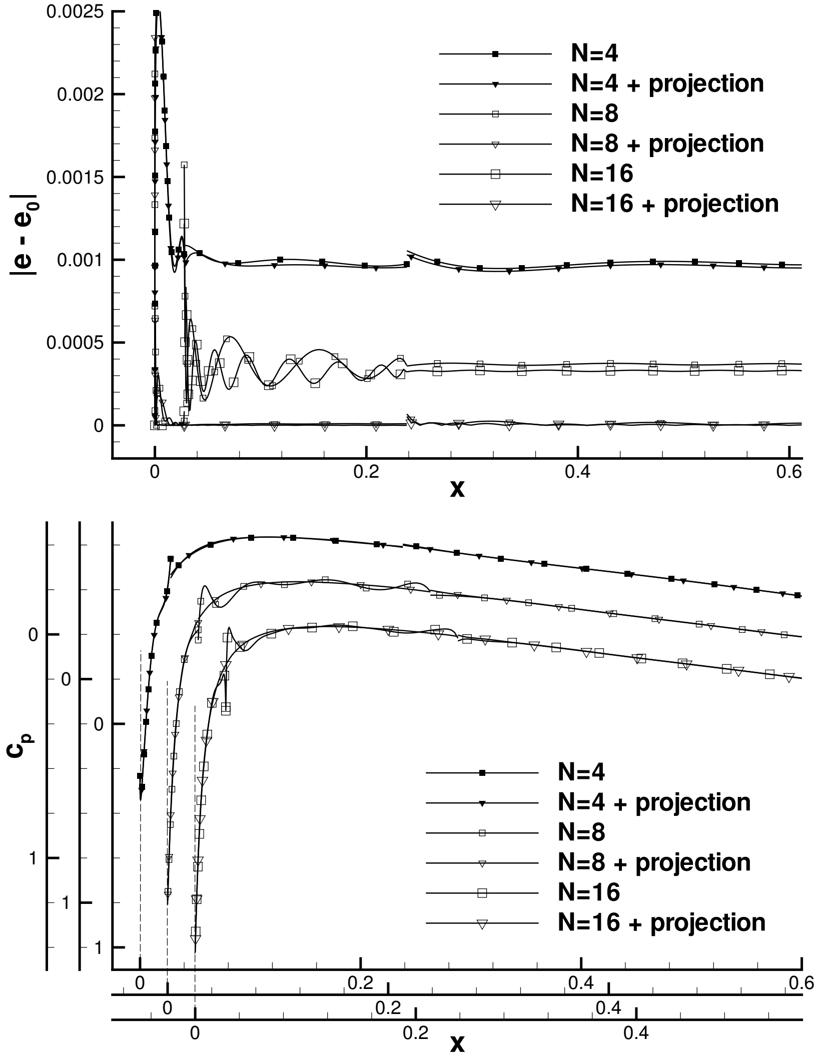


Figure 4.38.: Entropy error and pressure coefficient along the front of the airfoil, $n_e = 4, N_g = 8$, for an inviscid flow of $Ma = 0.3$.

		At int. points	polynomial			
n_e	N_g	$L_\infty(\Delta x)$	$L_\infty(\Delta x)$	$L_2(\Delta x)$	$L_\infty(n)$	$L_2(n)$
From mesh data						
4	2	7.47E-05	1.10E-03	5.29E-04	4.53E-02	2.90E-03
	4	1.60E-04	1.62E-04	3.15E-05	1.40E-04	8.82E-06
	8	1.60E-04	1.84E-04	3.64E-05	5.46E-03	3.49E-04
8	2	1.60E-04	2.61E-04	8.96E-05	6.95E-03	3.18E-04
	4	1.60E-04	1.66E-04	2.71E-05	2.91E-04	1.68E-05
16	2	1.60E-04	1.60E-04	2.81E-05	4.26E-04	1.95E-05
With projection of interpolation points onto geometry						
4	2	1.5E-16	1.09E-03	5.28E-04	4.44E-02	2.83E-03
	4	2.6E-16	3.72E-05	1.06E-05	2.45E-05	7.05E-07
	8	1.0E-16	8.97E-07	6.19E-08	9.66E-07	2.17E-08
8	2	1.3E-16	2.34E-04	8.47E-05	5.46E-03	2.49E-04
	4	2.6E-16	1.15E-05	9.99E-07	1.12E-04	3.20E-06
16	2	1.5E-16	4.08E-05	1.16E-05	2.37E-04	1.03E-05

Table 4.4.: Minimum distance error norms for NACA0012 mesh, using interpolation points from a mesh generator. The distance error $L_\infty(\Delta x)$ of the interpolation points is given as well as the error of the interpolation polynomial.

4.7. Applications

4.7.1. Coaxial Waveguide

We choose a simple test case to show the importance of high order grids for high order numerical schemes. The geometry is part of a coaxial cable, whose domain length equals two wavelengths, see Fig. 4.39. We solve the linear Maxwell equations with DG-SEM. An exact solution exists for this problem [82].

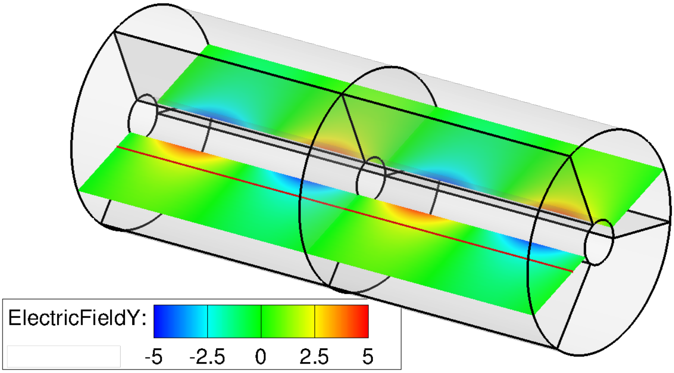


Figure 4.39.: Geometry and initial solution of the coaxial waveguide (slice at $x = 0$, extraction line at $y = 0.2$).

The exact solution, a three-dimensional wave, is initialized and then transported along the cable, which has periodic boundary conditions at both ends. A detailed description of the equations and the test case can be found in Munz et al. [82]. The cable dimensions are defined by the inner and outer radius of $r_1 = 0.1, r_2 = 0.5$ and a length of $L = 2.5$.

To show the influence of the geometry approximation, the domain is discretized by only six hexahedra. We approximate the DG solution to a fixed polynomial degree of $N = 5$, while we increase the polynomial degree of the geometry representation from $N_g = 1$ (linear) to $N_g = 5$, see Fig. 4.40. We use Chebychev-Lobatto interpolation points to represent the circular element edges and interpolation points are projected onto the exact cylindrical geometry. We use transfinite interpolation for the element surfaces and the volume as well, since we only have one element layer. It is clear that the straight sided mesh cannot

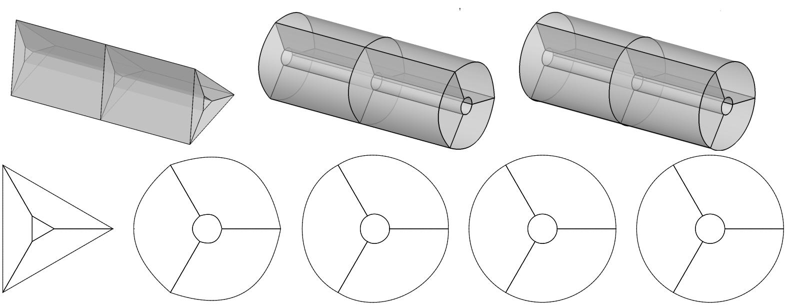


Figure 4.40.: 3D view of meshes for $N_g = 1, 2, 5$ and cross section for $N_g = 1, \dots, 5$.

represent the geometry at all. For quadratic and cubic elements, the surface normals at element edges still have visible kinks. For the cylinder mesh, the only source of the geometry error is the interpolation of the circular arc, which is investigated in Section 4.6.1. The edges of coaxial cable mesh are circular arcs of $\varphi = 120°$ and we list the error norms of the radius, normal vector and curvature in Table 4.5 for the four high order geometry approximations. The error norms are reduced by at least two magnitudes for $N_g = 5$ compared to the quadratic curve.

N_g	2	3	4	5
$L_2(r)$	2.01E-02	6.48E-03	2.66E-04	5.89E-05
$L_2(n)$	4.34E-03	3.68E-04	4.14E-06	7.46E-08
$L_2(c)$	3.42E-01	1.13E-01	2.41E-02	3.05E-03

Table 4.5.: Interpolation error norms for the radius, normal vector and curvature, for a circular arc of 120° for polynomial degrees $N_g = 2, \dots, 5$, taken from Section 4.6.1.

We compare one-dimensional profiles of the DG solution for $N = 5$ to the exact solution for different polynomial degrees of the mapping N_g in Fig. 4.41. We plot the solution after the wave has traveled two and ten times through the domain. The solution is extracted along the line $x = 0, y = 0.2$, depicted

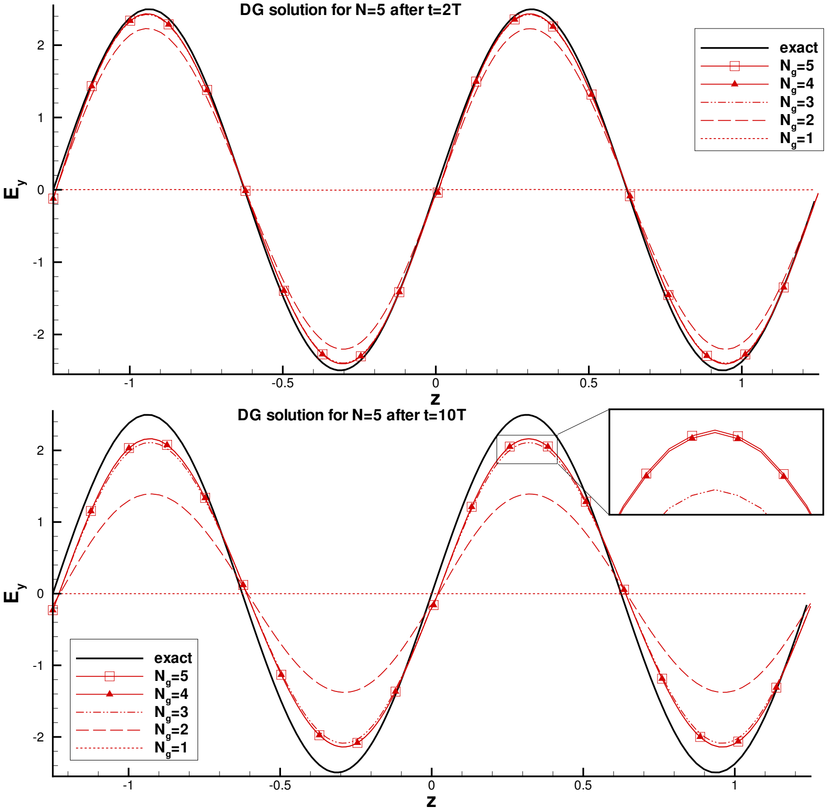


Figure 4.41.: Comparison of the electric field for $N = 5$ after 2 and 10 cycles for different geometry approximations.

Fig. 4.39. The domain is periodic and therefore no errors leave the domain. The dissipation of the $N = 5$ scheme reduces the amplitude of the wave as the solution is advanced in time. However, there is a strong influence of the geometry representation, up to a polynomial degree of $N_g = 4$, whereas the difference to $N_g = 5$ becomes marginal.

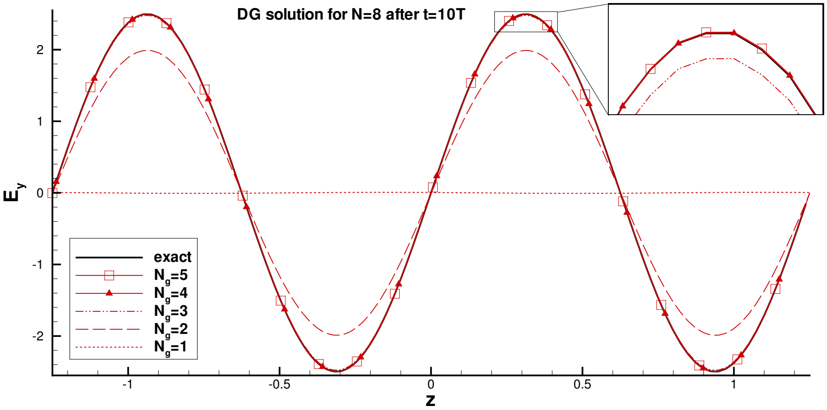


Figure 4.42.: Electric field for $N = 8$ after 10 cycles for different geometry approximations.

Hence, for $N_g \geq 4$, the error is dominated by the dissipation and dispersion properties of the DG scheme at $N = 5$. In Fig. 4.42, we increased the resolution of the scheme to $N = 8$, so that the dissipation of the scheme is reduced. The solution converges to the exact solution for $N_g = 4$, without any visible improvement with $N_g = 5$. These results show that for a chosen mesh, the high order geometry approximation convergences quickly by increasing the polynomial degree, and at a certain polynomial degree, the geometry errors are lower than the errors of the discretization. Thus the required polynomial degree of the geometry will be lower for finer meshes or, inversely, higher for coarser meshes.

4.7.2. Complex Three-dimensional Hybrid Curved Mesh of the DLR-F6

The DLR-F6 is an idealized wing-body-nacelle configuration of a passenger jetliner that was tested in multiple wind tunnels for comparison with numerical simulations. The CAD geometry as well as the experimental data can be found in Brodersen and Stürmer [23]. CAD patches were merged and split in ANSA©for the mesh generation. This test case shows the applicability of the surface curving to complex CAD geometries. Both the point-normal approach using CAD normals and the subdivision approach were applied. We will investigate the quality of the final curved hybrid mesh, without the mesh deformation.

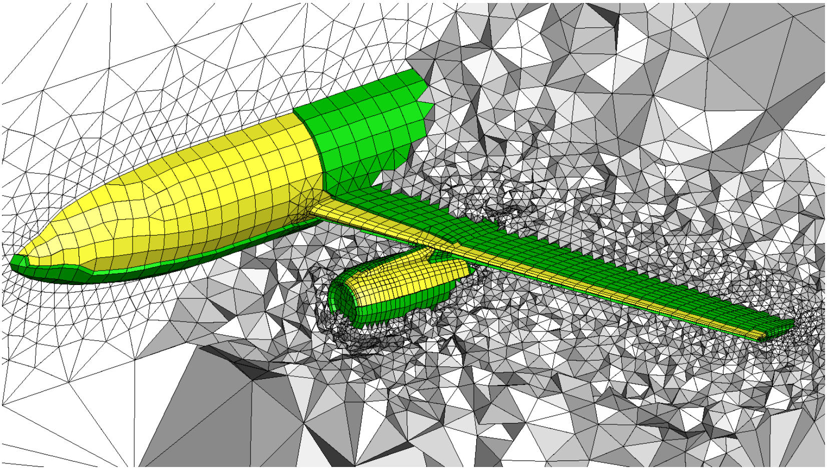


Figure 4.43.: Hybrid volume mesh of the DLR-F6 configuration (tetrahedra, pyramids,prisms and hexahedra).

The linear volume mesh consists of $\sim 231,000$ elements and is depicted in Fig. 4.43. The majority of the elements are linear tetrahedra ($\sim 225,000$). The near surface elements are meshed with one layer of curved prisms (~ 400) and curved hexahedra (~ 2600).

The curved elements are limited to the near surface region, as depicted in Fig. 4.45(a). Note that this is a mesh for an inviscid computation, without a boundary layer refinement.

The pictures in Fig. 4.45(b) show polynomial surface patches, generated with the point-normal approach and simply super-sampled for visualization. It can be clearly seen that the CAD edges are detected overall and a smooth surface curving is achieved.

As a measure of mesh quality, we already introduced the scaled Jacobian J_s, see (4.39) in Section 4.5. The best element quality is found for $J_s \to 1$. The statistics for all curved prisms and curved hexahedra are shown in Fig. 4.44. All elements have positive Jacobians, 90% of the hexahedra have a scaled Jacobian $J_s > 0.5$ and all prisms have $J_s > 0.4$.

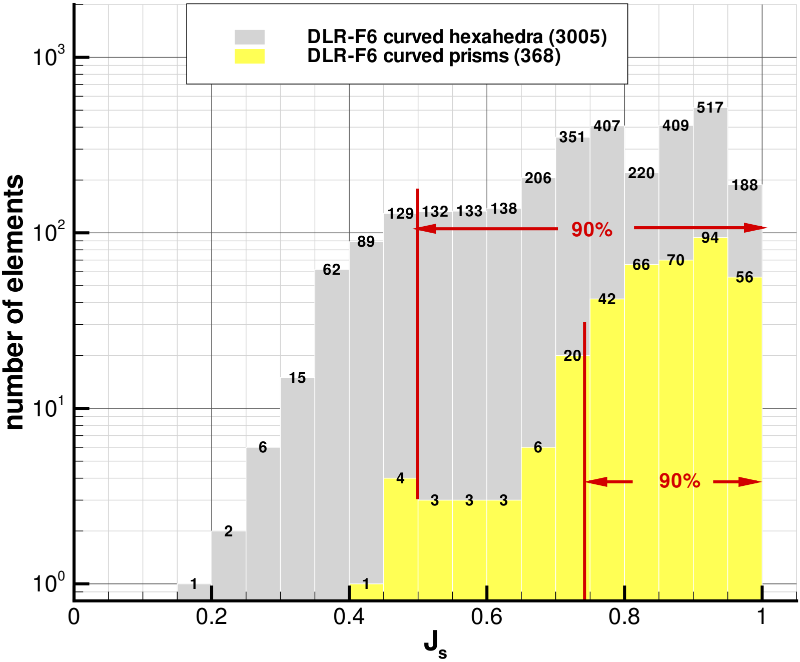
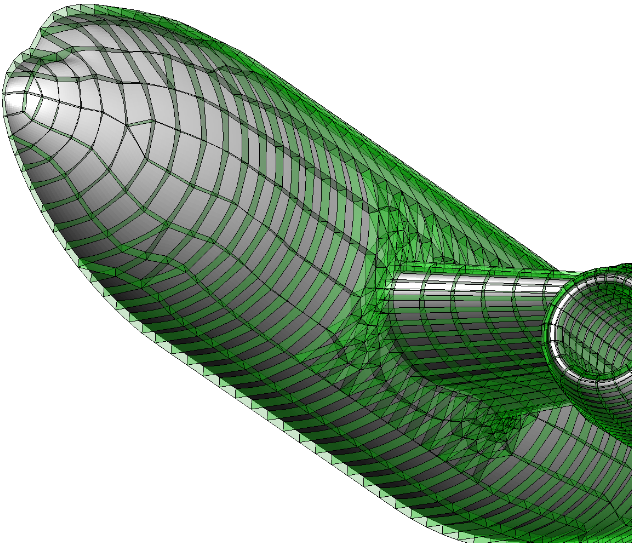
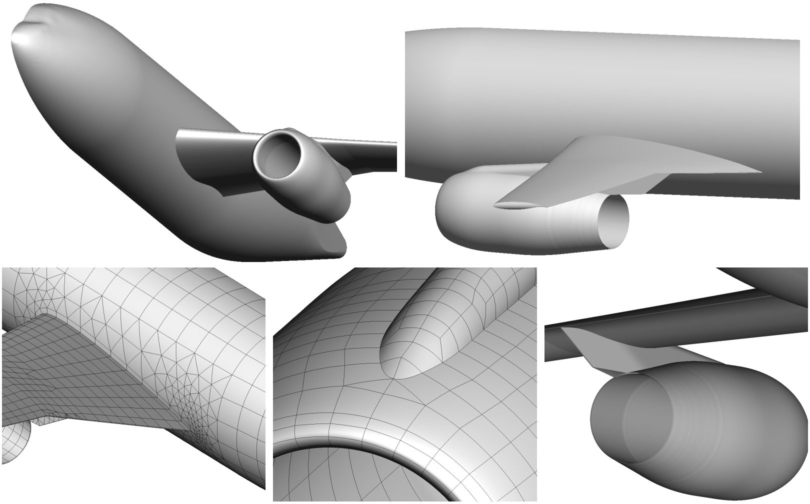


Figure 4.44.: DLR-F6 configuration: scaled Jacobian statistics for curved tetrahedra and prisms.



(a) Curved elements at the body surface



(b) Polynomial patches of the boundary surface elements

Figure 4.45.: Visualizations of the curved mesh.

4.8. Evaluation of the Curving Strategies

We proposed several strategies to generate a curved three-dimensional high order mesh. The question of which one to choose depends on the initial data provided. Our starting point is always a linear mesh, but there is a huge difference between a simple mesh file *as is*, with a list of discrete points and element connections, and the possibility to load the mesh with the mesh generator, where it was built, providing a connection to the geometry data. To our knowledge, there are no standardized mesh file formats for storing the mesh and the geometry together. We know that the accuracy of the surface representation is of utmost importance for a high order mesh, especially for wall boundary conditions in fluid dynamics or reflective boundaries in wave propagation problems. It is clear that for a linear mesh *as is*, the curved surface information has to be the generated or reconstructed *a-posteriori*, and the accuracy will simply depend on the linear mesh resolution. If the mesh generator is available, the additional information can be provided *a-priori*, and we are able to improve the accuracy by increasing the surface resolution.
The agglomeration approach with structured meshes is very attractive since we can directly generate a coarse three-dimensional curved mesh. This approach should be used wherever possible, since it naturally supports curved boundary layer meshes. However, three-dimensional structured mesh generation is a time-consuming task when meshing complex geometries and unstructured grids are commonly used instead. Any linear mesh generator can be used together with the *point-normal* approach, and exact normal vectors can be found if the CAD data is provided. We can use ANSA©and ICEM©to generate unstructured linear meshes and the additional surface data. The polynomial degree can be chosen arbitrarily, but the ANSA© *subdivision* approach allows only steps of $N_g = 2, 4, 8, \dots$ with a uniform point distribution, in comparison to the Chebychev-Lobatto point distribution of the ICEM© *spectral element* approach.
The analysis of approximation errors of the surface curving revealed that curved edges are best approximated by interpolation with Chebychev-Lobatto points and that the difference to an interpolation with uniformly spaced points is small. In the case of surfaces with two curvatures, the *spectral element* and the *point-normals* approach will only produce curved element edges, and a transfinite interpolation with coons surface patches has to be applied. We showed in Section 4.6.3 that even though point distribution is uniform, the *subdivision* approach gives a better surface representation. The investigation of the influence of mesh generation errors revealed that the errors from mesh generation can be dominant and much larger than polynomial approximation

errors and that an increased number of interpolation points with the same mesh generation errors will not improve the geometry approximation.

Finally, we are able to curve unstructured meshes internally with the *mesh deformation* approach to prevent inverted elements, but a high order solver needs to be implemented and computational costs will increase for large meshes.

5. Parallelization Aspects

5.1. Preprocessing for Parallel Simulations

In this section we describe the code development done for preparing the unstructured meshes for parallel simulations. A simulation of a complex three-dimensional flow with a mesh consisting of several thousands of high order elements must be run in parallel. To run a parallel simulation, following steps need to be performed:

1. Partition the mesh and assign a domain to each core of the parallel simulation

2. Find the inter-element connections of the mesh, especially domain-to-domain interfaces for data communication

These steps typically introduce global operations and therefore are not scalable with an increasing number of cores used for the simulation. In addition, the parallel implementation is very cumbersome.

It is clear that for a given mesh, the steps are always the same, thus would be repeated at the start of each simulation. This is why instead of a parallel implementation, we encapsulate the steps in the preprocessor HOPR, which already has the mesh information because of the curved mesh generation. A graph partitioning approach, like Metis, requires the number of domains *a-priori*. Instead, the domain decomposition is done with the space-filling curve approach, which is much more flexible, since it allows a partitioning with an the number of domains given *a-posteriori*. Examples of space-filling curves are shown in Fig. 5.1, and a comparison of the partitioning methods is done in Section 5.1.2.

HOPR is designed to provide an extended and *parallel readable* mesh file (using the binary HDF5 format). The mesh file includes element connectivity lists and high order curved element information. The mesh file consists of a one-dimensional list of elements sorted along a space-filling curve. The data is stored in an array containing blocks for each element, which enables fast parallel I-O. As the element list is fixed and follows a space-filling curve, the mesh

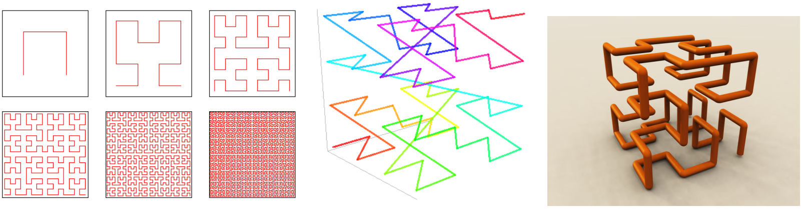


Figure 5.1.: Examples of space-filling curves, a 2D Hilbert curve with different refinement levels (left), a 3D Morton or Z-curve (middle) and 3D Hilbert curve (right), source: Wikipedia.

partitioning becomes very simple. The one-dimensional list can be distributed easily on an *arbitrary number* of domains. The number of domains is only limited by the number of elements in the mesh. Note that *HOPR* only needs to be run *once* for a given mesh and provides a single HDF5 mesh file. The simulation code reads this file in parallel at each start or restart of a simulation, and the number of cores can be arbitrarily chosen.

5.1.1. Sorting of Elements along the Space-Filling Curve

Space-filling curves (SFCs) are widely used to accelerate operations or data structures for large sets of arbitrarily distributed objects, for example nearest neighbor searches [36] or adaptivity [25], but also for domain decomposition and dynamic load balancing [2, 55]. They map point positions in multi-dimensional space to points on a one-dimensional curve, with the property that neighboring points in multi-dimensional spaces are located as close as possible on the curve. SFCs are fractal objects, and each recursive step refines the curve in each direction by a factor two, as shown in Fig. 5.2. In fact, SFCs are strongly connected to quadtrees and octrees [25], as they split the domain in quadrants and octants and handle different resolutions in space adaptively.

The starting point for the element sorting is a cubical bounding box of the computational domain, defined by the minimum and maximum coordinates of

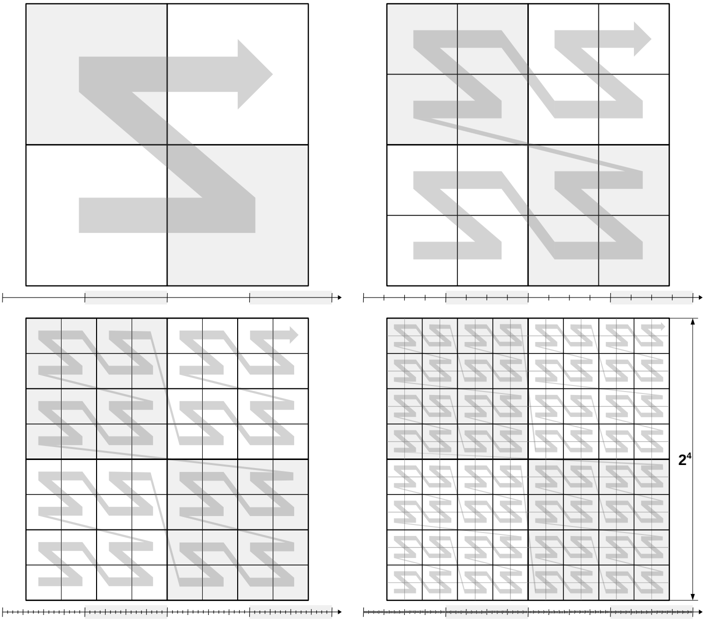


Figure 5.2.: Four levels of the two-dimensional Morton space filling curve (z-curve) and the one-dimensional curve below. Corresponding volumes and one-dimensional ranges are highlighted.

the set of all mesh points $\mathbb{M}$

$$
\begin{aligned}
(x_{\min}, y_{\min}, z_{\min}) &= (\min_{\mathbb{M}}(x), \min_{\mathbb{M}}(y), \min_{\mathbb{M}}(z)), \\
L_x = L_y = L_z = L_{\text{box}} &= \max\left(\max_{\mathbb{M}}(x - x_{\min}), \max_{\mathbb{M}}(y - y_{\min}), \max_{\mathbb{M}}(z - z_{\min})\right)
\end{aligned}
\tag{5.1}
$$

In theory, the space-filling curve is continuous since the curve is a fractal. As we have to express the position discretely along the curve using a numerical value, the index range of the curve is limited by the type of integer we use for its representation. If we choose a 8 Byte signed integer, we cover the range $[0; 2^{63} - 1]$. This gives us a maximum resolution of the three-dimensional bounding box with an edge resolution of

$$\sqrt[3]{2^{63}-1} > 2^{20} \approx 10^{6} \tag{5.2}$$

which corresponds to a maximum of 20 octree levels. Hence, we can assign a unique index for a volume with an edge length $10^{-6}L_{\text{box}}$.
To sort the elements, we express each element barycenter inside the bounding box as an index triple of integers

$$(i,j,k) \in [0;2^{20}]^{3}\,, \tag{5.3}$$

which is the input data for the SFC algorithm to compute the index on the SFC. A non-contiguous list of indices for all elements of the mesh is generated, which is easily sorted with a QuickSort algorithm.
Note that the SFC is only implicitly evaluated as a unique mapping between $i_{SFC} \leftrightarrow (i,j,k)$, and the evaluation is very fast. In contrast to graph partitioning, we only need the element barycenter positions instead of the global element connectivity information, leading to very low memory consumption. Once the elements are sorted, the domain decomposition consists of splitting the one-dimensional element list into equal parts, choosing an arbitrary number of domains. In the context of load balancing, for example if elements have different polynomial degrees and therefore different computational cost, element weights can be introduced and the element list is then split into equally *weighted* parts.
The Morton or Z-curve [81] is easy to implement, the index of a point in multiple dimensions can be calculated by interleaving the binary representations of its coordinate values, see [36]. The three-dimensional Hilbert curve [59] is not unique, and several implementations can be found. We employ the C-code of Doug Moore [80], which is based on the algorithm in [26].

5.1.2. Domain Decomposition with the Space-Filling Curve

In this section, we compare the partitioning of the Metis graph-partitioner [65] with the space-filling Curve (SFC) approach on the same mesh, for an increasing number of domains. We use the ratio of the mean number of inter-domain faces per core to the mean number of elements per core as a quality measure. It can be interpreted as the ratio between communication data and local data operations. The lower this ratio, the better for the parallel computation. We compare both approaches on an unstructured hexahedral mesh of a laser-cutting nozzle in free-stream configuration with 110,000 elements. The domain decomposition of the nozzle mesh on 160 cores using the SFC is shown in Fig. 5.3.

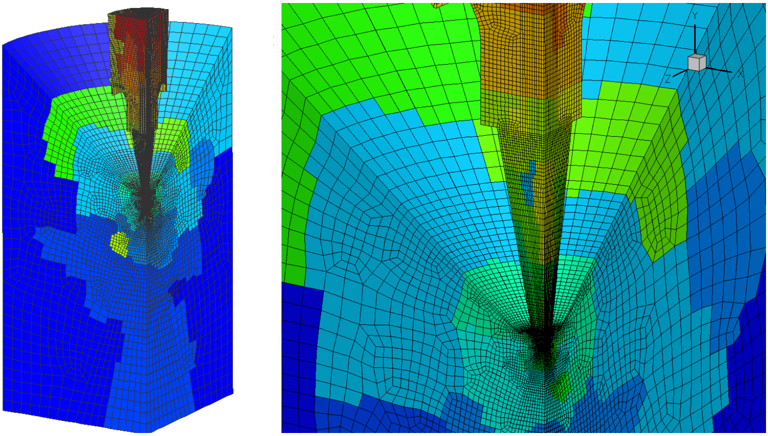


Figure 5.3.: Domain decomposition on 160 cores using the space-filling curve.

An estimate of the surface/volume ratio is found on a periodic cartesian hexahedral mesh, where each domain has a size of $n_e = n_x n_y n_z$ elements and the ratio is

$$R_{\text{cart}} = \frac{n_{\text{surfs}}}{n_e} = \frac{2(n_x n_y + n_x n_z + n_y n_z)}{n_x n_y n_z}\,, \tag{5.4}$$

with the smallest ratio for $n_x = n_y = n_z$

$$R_{\text{cart}} = \frac{6(n_e^{1/3})^2}{n_e} = 6n_e^{-\frac{1}{3}}\,. \tag{5.5}$$

This ratio is only an estimate and can be slightly smaller or larger for an unstructured mesh with boundary conditions. It is clear that for an increasing number of domains, the ratio increases up to the limit of $R_{cart} = 6$ for 1 element per core.

In Table 5.1 the ratios for Metis and space-filling curve are summarized. Metis is a graph-based partitioner, whereas the space-filling curve does not account for the mesh topology. Especially for a low number of cores, Metis will give better results, since it accounts for non-communication faces on the boundaries of the domain. Due to the boundary conditions, Metis reaches lower ratios than the SFC.

#Domains	8	16	40	80	160
#Elems/Domain	13760	6880	2752	1376	688
R_{cart}	0.250	0.315	0.428	0.539	0.680
R (Metis)	0.094	0.170	0.317	0.467	0.652
R (SFC)	0.204	0.349	0.562	0.729	0.948
R (SFC) / R (Metis)	2.17	2.05	1.77	1.56	1.45

Table 5.1.: Comparison of the Metis and space-filling curve (SFC) mesh partitioning.

However, the comparison shows that the difference between both approaches diminishes for increasing number of cores. Thus, for *HOPR*, the SFC approach is employed, since data structures and the domain decomposition process are greatly simplified and domain decomposition results are similar on large number of cores.

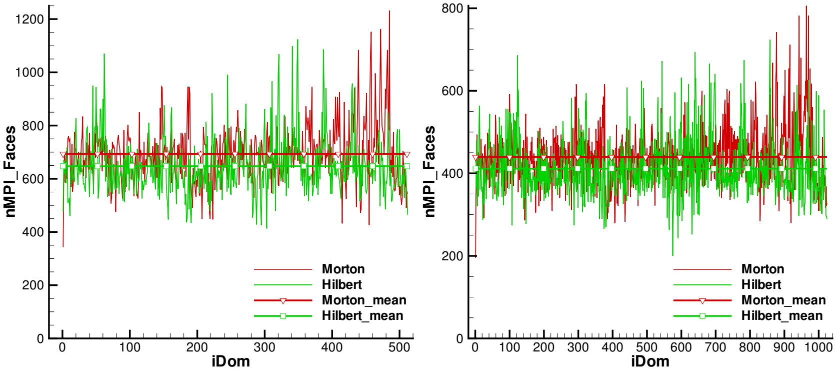


Figure 5.4.: Comparison of the number of inter-domain faces (nMPI_faces) for the Hilbert and Morton curve, on 512 (left) and 1024 domains (right), plotted over all domains.

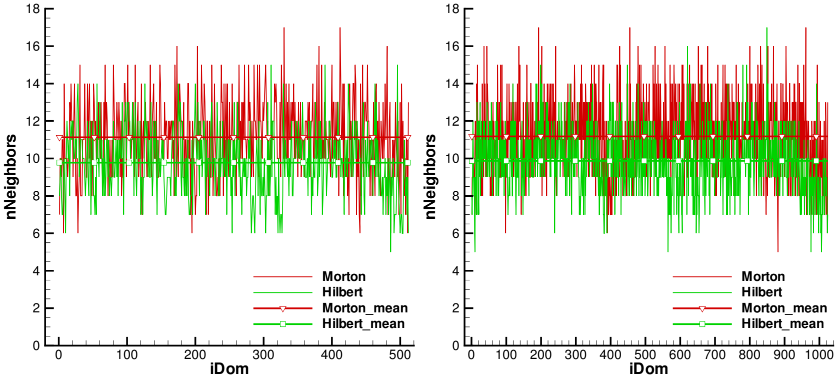


Figure 5.5.: Comparison of the number of domain neighbors (nNeighbors) for the Hilbert and Morton curve, on 512 (left) and 1024 domains (right), plotted over all domains.

The space-filling curves are not unique, the Morton curve and the Hilbert were implemented. We compare both on a cartesian stretched hexahedral mesh for a 3D jet flow, consisting of 357, 216 elements. The comparisons in Fig. 5.4 and Fig. 5.5 show that the number of MPI neighbor cores and the number of MPI faces is lower for the Hilbert curve on both domain decompositions of 512 and 1024 cores. We support the theoretical findings that the domains are more compact for the Hilbert curve, leading to a better domain decomposition than the Morton curve.

5.2. Parallelization Concept

Three-dimensional simulations are computationally very demanding, and as computer resources constantly cheapen, the scalability of numerical algorithms becomes more important to tackle large scale computations or to decrease the runtime of simulations. In this section, we present the parallelization concept of the DG-SEM code *Flexi*.

Besides the promising fundamental efficiency of the DG-SEM, a main advantages of DG schemes is their "high performance computing" capability. The DG algorithm with explicit time-stepping is inherently parallel, since all elements do communicate only with their direct neighbors via solution and flux exchange. Independent of the local polynomial degree, only the exchange of surface data between direct neighbors is necessary. Note that the DG operator can be split into the two building blocks, namely the volume integral – solely depending on element local DOF – and the surface integral, where neighbor information is needed. This fact helps to hide communication latency by exploiting local element operations and further reduces the negative influence of data transfer on efficiency. It is therefore possible to send surface data while simultaneously performing other data operations, as depicted in Fig. 5.6. This communication is realized via *non-blocking* send and receive commands provided by the MPI standard library [77]. In other words, the receive buffer is like a mailbox, and the receive checks the mail and would only wait if the communication has not yet finished. Latency hiding works if the network communication is faster than the buffer operations.

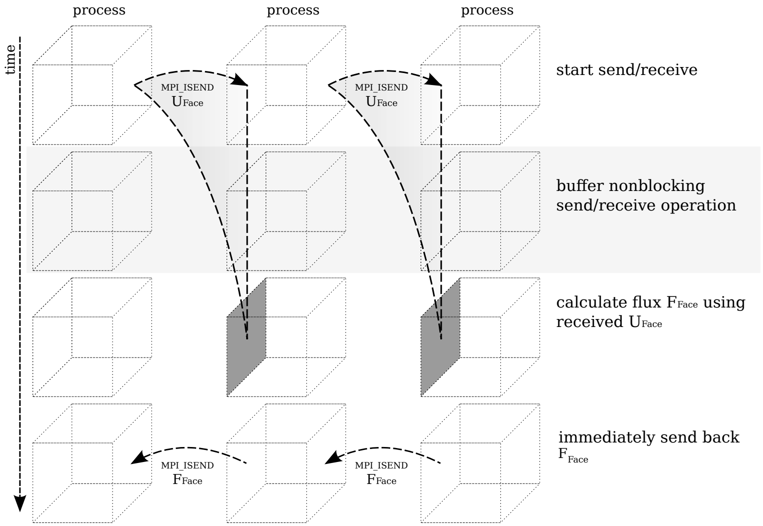


Figure 5.6.: Communication pattern for inter-processor flux computation with non-blocking communication.

To explain the details of the parallelization concept, we start by highlighting the data structures used in *Flexi*. The solution U is represented at the interpolation nodes of the volume, see Section 3.3. The data structure separates volume and surface data. All element faces or 'sides' except the boundaries form a pair, and element sides are either master or slave. The list of sides is sorted by side type. The first group are boundary sides. The second includes the inner sides. The third group are sides which communicate to neighbor domains. The three groups are abbreviated as BC-sides, Inner-sides and MPI-sides. The numerical flux between element sides is unique, and is computed once. For MPI-sides, the solution is sent from the slave and received on the master side, the flux is only computed on the master side and then sent back to the slave side, as shown in Fig. 5.6. Hence, except for communication, no additional operations are introduced, which is an important property of a scalable algorithm.

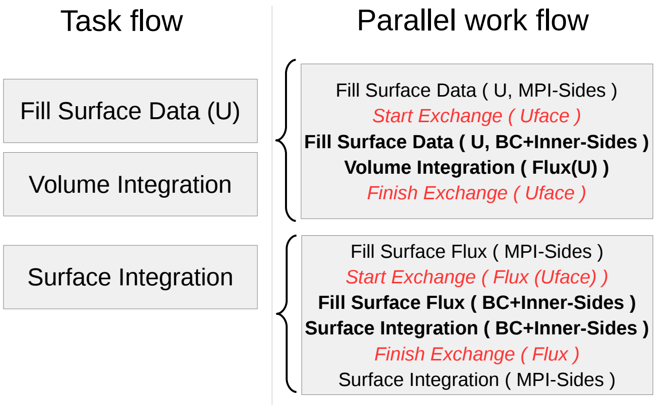

Task flow
Parallel work flow
Fill Surface Data (U)
Volume Integration
Surface Integration
Fill Surface Data (U, MPI-Sides)
Start Exchange (Uface)
Fill Surface Data (U, BC+Inner-Sides)
Volume Integration (Flux(U))
Finish Exchange (Uface)
Fill Surface Flux (MPI-Sides)
Start Exchange (Flux (Uface))
Fill Surface Flux (BC+Inner-Sides)
Surface Integration (BC+Inner-Sides)
Finish Exchange (Flux)
Surface Integration (MPI-Sides)

Figure 5.7.: Flow chart of the parallel DG operator for hyperbolic equations, with **buffer** and *communication* routines.

Fig. 5.7 shows the flowchart of the parallel DG operator implementation for hyperbolic equations. As a first step, we interpolate the solution at the surface (fill surface data). MPI-sides are treated first and the data is sent immediately. At BC- and Inner-sides, the interpolation represents a buffer to hide the first communication. The volume integration is also a buffer routine, because it depends only on local volume data. The communication of the surface data must be finished before the surface flux at MPI-sides can be calculated. Now again, the flux at MPI-sides is computed first and sent, followed by the flux computations and the surface integral at BC- and Inner-sides as buffer. After the flux is received, the surface integration adds the surface flux contribution at the MPI-sides to the volume data, and the evaluation of the parallel DG operator is finished.

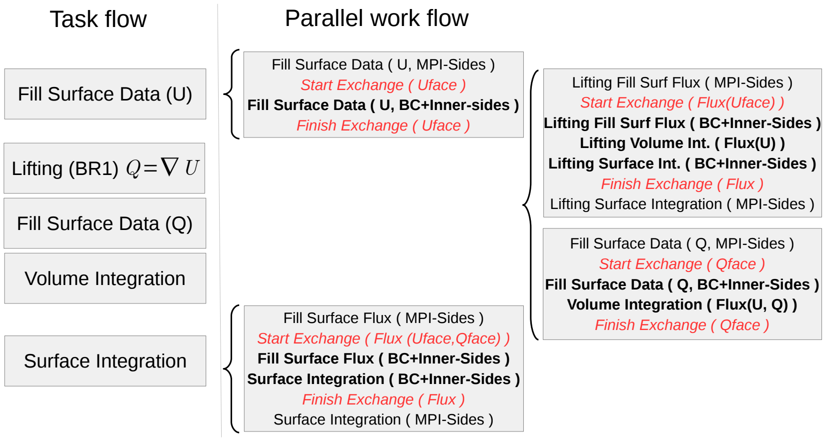

Task flow
Parallel work flow
Fill Surface Data (U)
Lifting (BR1) $Q = \nabla U$
Fill Surface Data (Q)
Volume Integration
Surface Integration
Fill Surface Data (U, MPI-Sides)
Start Exchange (Uface)
Fill Surface Data (U, BC+Inner-sides)
Finish Exchange (Uface)
Lifting Fill Surf Flux (MPI-Sides)
Start Exchange (Flux(Uface))
Lifting Fill Surf Flux (BC+Inner-Sides)
Lifting Volume Int. (Flux(U))
Lifting Surface Int. (BC+Inner-Sides)
Finish Exchange (Flux)
Lifting Surface Integration (MPI-Sides)
Fill Surface Data (Q, MPI-Sides)
Start Exchange (Qface)
Fill Surface Data (Q, BC+Inner-Sides)
Volume Integration (Flux(U, Q))
Finish Exchange (Qface)
Fill Surface Flux (MPI-Sides)
Start Exchange (Flux (Uface,Qface))
Fill Surface Flux (BC+Inner-Sides)
Surface Integration (BC+Inner-Sides)
Finish Exchange (Flux)
Surface Integration (MPI-Sides)

Figure 5.8.: Flow chart of the parallel DG operator for parabolic equations with lifting (BR1 scheme), with **buffer** and *communication* routines.

For equations with parabolic terms, like the Navier-Stokes equations, the gradient of the solution is needed, and the jump of the solution at the element interfaces must be considered for discontinuous approximations, which is known as *Lifting*. We employ the Bassi-Rebay 1 lifting scheme, [6]. The parallel scheme is shown in Fig. 5.8. The parallel concept of the hyperbolic DG operator is applied analogously to the Lifting equations. The only difference in the algorithm is that the volume integral must be called after the volume gradient is computed. In the limit of one element per core, all sides are MPI-sides and the volume integral remains the only buffer routine.

Operations at MPI-sides must be done before sending and after receiving. The domains have equal number of elements, but especially on unstructured meshes, the number of MPI-sides and per domain varies. We compute the flux only once on the master MPI-side, and a bad distribution of master sides could cause a

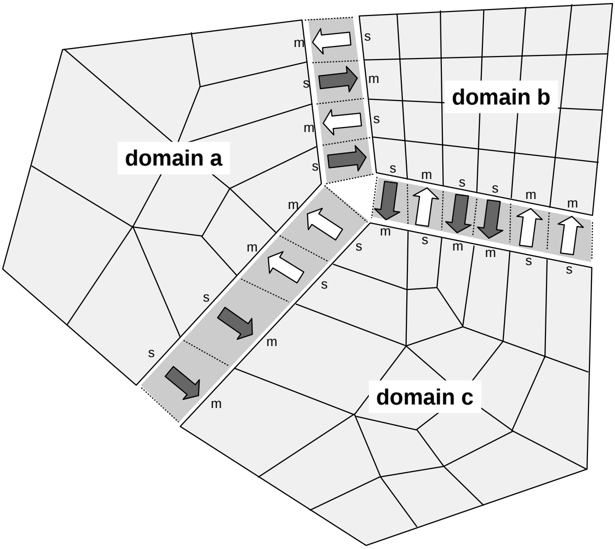


Figure 5.9.: Load balancing of the inter-processor communication data, by assignment of master (m) and slave (s) sides.

load imbalance. As a simple remedy, the MPI-sides are divided into subgroups of sides associated to each neighbor domain, and the master/slave switches are reassigned so that half of the subgroup sides are master and the other slave. This is sketched in Fig. 5.9. Even though not considered up to now, one could adapt the distribution of sides between the domains to compensate for load imbalances, for example caused by an uneven number of elements per domain.

In Table 5.2 we provides estimates of the operation counts (only multiplications) for the DG-SEM with Gauss points on a hexahedron. For the surface integral, which is evaluated once per interface, we count only half of the element faces. The evaluation of three-dimensional compressible Navier-Stokes fluxes ($n_{\mathrm{var}} = 5$) is an estimate, and the computation of Riemann fluxes is not included, since these are only two-dimensional operations. The overall estimates show that the number of operations per DOF for hexahedra using DG-

SEM scales with $\mathcal{O}(N)$, whereas DG methods without dimension-by-dimension splitting typically scale with $\mathcal{O}(N^3)$ in three dimensions, since the 3D stiffness matrix has a size of $\mathcal{O}(N^6)$, see Section 3.2.

DOF_e	$(N+1)^3$
1. prolong to face: 2. transform fluxes: 3. volume integral: 4. surface intergal: 5. eval. 3D fluxes (N-S):	$n_{\text{var}}6(N+1)^2(N+1)$ $n_{\text{var}}9(N+1)^3$ $n_{\text{var}}3(N+1)^2(N+1)^2$ $n_{\text{var}}3(N+1)^3$ $\approx 100(N+1)^3$
update per DOF: $(\sum_{1.}^{5.})/\text{DOF}_e$	$(n_{\text{var}}(18+3(N+1))+100)$
gradients per DOF: $3\times(\sum_{1.}^{4.})/\text{DOF}_e$	$3(n_{\text{var}}(18+3(N+1)))$

Table 5.2.: Estimated operation counts for DG-SEM for one hexahedron.

The total number of operation count of one element per DOF for the Navier-Stokes DG-SEM operator with gradient evaluation ($n_{\text{var}}=5$) is

$$n_{\text{ops}}^{\text{NS}}/\text{DOF}_e = (60(N+1)+460)\,, \tag{5.6}$$

which shows that for low polynomial degrees, even though the scaling is $\sim \mathcal{O}(N)$, the operation count per DOF increases slowly with N, since only the volume integral scales with $(N+1)^4$. We evaluated (5.6) in Table 5.3 for $N=2-10$.

$N=$	2	3	4	5	6	7	8	9	10
$n_{\text{ops}}^{\text{NS}}\times 10^3$	17.28	44.8	95.0	177.1	301.8	481.3	729	1060	1491
$n_{\text{ops}}^{\text{NS}}/\text{DOF}_e$	640	700	760	820	880	940	1000	1060	1120

Table 5.3.: Operation count and operation count per DOF for one hexahedra for Navier-Stokes DG-SEM operator with gradient evaluation.

5.3. Parallel Performance Analysis

In this section, we show the results of the performance analysis. We investigate the weak and strong scaling of the explicit DG-SEM code *Flexi* and the influence of the polynomial degree N of the solution inside the element. Moreover, we define a performance index to assess the efficiency of the code and show that up to a minimum load of roughly 2000 DOF per domain, an optimal scaling is found, being independent of the polynomial degree of the DG solution. The scaling analysis will be based on cartesian meshes, but at the end of the section we evaluate the performance of a simulation of the flow past a sphere and show that the results can be generalized to unstructured meshes.

5.3.1. Hardware and Memory Consumption

All simulations were conducted on the CRAY XE6 cluster at the supercomputing center in Stuttgart (HLRS). The cluster has 3552 nodes and two sockets per node. Each socket is an AMD Opteron Interlagos processor with 2.4 GHz and 16 cores. A standard node has 32 GB RAM (1GB per core) and 2 × 8MB L3 and 8 × 2MB L2 cache per node (2MB cache per core). The intra-node Hyper-Transport 3 has a memory bandwidth of 103.4 GB/s. The node-interconnect is realized via the CRAY Gemini Infiniband switch.

Note that we assign a MPI-domain to each core and consequently refer to the *cores* of a simulation in the following, instead of the commonly used word *processors*.

element volume data :		surface data per element :	
U_t^*, U_t, U :	$3 \times n_{\text{var}} \times (N+1)^3$	$U_{2D} \times 6$:	$6 \times n_{\text{var}} \times (N+1)^2$
∇U :	$3 \times n_{\text{var}} \times (N+1)^3$	$\nabla U_{2D} \times 6$:	$6 \times 3 \times n_{\text{var}} \times (N+1)^2$
Ja :	$3 \times 3 \times (N+1)^3$	Flux $\times 3$:	$3 \times n_{\text{var}} \times (N+1)^2$
$\vec{x}$:	$3 \times (N+1)^3$	$\vec{n}, \vec{t}_1, \vec{t}_2, \times 3$:	$3 \times (3 \times 3) \times (N+1)^2$
J :	$1 \times (N+1)^3$	$\hat{s} \times 3$:	$3 \times (N+1)^2$
$n_{3D} = (6n_{\text{var}} + 10) \times (N+1)^3$		$n_{2D} = (27n_{\text{var}} + 30) \times (N+1)^2$	

Table 5.4.: Number of REAL variables (8 Byte) per element for a given polynomial degree N and number of conserved variables n_{var}.

An estimate of the memory consumption of one hexahedral element is given in Table 5.4 for the DG-SEM code *Flexi*. The surface data of all six element sides is stored, but surface metrics are only stored on the master side, with a median of 3 master sides per element. The compressible 3D Navier-stokes equations have $n_{\text{var}} = 5$ conserved variables. Note that the DG-SEM operators

$$\underline{\underline{\hat{\boldsymbol{D}}}} \in \mathbb{R}^{[(N+1)\times(N+1)]};\ \underline{\ell}(\pm 1),\ \underline{\hat{\ell}}(\pm 1)\ \in \mathbb{R}^{[N+1]} \tag{5.7}$$

are stored once for all elements and are neglectable in terms of memory consumption.

N	2	3	4	5	6	7	8	9	10
Estimated memory per Element [kB]	20	41	71	114	170	242	332	441	571
Measured memory per Element [kB]	22.8	46.1	80.9	130	194	279	383	510	662
n_e in Cache	87	43	24	15	10	7	5	3	3
DOF in Cache	2349	2752	3000	3240	3430	3584	3645	3000	3993

Table 5.5.: Estimated and measured memory consumption per element for $n_{\text{var}} = 5$ and $N = 2 - 10$, as well as the number of elements that fit into 2MB cache.

In Table 5.5, we list the memory estimate per element for different polynomial degrees and $n_{\text{var}} = 5$. We compare the estimates to measured memory consumption during program execution. The memory consumption per element was derived from the total memory consumption of a simulation with a large number of elements. The real memory consumption is roughly 15% larger than the estimate. From the measured memory consumption, we deduce the number of elements that fit into a 2MB cache, as well as the resulting DOF, related to the number of elements and the polynomial degree by $\text{DOF} = n_e(N+1)^3$.

5.3.2. Strong Scaling

Strong scaling expresses the speedup of the same simulation when increasing the number of cores. A ideal scaling is found when the speedup increases linearly with the number of cores, thus a speedup of factor 2 when doubling the number of cores.

We use a sequence of refined cartesian meshes of a cube with periodic boundary conditions to minimize the influence of load imbalance. We start with a $n_e = (2 \times 2 \times 2)$ mesh and increase the number of elements by a factor of two in each direction, $n_e = (2 \times 2 \times 4)$, $(2 \times 4 \times 4)$, $(4 \times 4 \times 4)$, $(4 \times 4 \times 8)$ and so on. The number of elements will always be a multiple of 2, $n_e^k = 2^{3+k}$.

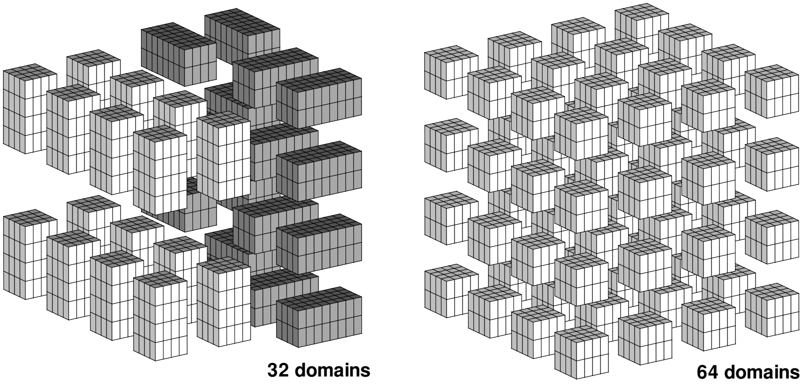


Figure 5.10.: Domain decomposition of a $16 \times 16 \times 8$ mesh into 32 and 64 domains, using the Hilbert space filling curve. Domains are scaled by a factor 0.5 for visualization.

For each mesh, we increase the number of cores by a factor of two, starting at $n_c = 8$ cores up to the maximum limit of one element per core $n_c = n_e$. The domain decomposition of a $16 \times 16 \times 8$ mesh with the Hilbert space filling curve is shown in Fig. 5.10. Due to the properties of the space filling curve, a perfect domain decomposition is only found if the number of cores is $n_c = 8, 64, 512, 8^4, \ldots$. Even though for $n_c = 32$ the domains remain blocks with the same number of elements but different shape ($4 \times 4 \times 4$ and $8 \times 4 \times 2$) and some domains have 7 instead of 6 neighbors.

In the top of Fig. 5.11, we show the strong scaling results for three meshes with $128, 1024$ and 8192 elements, keeping a constant polynomial degree $N = 5$. The maximum number of cores is limited by the number of elements, the rightmost simulation corresponds to one element per core ($n_{e/c} = 1$), and the number of elements per core doubles in each step to the left. Note that logarithmic axis were chosen, which helps to see all simulations, but impedes a visual judgement of the efficiency, and therefore the efficiency in percent for the last entry of each curve was added.

The speedup for small number of cores is even higher than the ideal, which is a caching effect, since the load per core decreases. For all three meshes, the speedup falls below the ideal speedup at the same number of elements

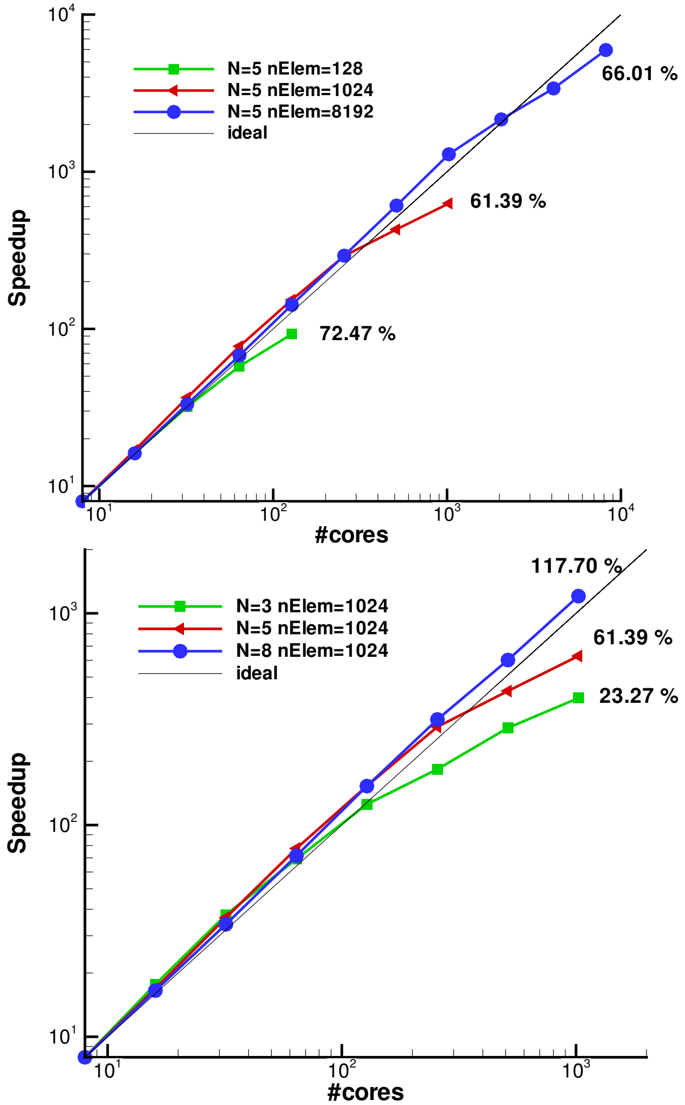


Figure 5.11.: Strong scaling for $N = 5$ and three mesh resolutions (top) and on the same mesh with an increasing polynomial degree (bottom), starting from $n_c = 8$ cores.

per core, $n_{e/c} = 4$. At this point, the limit of the latency hiding seems to be reached, meaning that the computation time for the buffer routines is less than the communication time. Note that the limit of the ideal speedup will depend on the architecture of the cluster, since the ratio of computational and communication bandwidth will differ.

The second parameter to look at is polynomial degree N. We keep a constant number of elements of 1024 and plot the speedup for $N = 3, 5, 8$ in the bottom of Fig. 5.11. The high polynomial degree of $N = 8$ has an optimal speedup up to one element per core, whereas $N = 3$ falls below the ideal speedup for $n_{e/c} = 8$ elements per core. However, the resolution as well as the number of operations changes for different polynomial degrees, and we should express the load per core with the number of DOF instead of the number of elements

$$\mathrm{DOF}/c = n_{e/c}(N+1)^3. \tag{5.8}$$

For $N = 3$, we get $\mathrm{DOF}/c = 512$, $\mathrm{DOF}/c = 864$ for $N = 5$ and $\mathrm{DOF}/c = 729$ for $N = 8$, which indicates that the ideal speedup stops at about $500-1000\,\mathrm{DOF}/c$.

5.3.3. Performance Index

For a better comparison of different meshes and polynomial degrees, we define the performance index PID, to be the mean computation time per degree of freedom and per timestep. The PID is computed for a given parallel simulation as

$$\mathrm{PID} = \frac{\text{runtime} \times \#\text{cores}}{\#\text{timesteps} \times \mathrm{DOF}} \quad [\mu s/\mathrm{DOF}]\,. \tag{5.9}$$

For all simulations, the time integrator is a fourth order low-storage Runge-Kutta scheme with 5 stages, thus calling the explicit Navier-Stokes DG operator 5 times per timestep.

The results from the strong scaling plots are summarized in Fig. 5.12 by plotting the PID over the load per core. The strong scaling plots are reversed, since the smallest load per core is now on the left, and a theoretically ideal behavior would be represented by a constant PID. It is remarkable that for all polynomial degrees, a local minimum of the PID is found at about 1000 DOF per core, representing the optimal load for the code on this architecture. To the left of the optimum, the load is too small and therefore communication costs dominate, to the right of the optimum, the caching effect vanishes and at very high load per core, the local data operations dominate.

In Fig. 5.13, we plot the performance index at a constant number of cores and an increasing load per core, for polynomial degrees of $N = 3, 4, 5$ and 8.

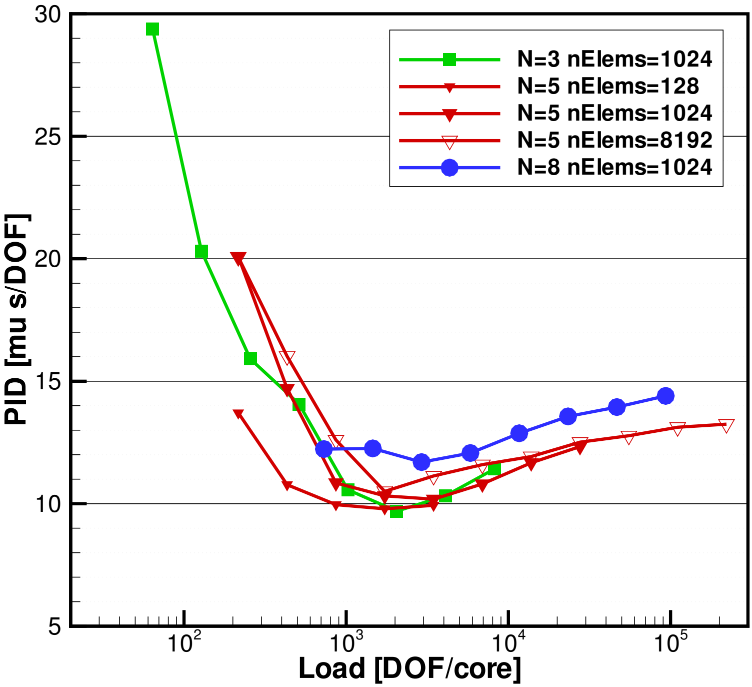


Figure 5.12.: Performance index of the strong scaling simulations, over the load per core.

We can see that the optimal load is still found at the same location, and the optimal performance index is only slightly increasing from 8 to 1024 cores. It is clear that the communication overhead to the left of the optimum is more pronounced for an increasing number of cores. We sketch the basic behavior of the PID over the load in Fig. 5.14, with the optimum range of $10^3 - 10^4$ DOF/c for the code on this machine.

In Section 5.2, we estimated the operation count for one DG-SEM element, with a theoretically linear behavior $\mathcal{O}(N)$, see (5.6). However, the performance of the implemented code is influenced by the programming language, data structures, the hardware architecture, memory consumption and last but not

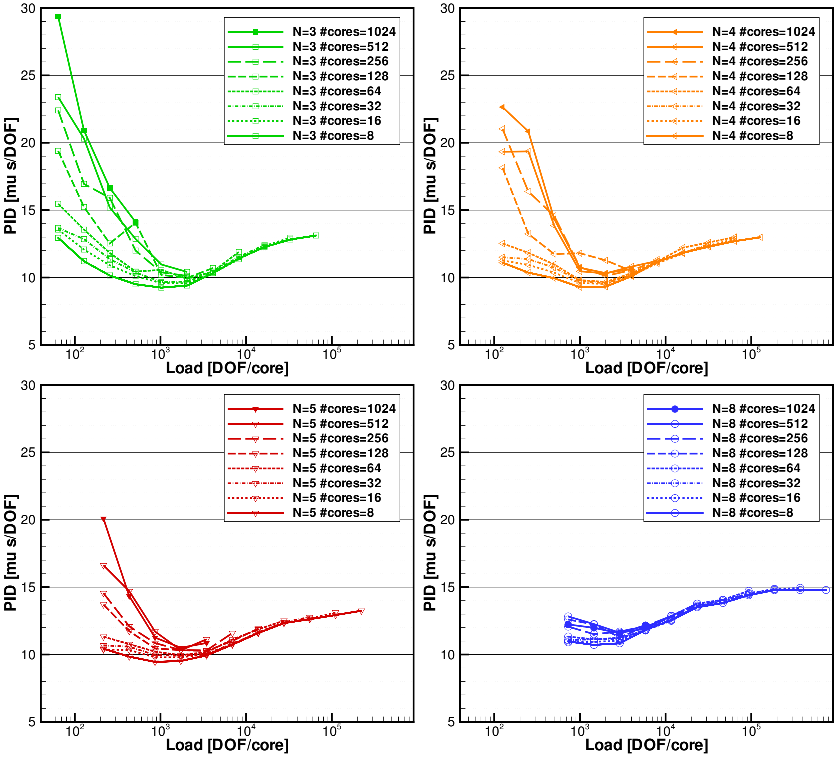


Figure 5.13.: Performance index for an increasing load and a fixed number of cores, ranging from $8-1024$, for polynomial degrees of $N = 3, 4, 5$ and 8.

least the compiler optimizations. In Fig. 5.15, we plot the PID at a constant load per core and see that for a range of polynomial degrees $N = 3-9$, the PID is nearly constant and does not follow the theoretical increase from the operation count estimate (5.6). It seems that high polynomial degrees become faster, since the loops are longer, matrices larger and data structures are more compact. Generally speaking, the computational costs are the same for

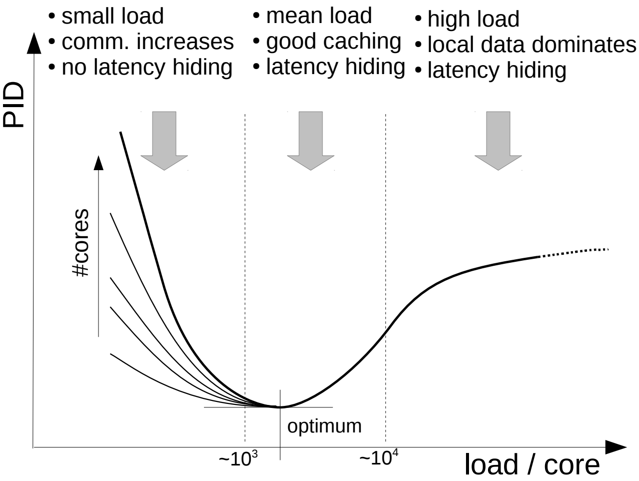


Figure 5.14.: Behavior of the performance index over the load per core.

a high order DG-SEM simulation with the same number of degrees of freedom as for a low order simulation, and the high order simulation is likely to be more accurate. To our knowledge, the DG-SEM is the only DG scheme to feature a constant computational cost per degree of freedom, independent of the polynomial degree.

In fact, the PID is a measure to evaluate the performance of completely different codes. In [43], we did a direct numerical simulation of a three-dimensional turbulent shear layer flow, using DG-SEM and a polynomial degree of $N = 5$, and compared to the simulations of the well-established 6th order compact Finite Difference (cFD) code NS3D by Babucke [5], using the same physical setup with a slightly higher resolution and running on the same machine (NEC-Nehalem cluster, HLRS). The amplification rates for the transition to turbulence were compared and the simulation results agreed well. The comparison of the performance revealed that the DG-SEM code was faster than the Finite Difference code, both regarding to total simulation time and performance index, see Table 5.6.

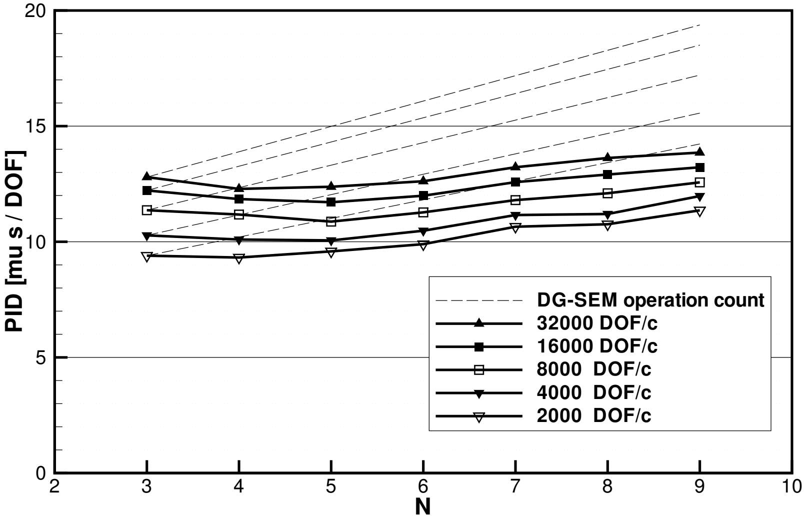


Figure 5.15.: Performance index over the polynomial degree N at constant load per core.

	PID $[\mu s/DOF]$	DOF	timestep	total sim. time
DG-SEM	9.90	26,376,192	1.270E-02	4596 core-h
cFD code NS3D	15.38	19,125,000	0.998E-02	6215 core-h

Table 5.6.: Comparison of the performance of explicit DG-SEM and cFD codes for the simulation of a turbulent shear layer, from [43].

5.3.4. Weak Scaling

The weak scaling property is defined as the scaling of the simulation time over the number of cores when keeping the load per core constant, which means that the problem size increases with the number of cores. It evaluates the ability of the code to deal with large scale simulations.

It is well known that for explicit time integration, the problem size has little effect on the performance, and the weak scaling is more or less a check on the parallel implementation, but as well on the communication hardware, since the number of messages and the distance between cores increases. We have to mention that in the following weak scaling results, the simulation time per time step is evaluated to compute the weak scaling, which excludes any influence of the timestep.

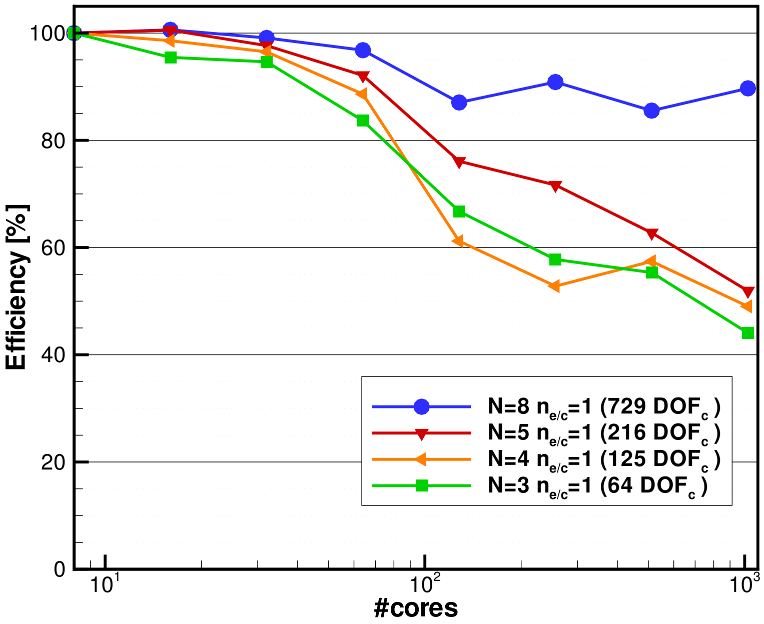


Figure 5.16.: Weak scaling for the limiting case of one element per core and different polynomial degrees.

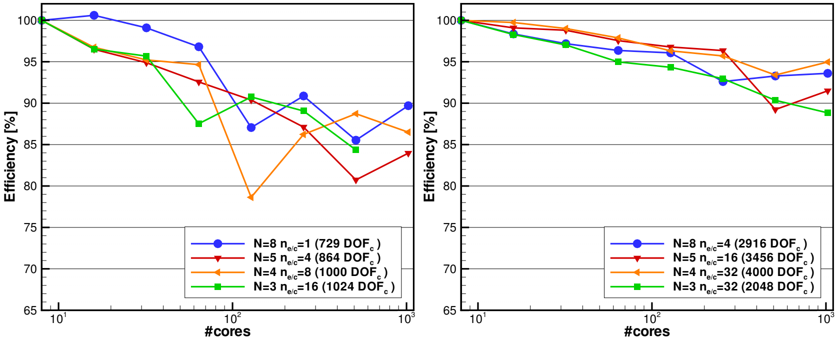


Figure 5.17.: Weak scaling for different polynomial degrees and a similar load per core, left $729 - 1024\,\mathrm{DOF}/c$ and right $2048 - 4000\,\mathrm{DOF}/c$.

As a consequence of the previous investigations, we know that the performance index is independent of the number of cores towards larger loads per core. The limiting case of one element per code scales well only for high polynomial degrees, see Fig. 5.16. If we increase the load for the low polynomial degrees and plot the weak scaling for the same number of DOF per core, the weak scaling efficiency is $> 80\%$, and up to 90% for a load $> 2000\,\mathrm{DOF}_c$, see Fig. 5.17.

5.3.5. Sphere Flow Simulation

The parallel performance analysis in Section 5.3 is based on cartesian meshes, and a perfect load balance was achieved. In this section, we investigate the scaling of the code for unstructured meshes. The flow past a sphere of diameter D is simulated on the unstructured mesh shown in Fig. 5.18, which consists of 21,128 hexahedra. The hexahedra at the sphere surface are curved using the point-normal approach with exact normal vectors. The domain extends $25D$ downstream and $4.5D$ upstream and circumferentially.

Load imbalances are unavoidable. For example, if the number of elements is not dividable by the number of cores, but also from the space-filling curve domain decomposition, which only balances the number of elements. Furthermore the number of MPI neighbors, inner sides and MPI sides strongly vary. The statistical distribution for three domain decompositions is listed in Table 5.7. Since the number of elements does not match the number of cores, we find an imbalance of one element between the domains. For a smaller number of elements per core, the computational imbalance created by one element increases. An average number of 11 neighbor domains is found, and reaches up to 20 neighbor domains.

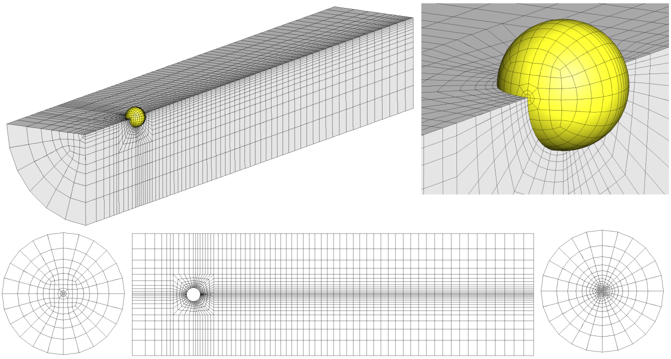


Figure 5.18.: Unstructured mesh of the sphere, 3D views, front view (left), slice and back view.

The surface to volume ratio, thus the ratio of MPI sides to the local number of elements, increases with the number of domains. An estimate of this ratio was already derived in (5.5) for a cubical cartesian domain, $R_{\text{cart}} = 6n_e^{-\frac{1}{3}}$, and equals $(1.09, 2.19, 3.47)$ for the mean domain size from Table 5.7. For the limiting case of one element per domain, the ratio is $R = 6$.
We can conclude that the domain decompositions of the unstructured mesh are far from being optimal, which we illustrate in Fig. 5.19. The domain decomposition is generated from the Hilbert space filling curve. Each domain is contracted towards its barycenter by 95% for visualization. We also plot the communication graph, showing the domain barycenter and the neighbor connections as lines.

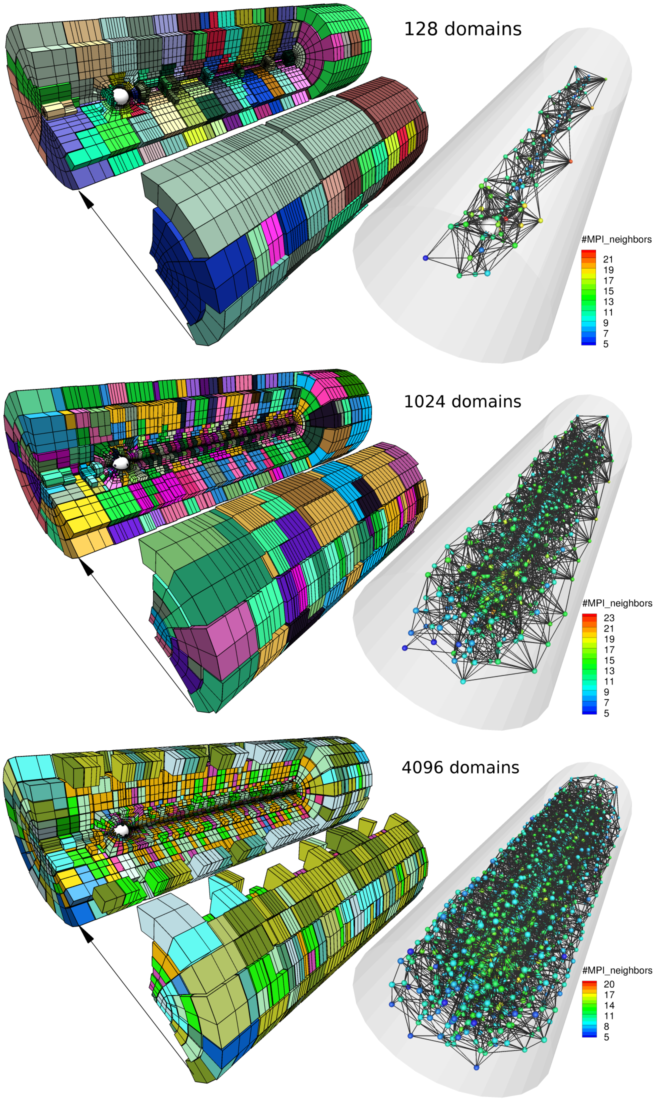


Figure 5.19.: Domain decomposition of the unstructured mesh for the sphere (left) and communication graph between the domains (right), with 128, 1024 and 4096 domains.

	elems.	number of sides				$\frac{n_{\mathrm{s,MPI}}}{n_{e/c}}$	#Neighbors
	$n_{e/c}$	n_s	$n_{\mathrm{s,inner}}$	$n_{\mathrm{s,BC}}$	$n_{\mathrm{s,MPI}}$		
128 domains							
Average	**165.06**	**646.05**	**344.32**	**20.50**	**281.23**	**1.70**	**11.75**
RMS	0.24	21.72	22.06	29.11	49.97		3.46
Min	165	595	279	0	150		5
Max	166	711	401	159	374		23
1024 domains							
Average	**20.63**	**95.08**	**28.72**	**2.56**	**63.80**	**3.09**	**11.99**
RMS	0.48	3.4	2.99	5.14	7.72		2.82
Min	20	84	21	0	34		5
Max	21	104	40	24	80		22
4096 domains							
Average	**5.16**	**27.04**	**3.91**	**0.64**	**22.49**	**4.36**	**10.37**
RMS	0.36	1.95	1.1	1.46	2.63		2.23
Min	5	24	0	0	13		5
Max	6	35	9	8	34		20

Table 5.7.: Statistics of the domain decomposition for 128, 1024 and 4096 domains, with the average, root mean square, minimum and maximum of all domains.

We solve a weakly turbulent flow past a sphere at Mach number $\mathrm{Ma}_\infty = 0.3$ and a Reynolds number $\mathrm{Re} = 1000$. The solution is discretized by a polynomial degree of $N = 4$, yielding 2.64 million degrees of freedom per conserved variable. The computation was done on the CRAY-XE6 cluster on 4096 cores, the computational effort for one convective time unit $T^* = D/u_\infty$ was 100 core-h. The simulation was run for $300T^*$ with a mean timestep of $\Delta t = 1.53{\cdot}10^{-4}T^*$. The physical results of the simulation compare well to values reported by Tomboulides and Orzag [93] and the references therein. The mean drag is $C_d = 0.48$ and the Strouhal number is $\mathrm{St} = 0.32$. At this Reynolds number, small scales are created in the wake of the sphere. A visualization of the vortices using the λ_2 criterion in Fig. 5.20 shows that the behavior of the flow is well captured and a single layer of curved hexahedra with the polynomial degree of $N = 4$ is sufficient to resolve the boundary layer.

The performance analysis was conducted for a range of $128 - 4096$ cores, each running for 10 minutes as a restarted simulation. We compare the strong scaling behavior with the perfectly load balanced cartesian meshes, for the same polynomial degree $N = 4$ in Fig. 5.21. Despite the strong imbalance

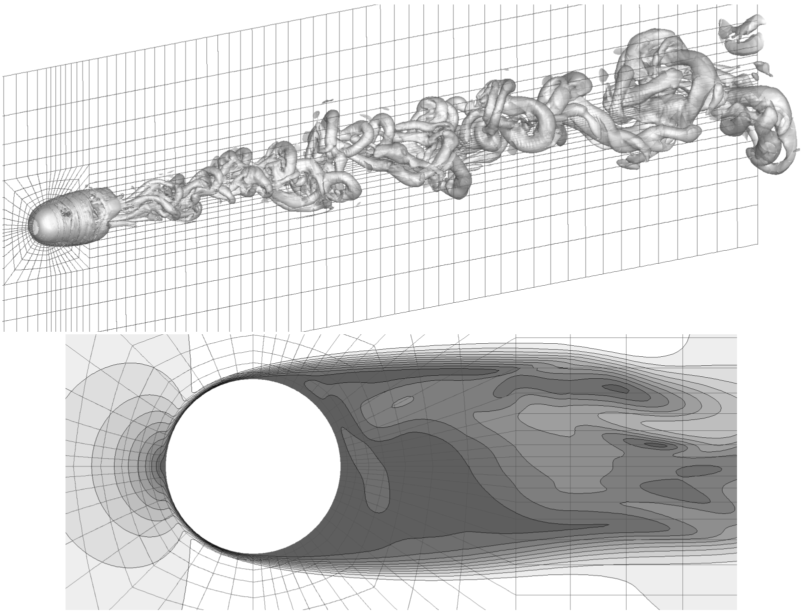


Figure 5.20.: Isosurface of $\lambda_2 = -0.001$ of the sphere flow at Re = 1000 and velocity magnitude contours (levels $[0,1]u_\infty$) in the x-y plane.

caused by the unstructured mesh, an ideal strong scaling is maintained up to 10 elements per core (2048 cores) and the scaling leaves the ideal behavior for 5 elements per core. Plotting the performance index over the load per core reveals that load imbalances slightly shift the optimum to higher loads. We conclude that the code on the CRAY XE6 cluster has the same performance either on an unstructured or a cartesian mesh, if the load per core is larger than 2000 DOF_c.

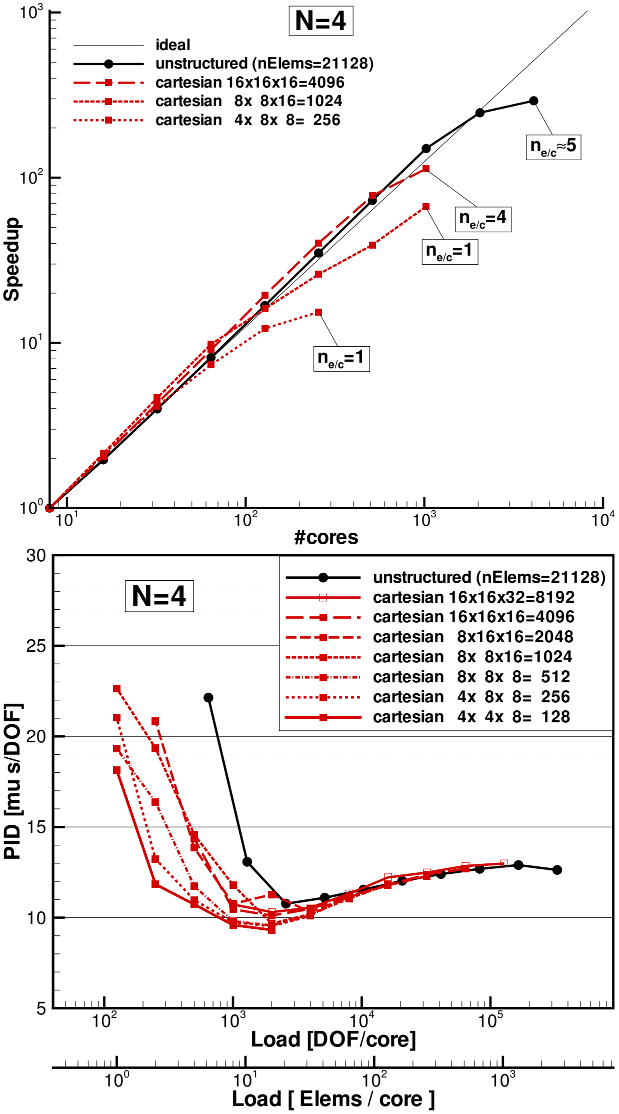


Figure 5.21.: Comparison of the strong scaling of the unstructured mesh for the sphere to the cartesian meshes for $N = 4$, speedup (top) and PID (bottom).

6. Conclusion and Prospects

6.1. Conclusions

This work provides a high order DG based framework for the simulation of time-dependent conservation laws on unstructured three-dimensional meshes for complex geometries. The polynomial approximation gives the DG scheme a sub-cell resolution and so meshes are typically coarser compared to standard Finite Volume or Finite Element meshes. The sub-cell resolution capability however leads to a very distinct requirement: The need for curved boundary faces of elements to accurately represent curved boundary conditions. Curved boundary faces are also need for other high order schemes like high order Finite Element Methods and Spectral Element Methods. Even though numerical implementations of high order schemes are very mature, they often suffer from the lack of adequate high order meshes if simulations have to be extended from simple geometries towards complex CAD models.

In the first part of this work, the DG scheme for general elements with non-linear element mappings was derived. We focused on details about the implementation, since special care has to be taken to guarantee an accurate treatment of the non-linear element mappings. One important step is the transformation of the equations to reference space using the non-linear metrics for each element separately, but also the choice of interpolation nodes and numerical integration play an important role.

Besides the classical nodal DG scheme for hybrid meshes, we describe a special implementation for hexahedral elements following ideas from the spectral element community, namely the DG-SEM. The main idea is to use an internal tensor-product Gauss grid for both the integration and the representation of the local DG solution. This sub-cell grid yields a dimension-by-dimension operator and greatly reduces the number of operations compared to the nodal DG scheme, leading to a highly efficient implementation. In addition, the scheme accurately incorporates the non-linear three-dimensional metrics [67].

In the second part, we presented the framework that was developed to generate high order meshes. Due to the complexity of the mesh generation process, we rely on existing (commercial) grid generators, which produce volume meshes with straight-edged elements. The high order preprocessor *HOPR* developed in this work fills the gap between the linear mesh and the high order simulation. Different strategies to create additional information for the surface curving were introduced. The first strategy extracts normal vectors at the surface grid points, either from reconstruction, analytical description or from the actual CAD model. In particular, sharp edges at the borders of CAD surfaces are automatically found by a CAD interface. With the normal vector strategy, the initial linear volume grid can be generated by nearly every mesh generator. The second strategy is an interpolation approach, which is tailored to specific grid generators, using ANSA© for sub-division of the surface mesh or ICEM© for high order edge data. We show the applicability of the surface curving on an unstructured hybrid mesh for a complex full aircraft.

We also compare the accuracy of the different surface curving strategies for the approximation of a circle and an airfoil. The interpolation approach was shown to lead higher accuracy, since higher order polynomials are constructed from an increased number of interpolation points, whereas the point-normal approach is restricted to cubic polynomials.

The influence of mesh generation errors can be a main error source, depending on the quality of the provided data. The errors introduced by tolerances of the CAD and mesh generation as well as the precision of the export format can easily dominate the error of the surface approximation. In some situations, like boundary layer meshes, the curving of only boundary element faces would result in inverted element mappings. Surface curving has to be propagated into the mesh domain to cure the problem. We confirm the findings of Persson [84] that inverted elements can be eliminated by a mesh deformation approach that solves a high order discretization of an elliptic equation and the linear mesh is deformed by the curved boundary.

Furthermore, we investigated the influence of high order geometry approximation for a simple test case of a coaxial wave transport using a very coarse mesh, which revealed a huge gain in accuracy when using curved elements. The tests also showed that the geometry error vanishes when increasing the polynomial degree of the mapping for a very coarse mesh. This means that depending on a given grid, only a certain minimal polynomial degree is required to resolve the geometry.

In the last part of this work, we discuss the parallelization concept for a DG scheme, for which only direct neighbor surface data needs to be communicated, and the use of non-blocking MPI commands allows us to hide the communication latency with dense local operations. We introduced a domain decomposition based on space-filling-curves and compared it to graph partitioning.

Graph partitioning outperforms the space-filling curve for a small number of domains, but the higher the domain granularity, the smaller the difference between both approaches. Therefore, we favor the space-filling curve for large scale computations, since it is only computed once and can be used to decompose the domain into an arbitrary number of sub-domains. It also greatly simplifies data structures and parallel IO.

We conducted a parallel performance analysis for the DG-SEM code *Flexi* and showed that nearly perfect strong scaling on a very high number of cores can be achieved. We showed that the limit to keep optimal performance is a very small load per core of 2000 degrees of freedom. This reflects a successful implementation of the parallelization concept. The strong scaling tests were conducted for cartesian meshes and an unstructured three-dimensional mesh. The scaling for both meshes shows the same behavior, in spite of the occurring load imbalances for the unstructured mesh.

We also want to emphasize that the overall simulation runtime with DG-SEM is nearly independent of the polynomial degree for the same resolution, contradicting the statement that high order is more expensive. To our knowledge, the computational costs for other DG schemes formulated for general elements increase rapidly with the polynomial degree. Regarding overall efficiency, a thorough comparison of a direct simulation of a three-dimensional turbulent shear layer [43] revealed that the DG-SEM performs as fast as a state of the art structured compact finite difference scheme. The results of the direct simulations of the sphere demonstrate the great potential of such a scheme for unsteady simulations, especially towards turbulent computations with complex unstructured meshes.

6.2. Prospects

The generation of curved high order meshes remains a cumbersome task, and further development towards more flexible and robust techniques need to be carried out.

The high order preprocessor *HOPR* was designed to support hybrid curved meshes consisting of tetrahedra, prisms, pyramids and hexahedra. The grid generation for complex geometries relies on the flexibility of hybrid meshes, especially automatic meshing algorithms that use tetrahedra and prisms for boundary layers. Some algorithms even generate hexa-dominant meshes [78], where hybrid meshes are only used for the transition regions. In comparison to hexahedral body-fitted meshes, the generation of valid high order tetrahedra and pyramids is more difficult since corner angles are much smaller. Here, the mesh deformation approach needs to be further investigated, especially regarding the best functionals to produce valid meshes and optimize them, like proposed by Tourlorge et al. [87].

For boundary layer meshes, high order element mappings can be used to increase the resolution of the grid cells at the wall. In [61], we investigated the stretched high order boundary layer grids, leading to an anisotropic element mapping with a higher resolution towards the wall. The approximation errors of the boundary layer profile were reduced by an order of magnitude for a one-dimensional linear advection-diffusion problem. The analysis could be extended to multiple dimensions and Navier-Stokes equations, and could greatly reduce the element count for boundary layer meshes.

A very efficient DG variant is the DG-SEM, which is restricted to hexahedra. Other element types are either less accurate or more expensive. However, fully unstructured hexa-only meshes are more difficult to generate. Automatic meshing in three-dimensions is a current topic of research, for example from the Sandia Laboratories with the CUBIT software [17, 79] and a work by Ruiz-Girones et al. [89, 90]. Semi-structured and block-structured hexahedral meshes are more common, but cannot be automatically generated and have geometric limitations. An interesting possibility to increase the flexibility of hexahedral meshing is to allow non-conforming element interfaces, provided by the commercial mesh generator HEXPRESS [72, 73]. Non-conforming element interfaces can be treated easily in DG using so-called mortar methods, already applied for DG-SEM by Kopriva et al. [70]. It would be possible to use very coarse curved hexahedral meshes, which represent curved geometries accurately, and combine this with h-adaptivity and non-conforming interfaces to adapt the mesh to the resolution required. A very promising open-source parallel adaptation library is called p4est, developed by Burstedde et al. in [25], that shows good strong scaling of grid adaptation and redistribution operations. P4est could be linked to the DG-SEM code *Flexi* to provide a very flexible and powerful high order framework for complex configurations.

A. Orthogonal Basis Functions on Triangles, Tetrahedra, Prisms and Pyramids

The description of the polynomial basis functions in this chapter are a compilation of the information issued by the book of Karniadakis and Sherwin [64].

A.1. Jacobi Polynomials

Jacobi polynomials $\mathcal{P}_n^{(\alpha,\beta)}(x)$ are a family of polynomials being orthogonal in the interval $[-1,1]$ with respect to the weighting function $(1-x)^\alpha(1+x)^\beta \quad \alpha, \beta > -1$. They can be computed recursively. Special cases are the Legendre polynomial $(\alpha,\beta) = (0,0)$ or the Chebychev polynomial $(\alpha,\beta) = (-0.5,-0.5)$. The Jacobi polynomials have the orthogonality property

$$\int\limits_{-1}^{1} \mathcal{P}_i^{(\alpha,\beta)}(x)\mathcal{P}_j^{(\alpha,\beta)}(x)(1-x)^\alpha(1+x)^\beta \, dx = \delta_{ij} \, . \tag{A.1}$$

Here, the Jacobi polynomials are already *orthonormal*. In the book of Hesthaven and Warburton [58], we find the following important relations. The differentiation property

$$\frac{d}{dx}\mathcal{P}_n^{(\alpha,\beta)}(x) = \sqrt{n(n+\alpha+\beta+1)}\mathcal{P}_{n-1}^{(\alpha+1,\beta+1)}(x) \tag{A.2}$$

the symmetry property (only if useful if $\alpha = \beta$)

$$\mathcal{P}_n^{(\alpha,\beta)}(-x) = (-1)^n \mathcal{P}_n^{(\beta,\alpha)}(x) \tag{A.3}$$

and the recurrence relation

$$a_{n+1}\mathcal{P}_{n+1}^{(\alpha,\beta)}(x) = (x-b_n)\mathcal{P}_n^{(\alpha,\beta)}(x) - a_n\mathcal{P}_{n-1}^{(\alpha,\beta)}(x) \tag{A.4}$$

where the coefficients are

$$\begin{aligned} a_n &= \frac{2}{2n+\alpha+\beta}\sqrt{\frac{n(n+\alpha+\beta)(n+\alpha)(n+\beta)}{(2n+\alpha+\beta-1)(2n+\alpha+\beta+1)}} \\ b_n &= \frac{\alpha^2-\beta^2}{(2n+\alpha+\beta)(2n+\alpha+\beta+2)}\,. \end{aligned} \tag{A.5}$$

To start the recurrence, we need the first two Jacobi polynomials

$$\mathcal{P}_0^{(\alpha,\beta)}(x) = \sqrt{\frac{\Gamma(\alpha+\beta+2)}{2^{\alpha+\beta+1}\Gamma(\alpha+1)\Gamma(\beta+1)}} \tag{A.6}$$

$$\mathcal{P}_1^{(\alpha,\beta)}(x) = \frac{1}{2}\mathcal{P}_0^{(\alpha,\beta)}(x)\sqrt{\frac{\alpha+\beta+3}{(\alpha+1)(\beta+1)}}\left((\alpha+\beta+2)x+(\alpha-\beta)\right)\,, \tag{A.7}$$

where $\Gamma(x)$ is the classical Gamma function.
For positive integer $\Gamma(n) = (n-1)!\,.$

A.2. Orthonormal Basis of a Triangle

The unit triangle is defined in reference coordinates $\boldsymbol{\xi} = (\xi, \eta)$ by

$$\mathrm{Tri} = \{(\xi,\eta)|\xi,\eta \geq -1; \xi+\eta \leq 0\} \tag{A.8}$$

with the orthonormal basis of degree N being

$$\begin{gathered} \psi_{ij}(\boldsymbol{\xi}) = \sqrt{2}\mathcal{P}_i^{(0,0)}(a)\mathcal{P}_j^{(2i+1,0)}(b)(1-b)^i \\ \forall (i,j) \geq 0;\; i+j \leq N\,. \end{gathered} \tag{A.9}$$

The number of basis functions is

$$M = \frac{1}{2}(N+1)(N+2)\,. \tag{A.10}$$

The extended coordinates $(a,b) \in [-1,1]^2$ relates to $(\xi,\eta) \in \mathrm{T}$ as

$$a = \begin{cases} 2\frac{1+\xi}{1-\eta} - 1 & 1-s \neq 0 \\ -1 & 1-\eta = 0 \end{cases}\,, \quad b = \eta\,. \tag{A.11}$$

and the inverse mapping

$$\xi = \frac{1}{2}(1+a)(1-b) - 1\,, \quad \eta = b \tag{A.12}$$

Differentiation of the Triangle Basis

To define differentiation matrices for the DG scheme, we need differentiations of the basis with respect to (ξ, η) coordinates

$$\begin{pmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{pmatrix} \psi_{ij}(\boldsymbol{\xi}) = \begin{pmatrix} \frac{\partial a}{\partial \xi} & \frac{\partial b}{\partial \xi} \\ \frac{\partial a}{\partial \eta} & \frac{\partial b}{\partial \eta} \end{pmatrix} \begin{pmatrix} \frac{\partial \psi}{\partial a} \\ \frac{\partial \psi}{\partial b} \end{pmatrix} = \begin{pmatrix} \frac{\partial \xi}{\partial a} & \frac{\partial \eta}{\partial a} \\ \frac{\partial \xi}{\partial b} & \frac{\partial \eta}{\partial b} \end{pmatrix}^{-1} \begin{pmatrix} \frac{\partial \psi}{\partial a} \\ \frac{\partial \psi}{\partial b} \end{pmatrix} \tag{A.13}$$

with the Jacobi matrix of the mapping

$$J_{\xi a} = \begin{pmatrix} \frac{1}{2}(1-b) & 0 \\ -\frac{1}{2}(1+a) & 1 \end{pmatrix}, \tag{A.14}$$

the Jacobian $det(J_{\xi a}) = \frac{1}{2}(1-b)$ and the inverse

$$J_{\xi a}^{-1} = \frac{1}{(1-b)} \begin{pmatrix} 1 & 0 \\ (1+a) & (1-b) \end{pmatrix} \tag{A.15}$$

The differentiation of the basis with respect to (a, b) leads to

$$\begin{aligned} \frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial a} &= \sqrt{2} \left(\frac{\partial \mathcal{P}_i^{(0,0)}(a)}{\partial a} \right) \mathcal{P}_j^{(2i+1,0)}(b)(1-b)^i \\ \frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial b} &= \sqrt{2} \mathcal{P}_i^{(0,0)}(a) \\ & \left[\left(\frac{\partial \mathcal{P}_j^{(2i+1,0)}(b)}{\partial b} \right) (1-b)^i - i(1-b)^{i-1} \mathcal{P}_j^{(2i+1,0)}(b) \right] \end{aligned} \tag{A.16}$$

Special attention has to be made for the case

- $i = 0, j > 0$:
  $\Rightarrow \mathcal{P}_0^{(\alpha,\beta)}(x) = \text{const.} \Rightarrow \frac{\partial \mathcal{P}_0^{(\alpha,\beta)}(x)}{\partial x} = 0$

$$\begin{aligned} \left. \frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial a} \right|_{i=0} &= 0 \\ \left. \frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial b} \right|_{i=0} &= \sqrt{2} \mathcal{P}_i^{(0,0)}(a) \left(\frac{\partial \mathcal{P}_j^{(2i+1,0)}(b)}{\partial b} \right) \end{aligned} \tag{A.17}$$

Finally, we have to assemble the derivatives. The terms in the curly brace can be formulated without fraction, thus we can evaluate the derivatives at the singularity.
We start with the first coordinate direction $\frac{\partial}{\partial \xi}$.

$$\begin{aligned}\frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial \xi} &= \frac{\partial a}{\partial \xi}\frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial a} + \frac{\partial b}{\partial \xi}\frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial b} \\ &= \left\{ \frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial a}(1-b)^{-1} \right\} ,\end{aligned} \tag{A.18}$$

then the second coordinate direction $\frac{\partial}{\partial \eta}$

$$\begin{aligned}\frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial \eta} &= \frac{\partial a}{\partial \eta}\frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial a} + \frac{\partial b}{\partial \eta}\frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial b} \\ &= (1+a)\left\{ \frac{\partial \psi_{ij}(\boldsymbol{\xi})}{\partial a}(1-b)^{-1} \right\} \\ &\quad + \left\{ \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b} \right\} ,\end{aligned} \tag{A.19}$$

A.3. Orthonormal Basis of a Tetrahedron

The unit tetrahedra is defined in reference coordinates $\boldsymbol{\xi} = (\xi, \eta, \zeta)$ by

$$\text{Tet} = \{(\xi, \eta, \zeta) | \xi, \eta, \zeta \geq -1; \xi + \eta + \zeta \leq 0\} \tag{A.20}$$

with the orthonormal basis of degree N being

$$\begin{gathered}\psi_{ijk}(\boldsymbol{\xi}) = 2\sqrt{2}\mathcal{P}_i^{(0,0)}(a)\mathcal{P}_j^{(2i+1,0)}(b)\mathcal{P}_k^{(2(i+j)+2,0)}(c)(1-b)^i(1-c)^{i+j} \\ \forall (i,j,k) \geq 0;\ i+j+k \leq N .\end{gathered} \tag{A.21}$$

The number of basis functions is

$$M = \frac{1}{6}(N+1)(N+2)(N+3) . \tag{A.22}$$

The extended coordinates $(a, b, c) \in [-1, 1]^3$ relates to $(\xi, \eta, \zeta) \in \mathrm{T}$ as

$$a = \begin{cases} 2\frac{1+\xi}{\eta+\zeta} - 1 & \eta + \zeta \neq 0 \\ -1 & \eta + \zeta = 0 \end{cases} , \quad b = \begin{cases} 2\frac{1+\eta}{1-\zeta} - 1 & t \neq 1 \\ -1 & t = 1 \end{cases} , \quad c = \zeta . \tag{A.23}$$

and the inverse mapping

$$\xi = \frac{1}{4}(1+a)(1-b)(1-c) - 1 , \quad \eta = \frac{1}{2}(1+b)(1-c) - 1 , \quad \zeta = c \tag{A.24}$$

Differentiation of the Tetrahedron Basis

To define differentiation matrices for the DG scheme, we need differentiations of the basis with respect to (ξ, η, ζ) coordinates

$$\begin{pmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \\ \frac{\partial}{\partial \zeta} \end{pmatrix} \psi_{ijk}(\boldsymbol{\xi}) = \begin{pmatrix} \frac{\partial a}{\partial \xi} & \frac{\partial b}{\partial \xi} & \frac{\partial c}{\partial \xi} \\ \frac{\partial a}{\partial \eta} & \frac{\partial b}{\partial \eta} & \frac{\partial c}{\partial \eta} \\ \frac{\partial a}{\partial \zeta} & \frac{\partial b}{\partial \zeta} & \frac{\partial c}{\partial \zeta} \end{pmatrix} \begin{pmatrix} \frac{\partial \psi}{\partial a} \\ \frac{\partial \psi}{\partial b} \\ \frac{\partial \psi}{\partial c} \end{pmatrix} = \begin{pmatrix} \frac{\partial \xi}{\partial a} & \frac{\partial \eta}{\partial a} & \frac{\partial \zeta}{\partial a} \\ \frac{\partial \xi}{\partial b} & \frac{\partial \eta}{\partial b} & \frac{\partial \zeta}{\partial b} \\ \frac{\partial \xi}{\partial c} & \frac{\partial \eta}{\partial c} & \frac{\partial \zeta}{\partial c} \end{pmatrix}^{-1} \begin{pmatrix} \frac{\partial \psi}{\partial a} \\ \frac{\partial \psi}{\partial b} \\ \frac{\partial \psi}{\partial c} \end{pmatrix} \tag{A.25}$$

with the Jacobi matrix of the mapping

$$J_{\xi a} = \begin{pmatrix} \frac{1}{4}(1-b)(1-c) & 0 & 0 \\ -\frac{1}{4}(1+a)(1-c) & \frac{1}{2}(1-c) & 0 \\ -\frac{1}{4}(1+a)(1-b) & -\frac{1}{2}(1+b) & 1 \end{pmatrix}, \tag{A.26}$$

the Jacobian $det(J_{\xi a}) = \frac{1}{8}(1-b)(1-c)^2$ and the inverse

$$J_{\xi a}^{-1} = \frac{1}{(1-b)(1-c)} \begin{pmatrix} 4 & 0 & 0 \\ 2(1+a) & 2(1-b) & 0 \\ 2(1+a) & (1-b)(1+b) & (1-b)(1-c) \end{pmatrix} \tag{A.27}$$

The differentiation of the basis with respect to (a, b, c) leads to

$$\begin{aligned} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} &= 2\sqrt{2} \left(\frac{\partial \mathcal{P}_i^{(0,0)}(a)}{\partial a} \right) \mathcal{P}_j^{(2i+1,0)}(b) \mathcal{P}_k^{(2(i+j)+2,0)}(c)(1-b)^i (1-c)^{i+j} \\ \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b} &= 2\sqrt{2} \mathcal{P}_i^{(0,0)}(a) \mathcal{P}_k^{(2(i+j)+2,0)}(c)(1-c)^{i+j} \\ &\quad \left[\left(\frac{\partial \mathcal{P}_j^{(2i+1,0)}(b)}{\partial b} \right) (1-b)^i - i(1-b)^{i-1} \mathcal{P}_j^{(2i+1,0)}(b) \right] \\ \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c} &= 2\sqrt{2} \mathcal{P}_i^{(0,0)}(a) \mathcal{P}_j^{(2i+1,0)}(b)(1-b)^i \\ &\quad \left[\left(\frac{\partial \mathcal{P}_k^{(2(i+j)+2,0)}(c)}{\partial c} \right) (1-c)^{i+j} \right. \\ &\quad \left. - (i+j)(1-c)^{i+j-1} \mathcal{P}_k^{(2(i+j)+2,0)}(c) \right] \end{aligned} \tag{A.28}$$

Special attention has to be made for the cases

- $i = 0, j > 0$:
  $\Rightarrow \mathcal{P}_0^{(\alpha,\beta)}(x) = \text{const.} \Rightarrow \frac{\partial \mathcal{P}_0^{(\alpha,\beta)}(x)}{\partial x} = 0$

$$\begin{aligned}
\left.\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a}\right|_{i=0} &= 0 \\
\left.\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b}\right|_{i=0} &= 2\sqrt{2}\mathcal{P}_i^{(0,0)}(a)\mathcal{P}_k^{(2(i+j)+2,0)}(c)(1-c)^{i+j}\left(\frac{\partial \mathcal{P}_j^{(2i+1,0)}(b)}{\partial b}\right) \\
\left.\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c}\right|_{i=0} &= 2\sqrt{2}\mathcal{P}_i^{(0,0)}(a)\mathcal{P}_j^{(2i+1,0)}(b)(1-b)^{i} \\
&\quad \left[\left(\frac{\partial \mathcal{P}_k^{(2(i+j)+2,0)}(c)}{\partial c}\right)(1-c)^{i+j}\right. \\
&\quad \left. -(i+j)(1-c)^{i+j-1}\mathcal{P}_k^{(2(i+j)+2,0)}(c)\right]
\end{aligned} \tag{A.29}$$

- $i = 0, j = 0$:

$$\begin{aligned}
\left.\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a}\right|_{i,j=0} &= 0 \\
\left.\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b}\right|_{i,j=0} &= 0 \\
\left.\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c}\right|_{i,j=0} &= 2\sqrt{2}\mathcal{P}_i^{(0,0)}(a)\mathcal{P}_j^{(2i+1,0)}(b)\left(\frac{\partial \mathcal{P}_k^{(2(i+j)+2,0)}(c)}{\partial c}\right)
\end{aligned} \tag{A.30}$$

Finally, we have to assemble the derivatives. The terms in the curly brace can be formulated without fraction, thus we can evaluate the derivatives at the singularity.
We start with the first coordinate direction $\frac{\partial}{\partial \xi}$.

$$\begin{aligned}
\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial \xi} &= \frac{\partial a}{\partial \xi}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} \\
&= 4\left\{\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a}(1-b)^{-1}(1-c)^{-1}\right\},
\end{aligned} \tag{A.31}$$

then the second coordinate direction $\frac{\partial}{\partial \eta}$

$$\begin{aligned} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial \eta} &= \frac{\partial a}{\partial \eta}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} + \frac{\partial b}{\partial \eta}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b} \\ &= 2(1+a)\left\{\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a}(1-b)^{-1}(1-c)^{-1}\right\} \\ &\quad + 2\left\{\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b}(1-c)^{-1}\right\}, \end{aligned} \tag{A.32}$$

and the third coordinate direction $\frac{\partial}{\partial \zeta}$

$$\begin{aligned} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial \zeta} &= \frac{\partial a}{\partial \zeta}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} + \frac{\partial b}{\partial \zeta}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b} + \frac{\partial c}{\partial \zeta}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c} \\ &= 2(1+a)\left\{\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a}(1-b)^{-1}(1-c)^{-1}\right\} \\ &\quad + (1+b)\left\{\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b}(1-c)^{-1}\right\} + \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c}, \end{aligned} \tag{A.33}$$

A.4. Orthonormal Basis of a Prism

The unit prism is defined in reference coordinates $\boldsymbol{\xi} = (\xi, \eta, \zeta)$ by

$$\mathrm{Pri} = \{(\xi,\eta,\zeta) | \xi,\eta,\zeta \geq -1; \xi+\eta \leq 0; \zeta \leq 1\} \tag{A.34}$$

with the orthonormal basis of degree N being

$$\begin{gathered} \psi_{ijk}(\boldsymbol{\xi}) = \sqrt{2}\mathcal{P}_i^{(0,0)}(a)\mathcal{P}_j^{(2i+1,0)}(b)\mathcal{P}_k^{(0,0)}(c)(1-b)^i \\ \forall (i,j,k) \geq 0;\ i+j \leq N;\ k \leq N\,. \end{gathered} \tag{A.35}$$

The number of basis functions is

$$M = \frac{1}{2}(N+1)(N+2)(N+1)\,. \tag{A.36}$$

The extended coordinates $(a,b,c) \in [-1,1]^3$ relates to $(\xi,\eta,\zeta) \in \mathrm{P}$ as

$$a = \begin{cases} 2\frac{1+\xi}{1-\eta} - 1 & \eta \neq 1 \\ -1 & \eta = 1 \end{cases}, \quad b = \eta\,, \quad c = \zeta\,. \tag{A.37}$$

and the inverse mapping

$$\xi = \frac{1}{2}(1+a)(1-b) - 1\,, \quad \eta = b\,, \quad \zeta = c \tag{A.38}$$

Differentiation of the Prism Basis

The Jacobi matrix of the mapping

$$J_{\xi a} = \begin{pmatrix} \frac{\partial \xi}{\partial a} & \frac{\partial \eta}{\partial a} & \frac{\partial \zeta}{\partial a} \\ \frac{\partial \xi}{\partial b} & \frac{\partial \eta}{\partial b} & \frac{\partial \zeta}{\partial b} \\ \frac{\partial \xi}{\partial c} & \frac{\partial \eta}{\partial c} & \frac{\partial \zeta}{\partial c} \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(1-b) & 0 & 0 \\ -\frac{1}{2}(1+a) & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{A.39}$$

the Jacobian $det(J_{\xi a}) = \frac{1}{2}(1-b)$ and the inverse

$$J_{\xi a}^{-1} = \begin{pmatrix} \frac{2}{1-b} & 0 & 0 \\ \frac{1+a}{1-b} & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{A.40}$$

The differentiation of the basis with respect to (a, b, c) leads to

$$\begin{aligned} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} &= \sqrt{2} \left(\frac{\partial \mathcal{P}_i^{(0,0)}(a)}{\partial a} \right) \mathcal{P}_j^{(2i+1,0)}(b) \mathcal{P}_k^{(0,0)}(c)(1-b)^i \\ \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b} &= \sqrt{2} \mathcal{P}_i^{(0,0)}(a) \mathcal{P}_k^{(0,0)}(c) \\ &\quad \left[\left(\frac{\partial \mathcal{P}_j^{(2i+1,0)}(b)}{\partial b} \right) (1-b)^i - i(1-b)^{i-1} \mathcal{P}_j^{(2i+1,0)}(b) \right] \\ \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c} &= \sqrt{2} \mathcal{P}_i^{(0,0)}(a) \mathcal{P}_j^{(2i+1,0)}(b)(1-b)^i \left(\frac{\partial \mathcal{P}_k^{(0,0)}(c)}{\partial c} \right) \end{aligned} \tag{A.41}$$

Special attention has to be made for the case $i = 0$, yielding to

$$\begin{aligned} \left. \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} \right|_{i=0} &= 0 \\ \left. \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b} \right|_{i=0} &= \sqrt{2} \mathcal{P}_i^{(0,0)}(a) \mathcal{P}_k^{(2(i+j)+2,0)}(c) \left(\frac{\partial \mathcal{P}_j^{(2i+1,0)}(b)}{\partial b} \right) \\ \left. \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c} \right|_{i=0} &= \sqrt{2} \mathcal{P}_i^{(0,0)}(a) \mathcal{P}_j^{(2i+1,0)}(b)(1-b)^i \left(\frac{\partial \mathcal{P}_k^{(0,0)}(c)}{\partial c} \right) \end{aligned} \tag{A.42}$$

Finally, we have to assemble the derivatives. The terms in the curly brace can be formulated without fraction, thus we can evaluate the derivatives at the

singularity.
We start with the first coordinate direction $\frac{\partial}{\partial \xi}$.

$$\begin{aligned} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial \xi} &= \frac{\partial a}{\partial \xi} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} \\ &= 2 \left\{ \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} (1-b)^{-1} \right\} , \end{aligned} \tag{A.43}$$

then the second coordinate direction $\frac{\partial}{\partial \eta}$

$$\begin{aligned} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial \eta} &= \frac{\partial a}{\partial \eta} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} + \frac{\partial b}{\partial \eta} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b} \\ &= (1+a) \left\{ \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} (1-b)^{-1} \right\} + \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b} , \end{aligned} \tag{A.44}$$

and the third coordinate direction $\frac{\partial}{\partial \zeta}$

$$\begin{aligned} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial \zeta} &= \frac{\partial a}{\partial \zeta} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} + \frac{\partial b}{\partial \zeta} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b} + \frac{\partial c}{\partial \zeta} \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c} \\ &= \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c} \end{aligned} \tag{A.45}$$

A.5. Orthonormal Basis of a Pyramid

The unit pyramid is defined in reference coordinates $\boldsymbol{\xi} = (\xi, \eta, \zeta)$ by

$$\text{Pyr} = \{(\xi, \eta, \zeta) | \xi, \eta, \zeta \geq -1; \xi + \zeta \leq 0; \eta + \zeta \leq 0; \} \tag{A.46}$$

with the orthonormal basis of degree N being

$$\begin{gathered} \psi_{ijk}(\boldsymbol{\xi}) = 2 \mathcal{P}_i^{(0,0)}(a) \mathcal{P}_j^{(0,0)}(b) \mathcal{P}_k^{(2(ij)+2,0)}(c)(1-c)^{(ij)} \\ \forall (i,j,k) \geq 0;\; i+k \leq N;\; j+k \leq N \,. \end{gathered} \tag{A.47}$$

with (ij) being either $(ij) = \max(i,j)$ (by Bergot,Durufle, [12]) or $(ij) = (i+j)$ (by Karniadakis [64]).
The number of basis functions is

$$M = \frac{1}{6}(N+1)(N+2)(2N+3) \,. \tag{A.48}$$

The extended coordinates $(a,b,c) \in [-1,1]^3$ relates to $(\xi,\eta,\zeta) \in \mathrm{Py}$ as

$$a = \begin{cases} 2\frac{1+\xi}{1-\zeta} - 1 & \zeta \neq 1 \\ -1 & \zeta = 1 \end{cases}, \quad b = \begin{cases} 2\frac{1+\eta}{1-\zeta} - 1 & \zeta \neq 1 \\ -1 & \zeta = 1 \end{cases}, \quad c = \zeta\,. \tag{A.49}$$

and the inverse mapping

$$\xi = \frac{1}{2}(1+a)(1-c) - 1\,, \quad \eta = \frac{1}{2}(1+b)(1-c) - 1\,, \quad \zeta = c \tag{A.50}$$

The Jacobi Matrix of the mapping

$$J_{\xi a} = \begin{pmatrix} \frac{\partial\xi}{\partial a} & \frac{\partial\eta}{\partial a} & \frac{\partial\zeta}{\partial a} \\ \frac{\partial\xi}{\partial b} & \frac{\partial\eta}{\partial b} & \frac{\partial\zeta}{\partial b} \\ \frac{\partial\xi}{\partial c} & \frac{\partial\eta}{\partial c} & \frac{\partial\zeta}{\partial c} \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(1-c) & 0 & 0 \\ 0 & \frac{1}{2}(1-c) & 0 \\ -\frac{1}{2}(1+a) & -\frac{1}{2}(1+b) & 1 \end{pmatrix}, \tag{A.51}$$

the Jacobian $det(J_{\xi a}) = \frac{1}{4}(1-c)^2$ and the inverse

$$J_{\xi a}^{-1} = \frac{1}{(1-c)} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ (1+a) & (1+b) & (1-c) \end{pmatrix}. \tag{A.52}$$

Differentiation of the Pyramid Basis

The differentiation of the basis with respect to (a,b,c) leads to

$$\begin{aligned} \frac{\partial\psi_{ijk}(\boldsymbol{\xi})}{\partial a} &= 2\left(\frac{\partial\mathcal{P}_i^{(0,0)}(a)}{\partial a}\right)\mathcal{P}_j^{(0,0)}(b)\mathcal{P}_k^{(2(ij)+2,0)}(c)(1-c)^{(ij)} \\ \frac{\partial\psi_{ijk}(\boldsymbol{\xi})}{\partial b} &= 2\mathcal{P}_i^{(0,0)}(a)\left(\frac{\partial\mathcal{P}_j^{(0,0)}(b)}{\partial b}\right)\mathcal{P}_k^{(2(ij)+2,0)}(c)(1-c)^{(ij)} \\ \frac{\partial\psi_{ijk}(\boldsymbol{\xi})}{\partial c} &= 2\mathcal{P}_i^{(0,0)}(a)\mathcal{P}_j^{(0,0)}(b) \\ &\qquad \left[\left(\frac{\partial\mathcal{P}_k^{(2(ij)+2,0)}(c)}{\partial c}\right)(1-c)^{(ij)}\right. \\ &\qquad\quad \left. -(ij)(1-c)^{(ij)-1}\mathcal{P}_k^{(2(ij)+2,0)}(c)\right] \end{aligned} \tag{A.53}$$

Special attention has to be made for the case $i, j = 0$

$$\begin{aligned}\left.\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a}\right|_{i,j=0} &= 0\\ \left.\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b}\right|_{i,j=0} &= 0\\ \left.\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c}\right|_{i,j=0} &= 2\mathcal{P}_i^{(0,0)}(a)\mathcal{P}_j^{(0,0)}(b)\left(\frac{\partial \mathcal{P}_k^{(2(ij)+2,0)}(c)}{\partial c}\right)\end{aligned} \tag{A.54}$$

Finally, we have to assemble the derivatives. The terms in the curly brace can be formulated without fraction, thus we can evaluate the derivatives at the singularity.
We start with the first coordinate direction $\frac{\partial}{\partial \xi}$.

$$\begin{aligned}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial \xi} &= \frac{\partial a}{\partial \xi}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a}\\ &= 2\left\{\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a}(1-c)^{-1}\right\},\end{aligned} \tag{A.55}$$

then the second coordinate direction $\frac{\partial}{\partial \eta}$

$$\begin{aligned}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial \eta} &= \frac{\partial b}{\partial \eta}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b}\\ &= 2\left\{\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b}(1-c)^{-1}\right\},\end{aligned} \tag{A.56}$$

and the third coordinate direction $\frac{\partial}{\partial \zeta}$

$$\begin{aligned}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial \zeta} &= \frac{\partial a}{\partial \zeta}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a} + \frac{\partial b}{\partial \zeta}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b} + \frac{\partial c}{\partial \zeta}\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c}\\ &= (1+a)\left\{\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial a}(1-c)^{-1}\right\}\\ &\quad + (1+b)\left\{\frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial b}(1-c)^{-1}\right\} + \frac{\partial \psi_{ijk}(\boldsymbol{\xi})}{\partial c}.\end{aligned} \tag{A.57}$$

B. General Gauss-Type Quadrature Rules

The integration rules for different element types make use of different one-dimensional quadrature points to account for the mapping Jacobian to a tensor-product integration space. The descriptions in this section are found in the book of Karniadakis [64]. The general one-dimensional quadrature points are defined as $(N+1)$ roots of the Jacobi polynomial of degree N, and integrate a polynomial $p(x)$ of degree $2N+1$ with the weights $(1-x)^\alpha(1+x)^\beta$ exactly

$$\int_{-1}^{1} (1-x)^\alpha (1+x)^\beta p(x)dx = \sum_{i=0}^{N} p(x_i)\omega_i \tag{B.1}$$

The multi-dimensional integrals are transformed from $\boldsymbol{\xi}$ to (a,b) or (a,b,c) via the Jacobian $det(J_{\xi a})$, given in the previous sections for each element type. A tensor-product integration has to be applied. For quadrilaterals and hexahedra, the Jacobian is 1, and Legendre-Gauss quadrature points are used in all directions. These quadrature points are the roots of the Legendre polynomial, which is the Jacobi polynomial with $(\alpha,\beta)=(0,0)$. Other element types have $det(J_{\xi a}) \neq 1$, and thus use different Jacobi polynomials in different directions. For example , the triangle has

$$det(J_{\xi a}) = \frac{1}{2}(1-b) \tag{B.2}$$

and the integral reads as

$$\int_{-1}^{1}\int_{-1}^{1} p(a(\boldsymbol{\xi}), b(\boldsymbol{\xi}))\frac{1}{2}(1-b)\, da\, db\,. \tag{B.3}$$

Here, the Gauss points and weights are defined by the Jacobi-polynomial $(\alpha,\beta)=(0,0)$ in a-direction and $(\alpha,\beta)=(1,0)$ in b-direction.
The weights for the other element types are summarized in Table B.1. The mapping $\boldsymbol{\xi}(a,b,c)$ can be used to find the Gauss point positions $\boldsymbol{\xi}_{ijk}=\boldsymbol{\xi}(a_i,b_j,c_k)$ in the reference element. Since the mapping $\boldsymbol{\xi}(a,b,c)$ is always linear in each coordinate direction, the integration precision of the Gauss quadrature remains $(2N+1)$ in the reference domain.

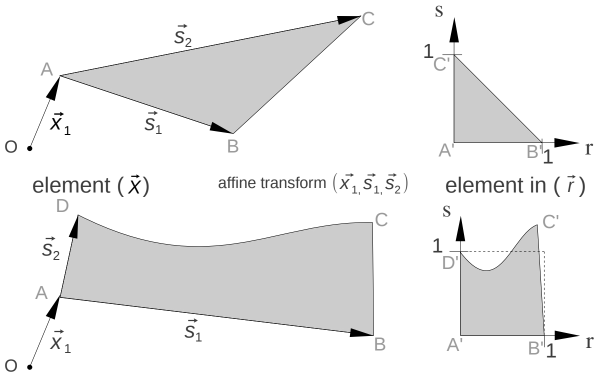
Element type	$det(J_{\xi a})$	(α, β),a	(α, β),b	(α, β),c
Triangle	$\frac{1}{2}(1-b)$	$(0,0)$	$(1,0)$	–
Tetrahedra	$\frac{1}{8}(1-b)(1-c)^2$	$(0,0)$	$(1,0)$	$(2,0)$
Prism	$\frac{1}{2}(1-b)$	$(0,0)$	$(1,0)$	$(0,0)$
Pyramid	$\frac{1}{4}(1-c)^2$	$(0,0)$	$(0,0)$	$(2,0)$

Table B.1.: Jacobi-polynomial weight exponents in (a, b, c) direction for Gauss quadrature rules for all element types.

C. Element Mappings

In this section, the properties of the element mappings for all element types are investigated. A summary is found in Tables Table C.1-Table C.3. We introduce two classes of element mappings. The element mapping is called *linear* if there exists an affine transformation between the element in physical space and the reference element, else it is called non-linear.

The vector basis for the affine mapping is spanned by 4 element corner nodes in 3D and 3 nodes in 2D, which form the basis vectors $\boldsymbol{s}_i$, see Fig. C.1. To classify the element type, the element is mapped form phyxical coordinates $\boldsymbol{x} = (x, y, z)^T$ to unit length coordinates $\boldsymbol{r} = (r, s, t)^T$.



$$\boldsymbol{x} = \boldsymbol{T}_r\,\boldsymbol{r} + \boldsymbol{x}_1$$

$$\boldsymbol{r} = \boldsymbol{T}\,(\boldsymbol{x} - \boldsymbol{x}_1)$$

$$\boldsymbol{T} = \boldsymbol{T}_r^{-1}$$

$$\boldsymbol{T}_r^{(2D)} = (\boldsymbol{s}_1, \boldsymbol{s}_2)$$

$$\boldsymbol{T}_r^{(3D)} = (\boldsymbol{s}_1, \boldsymbol{s}_2, \boldsymbol{s}_3)$$

Figure C.1.: Affine transformation to unit length

Thus, triangles and tetrahedra with straight element edges always have a linear element mapping. If the quadrilateral in 2D or the quadrilateral element faces in 3D have parallel straight edges, the affine mapping exists, too. However, in the case of straight edges only, quadrilaterals, pyramids, prisms and hexahedra have a non-linear element mapping. At curved mesh boundaries, elements are

likely to be curved. The element mapping is then a polynomial of degree N_g, normally defined by an interpolation polynomial. In the case of quadrilaterals and hexahedra, a tensor-product 1D Lagrange basis is assumed. The evaluation of Lagrange basis functions for the other element types is described in Section 3.1.
The sketch of the element shape in Tables Table C.1-Table C.3 show the element after the affine transformation to unit length.

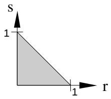
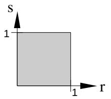
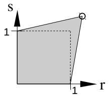
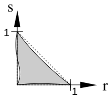
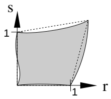
element	**description**	**maximum polynomial degree**			
shape		**mapping** $\boldsymbol{X}(\boldsymbol{\xi})$	**Jacobian** J	**metric terms** (Ja)	**surface metrics** $\boldsymbol{n}, \hat{s}$
linear element mappings					
	triangle with straight edges	$N_g = 1$	=const.	=const.	=const.
	quadrilateral with parallel straight edges	$N_g = 1$	=const.	=const.	=const.
non-linear element mappings					
	quadrilateral with non-parallel straight edges	$N_g = 1$	linear	linear	=const.
	fully curved triangle (degree N_g)	$\sim N_g$	$\sim 2N_g - 2$	$\sim 2N_g - 2$	$\sim N_g - 1$
	fully curved quadrilateral (degree N_g)	$\sim N_g$	$\sim 2N_g - 1$	$\sim 2N_g - 1$	$\sim N_g - 1$

Table C.1.: Relationship between element shape and polynomial degree of the 2D element metrics

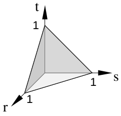
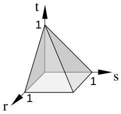
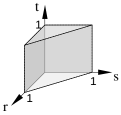
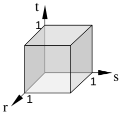
affine map of element	**description**	**maximum polynomial degree**			
		mapping $\boldsymbol{X}(\boldsymbol{\xi})$	**Jacobian** J	**metric terms** (Ja)	**surface metrics** $\boldsymbol{n}$, $\hat{s}$
	tetrahedron with straight edges	$N_g = 1$	=const.	=const.	=const.
	pyramid with straight edges, parallel at quad. face	$N_g = 1$	=const.	=const.	=const.
	prism with straight edges, parallel at quad. faces	$N_g = 1$	=const.	=const.	=const.
	hexahedron with parallel plane faces	$N_g = 1$	=const.	=const.	=const.

Table C.2.: Relationship between element shape and polynomial degree of the 3D element metrics for linear element mappings

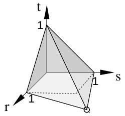
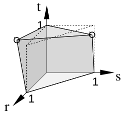
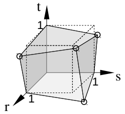
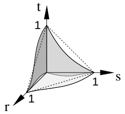
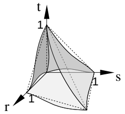
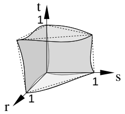
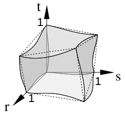
affine map of element	**description**	**maximum polynomial degree**			
		mapping $\boldsymbol{X}(\boldsymbol{\xi})$	**Jacobian** J	**metric terms** (Ja)	**surface metrics** $\boldsymbol{n}, \hat{s}$
	pyramid with straight edges	$N_g = 1$	linear	linear	=const. on tri. faces linear on quad. faces
	prism with straight edges	$N_g = 1$	quadratic	quadratic	=const. on tri. faces linear on quad. faces
	hexahedron with straight edges	$N_g = 1$	quadratic	quadratic	linear
	fully curved tetrahedron (degree N_g)	$\sim N_g$	$\sim 3N_g - 3$	$\sim 2N_g - 2$	$\sim 2N_g - 2$
	fully curved pyramid (degree N_g)	$\sim N_g$	$\sim 3N_g - 2$	$\sim 2N_g - 1$	$\sim 2N_g - 2$ on tri. faces $\sim 2N_g - 1$ on quad. faces
	fully curved prism (degree N_g)	$\sim N_g$	$\sim 3N_g - 1$	$\sim 2N_g$	$\sim 2N_g - 2$ on tri. faces $\sim 2N_g - 1$ on quad. faces
	fully curved hexahedron (degree N_g)	$\sim N_g$	$\sim 3N_g - 1$	$\sim 2N_g$	$\sim 2N_g - 1$

Table C.3.: Relationship between element shape and polynomial degree of the 3D element metrics for non-linear element mappings

C.1. Jacobian of Straight-Edge Tensor-Product Elements

In this section, we want to derive the maximum degree of the Jacobian of straight-sided *bilinear* quadrilaterals and *bilinear* and *trilinear* hexahedra.
In 2D, a *bilinear* quadrilateral has non-parallel edges, and the element mapping can be written as

$$\boldsymbol{X}(\boldsymbol{\xi}) = \boldsymbol{r}_0 + \boldsymbol{r}_1\xi + \boldsymbol{r}_2\eta + \boldsymbol{r}_3\xi\eta\,, \quad |\boldsymbol{r}_i| \neq 0\,, i = 1,\ldots,3\,. \tag{C.1}$$

The 2D Jacobian is defined in (2.27) and reads as

$$J_{2D} = a_1^1 a_2^2 - a_2^1 a_1^2\,. \tag{C.2}$$

With the derivatives of (C.1)

$$\boldsymbol{a}_1 = \frac{\partial \boldsymbol{X}}{\partial \xi} = \boldsymbol{r}_1 + \boldsymbol{r}_3\eta\,, \quad \boldsymbol{a}_2 = \frac{\partial \boldsymbol{X}}{\partial \eta} = \boldsymbol{r}_2 + \boldsymbol{r}_3\xi \tag{C.3}$$

the Jacobian computes as

$$J_2D = (r_1^1 + r_3^1\eta)(r_2^2 + r_3^2\xi) - (r_2^1 + r_3^1\xi)(r_1^2 + r_3^2\eta)\,, \tag{C.4}$$

and is *linear*, since the maximum degree in ξ and η is 1.
In 3D, a *bilinear* hexahedra has one prismatic direction, thus the element mapping reads as

$$\boldsymbol{X}(\boldsymbol{\xi}) = \boldsymbol{r}_0 + \boldsymbol{r}_1\xi + \boldsymbol{r}_2\eta + \boldsymbol{r}_3\zeta + \boldsymbol{r}_4\xi\eta\,, \quad |\boldsymbol{r}_i| \neq 0\,, i = 1,\ldots,4\,, \tag{C.5}$$

with the prismatic direction ζ. The 3D Jacobian is defined in (2.16) as

$$J = \boldsymbol{a}_1 \cdot (\boldsymbol{a}_2 \times \boldsymbol{a}_3)\,. \tag{C.6}$$

With the derivatives of (C.5)

$$\begin{aligned} \boldsymbol{a}_1 &= \boldsymbol{r}_1 + \boldsymbol{r}_4\eta\,, \\ \boldsymbol{a}_2 &= \boldsymbol{r}_2 + \boldsymbol{r}_4\xi\,, \\ \boldsymbol{a}_3 &= \boldsymbol{r}_3\,, \end{aligned} \tag{C.7}$$

the Jacobian remains *linear*.
A general *trilinear* hexahedra has at least 3 non-parallel edges, and the element mapping has the form

$$\boldsymbol{X}(\boldsymbol{\xi}) = \boldsymbol{r}_0 + \boldsymbol{r}_1\xi + \boldsymbol{r}_2\eta + \boldsymbol{r}_3\zeta + \boldsymbol{r}_4\xi\eta + \boldsymbol{r}_5\xi\zeta + \boldsymbol{r}_6\eta\zeta + \boldsymbol{r}_7\xi\eta\zeta\,, \quad |\boldsymbol{r}_i| \neq 0\,, i = 1,\ldots,7\,. \tag{C.8}$$

The derivatives of (C.8) are

$$\begin{aligned} \boldsymbol{a}_1 &= \boldsymbol{r}_1 + \boldsymbol{r}_4\eta + \boldsymbol{r}_5\zeta + \boldsymbol{r}_7\eta\zeta \,, \\ \boldsymbol{a}_2 &= \boldsymbol{r}_2 + \boldsymbol{r}_4\xi + \boldsymbol{r}_6\zeta + \boldsymbol{r}_7\xi\zeta \,, \\ \boldsymbol{a}_3 &= \boldsymbol{r}_3 + \boldsymbol{r}_5\xi + \boldsymbol{r}_6\eta + \boldsymbol{r}_7\xi\eta \,, \end{aligned} \tag{C.9}$$

and since the Jacobian is a product of all three derivatives,

$$J = \ldots \boldsymbol{r}_4 \cdot (\boldsymbol{r}_6 \times \boldsymbol{r}_7)\xi\eta^2\zeta \ldots \,, \tag{C.10}$$

it can be *quadratic*.

D. Blending Faces to Elements

Given the mappings (curved/linear) of the element sides, the volume mapping of an element, introduced in Section 2.1, can be found by a linear blending of the element sides. One chooses boolean sums of faces, then edges and corner nodes.

The hexahedral mapping is defined by

$$\boldsymbol{X}_C(\boldsymbol{\xi}) = \boldsymbol{X}_F(\boldsymbol{\xi}) - \boldsymbol{X}_E(\boldsymbol{\xi}) + \boldsymbol{X}_N(\boldsymbol{\xi}) \, , \tag{D.1}$$

with the blending of the faces

$$\begin{aligned} \boldsymbol{X}_F(\boldsymbol{\xi}) = \frac{1}{2}\{ & \boldsymbol{X}(-1,\xi^2,\xi^3)\,(1-\xi^1) + \boldsymbol{X}(+1,\xi^2,\xi^3)\,(1+\xi^1) \\ & +\boldsymbol{X}(\xi^1,-1,\xi^3)\,(1-\xi^2) + \boldsymbol{X}(\xi^1,+1,\xi^3)\,(1+\xi^2) \\ & +\boldsymbol{X}(\xi^1,\xi^2,-1)\,(1-\xi^3) + \boldsymbol{X}(\xi^1,\xi^2,+1)\,(1+\xi^3)\} \, , \end{aligned} \tag{D.2}$$

the edge blending

$$\begin{aligned} \boldsymbol{X}_E(\boldsymbol{\xi}) = \frac{1}{4}\{ & \boldsymbol{X}(-1,-1,\xi^3)\,(1-\xi^1)(1-\xi^2) + \boldsymbol{X}(-1,+1,\xi^3)\,(1-\xi^1)(1+\xi^2) \\ & +\boldsymbol{X}(+1,-1,\xi^3)\,(1+\xi^1)(1-\xi^2) + \boldsymbol{X}(+1,+1,\xi^3)\,(1+\xi^1)(1+\xi^2) \\ & +\boldsymbol{X}(-1,\xi^2,-1)\,(1-\xi^1)(1-\xi^3) + \boldsymbol{X}(-1,\xi^2,+1)\,(1-\xi^1)(1+\xi^3) \\ & +\boldsymbol{X}(+1,\xi^2,-1)\,(1+\xi^1)(1-\xi^3) + \boldsymbol{X}(+1,\xi^2,+1)\,(1+\xi^1)(1+\xi^3) \\ & +\boldsymbol{X}(\xi^1,-1,-1)\,(1-\xi^2)(1-\xi^3) + \boldsymbol{X}(\xi^1,-1,+1)\,(1-\xi^2)(1+\xi^3) \\ & +\boldsymbol{X}(\xi^1,+1,-1)\,(1+\xi^2)(1-\xi^3) + \boldsymbol{X}(\xi^1,+1,+1)\,(1+\xi^2)(1+\xi^3)\} \, , \end{aligned} \tag{D.3}$$

and the corner nodes

$$
\begin{aligned}
\boldsymbol{X}_N(\boldsymbol{\xi}) = \frac{1}{8}\{ & \boldsymbol{X}(-1,-1,-1)\,(1-\xi^1)(1-\xi^2)(1-\xi^3) \\
& +\boldsymbol{X}(-1,-1,+1)\,(1-\xi^1)(1-\xi^2)(1+\xi^3) \\
& +\boldsymbol{X}(-1,+1,-1)\,(1-\xi^1)(1+\xi^2)(1-\xi^3) \\
& +\boldsymbol{X}(-1,+1,+1)\,(1-\xi^1)(1+\xi^2)(1+\xi^3) \\
& +\boldsymbol{X}(+1,-1,-1)\,(1+\xi^1)(1-\xi^2)(1-\xi^3) \\
& +\boldsymbol{X}(+1,-1,+1)\,(1+\xi^1)(1-\xi^2)(1+\xi^3) \\
& +\boldsymbol{X}(+1,+1,-1)\,(1+\xi^1)(1+\xi^2)(1-\xi^3) \\
& +\boldsymbol{X}(+1,+1,+1)\,(1+\xi^1)(1+\xi^2)(1+\xi^3)\}\,.
\end{aligned}
\tag{D.4}
$$

The signs in the resulting blending formula (D.1) come from the boolean sum, which can be explained geometrically. If one evaluates the face blending of the hexahedron at a edge, the edge will be evaluated twice, and a corner node is found 3 times, thus not yielding the correct geometry. We can write this shortly as $\boldsymbol{X}_F[1F, 2E, 3N]$. The edge blending produces $\boldsymbol{X}_E[1E, 3N]$ and the corner blending finally $\boldsymbol{X}_N[1N]$. The boolean sum should result in $[1F, 1E, 1N]$, thus for the hexahedron

$$
\boldsymbol{X}_{\text{hex}}[1F, 1E, 1N] = \boldsymbol{X}_F[1F, 2E, 3N] - \boldsymbol{X}_E[1E, 3N] + \boldsymbol{X}_N[1N]\,. \tag{D.5}
$$

The coons triangle from Section 4.2.2 follows

$$
\boldsymbol{X}_{\text{tri}}[1E, 1N] = \frac{1}{2}\left(\boldsymbol{X}_E[2E, 3N] - \boldsymbol{X}_N[1N]\right)\,, \tag{D.6}
$$

and the coons quadrilateral

$$
\boldsymbol{X}_{\text{quad}}[1E, 1N] = \boldsymbol{X}_E[1E, 2N] - \boldsymbol{X}_N[1N]\,. \tag{D.7}
$$

The blendings of faces edges and corners for tetrahedra, pyramids and prisms are depicted in Fig. D.1. Thick lines and filled faces are curved edges and curved surfaces, which are then blended linearly to an element volume. The blending function for tetrahedra, pyramid and prism is then given by

$$
\boldsymbol{X}_{\text{tet}}[1F, 1E, 1N] = \frac{1}{3}\left(\boldsymbol{X}_F[3F, 5E, 6N] - 2\boldsymbol{X}_E[1E, 3N] + \boldsymbol{X}_N[1N]\right)\,, \tag{D.8}
$$

$$
\boldsymbol{X}_{\text{pyra}}[1F, 1E, 1N] = \boldsymbol{X}_F[1F, 2E, 3N] - \boldsymbol{X}_E[1E, 2N]\,, \tag{D.9}
$$

$$
\boldsymbol{X}_{\text{prism}}[1F, 1E, 1N] = \boldsymbol{X}_F[1F, 2E, 4N] - \boldsymbol{X}_E[1E, 2N] - \boldsymbol{X}_N[1N]\,. \tag{D.10}
$$

The formulas for the particular face and edge blendings are not given here, but follow the same logic than the triangle and quadrilateral patches. Their derivation is often easier when implementing the discrete blending function for the inner points, since regular spaced points in reference space are used to describe all mappings.

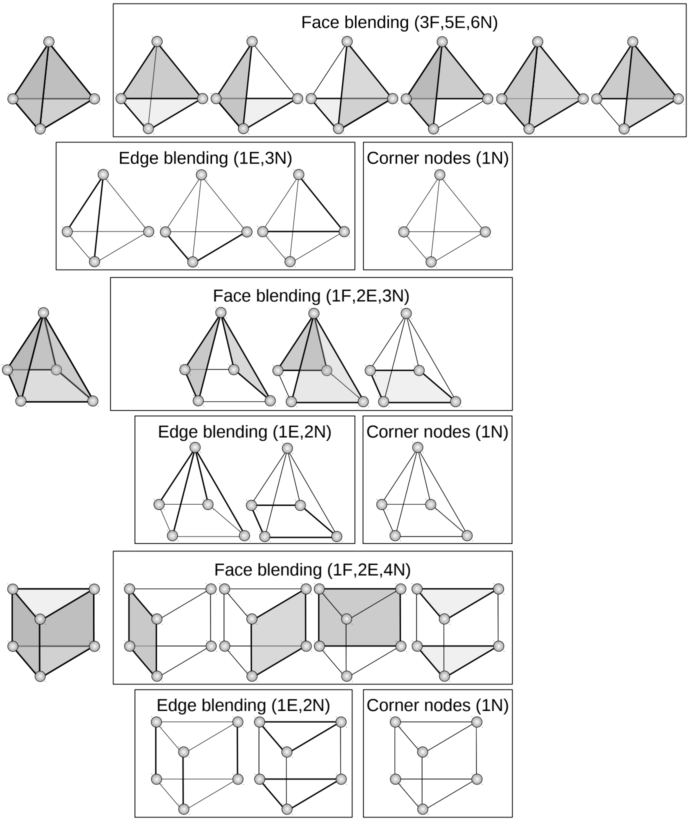


Figure D.1.: Blending of faces edges and corners for tetrahedra, pyramids and prisms.

E. Multiple Node Detection with Space-Filling Curves

The occurrence of node duplicates is very common in grid generation. A water tight mesh should only have unique nodes. In *HOPR*, this is extensively used, for example to facilitate mesh read-in by allowing multiple nodes, but also to match points for the subdivision approach Section 4.3. The task is to search the nearest points within a given distance (tolerance) in an arbitrarily distributed three-dimensional point cloud, see Fig. E.1. We adopt the idea from [36] to use space-filling curves (SFC) to describe the neighborhood of nodes for a coarse search, reducing significantly the number of nodes to check with the distance criterion. The sorting of nodes along a SFC was already introduced in Section 5.1.1.

All SFC act like an octree, since they divide the volume recursively by bisection in all dimensions. It is important that at any level of the octree, each octant covers a unique and contiguous range of SFC indices.

Hence, the algorithm to find unique nodes has the following steps:

1. set up a list of nodes which define the bounding box of the SFC
2. compute the unique SFC index (see Fig. E.2(a))
3. sort the node list
4. for each node: (see Fig. E.2(b))
    - 4.1 find the octant of the node
    - 4.2 compute the SFC index ranges of all surrounding octants
    - 4.3 compute distance function only to nodes within the given index ranges

The algorithm is very fast, and especially scale linearly with the number of nodes. A list of 335,000 nodes is checked in 3.2 seconds ($\sim 9.5\mu s$ per node), and a list of 21,090,000 nodes in 175 seconds ($\sim 8.3\mu s$ per node).

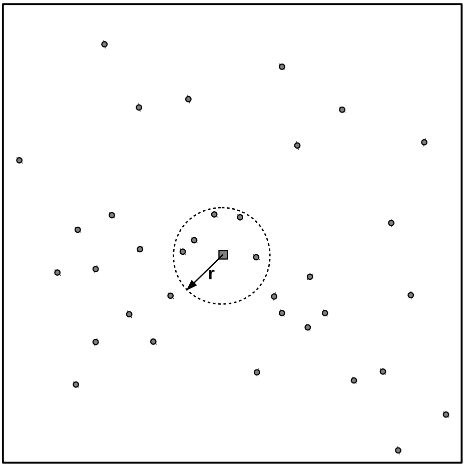
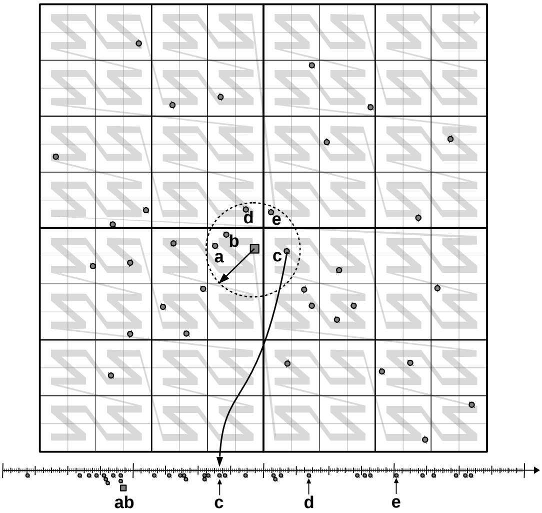
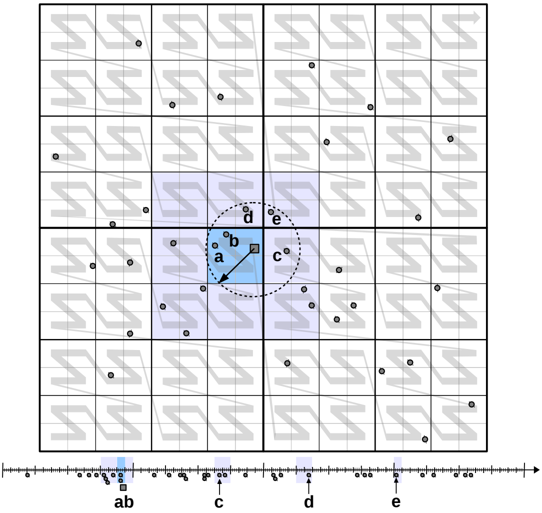


Figure E.1.: Sketch of the problem to find neighboring points within a given distance.



(a) Mapping of the point position to the position on the space filling curve.



(b) Two-dimensional search region mapped to 9 ranges (27 in 3D) on the space filling curve.

Figure E.2.: Two stages of the node matching algorithm.

Bibliography

[1] D. Ait-Ali-Yahia, M. P. Robichaud, D. Stanescu, and W. G. Habashi. Spectral Element grid generation and nonlinear computations for noise radiation from aircraft engines. In *AIAA/CEAS Aeroacoustics Conference and Exhibit, 5th, Bellevue, WA,*. AIAA, 1999.

[2] S. Aluru and F. E. Sevilgen. Parallel domain decomposition and load balancing using space-filling curves. In *in Proceedings of the 4th IEEE Conference on High Performance Computing*, pages 230–235, 1997.

[3] D. Arnold. *An Interior Penalty Finite Element Method with Discontinuous Elements.* PhD thesis, The University od Chicago, 1979.

[4] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. Discontinuous Galerkin methods for elliptic problems. In B. Cockburn, G. Karniadakis, and C.-W. Shu, editors, *Discontinuous Galerkin Methods. Lecture Notes in Computational Science and Engineering*, pages 89–101. Springer, 2000.

[5] A. Babucke. *Direct Numerical Simulation of Noise-Generation Mechanisms in the Mixing Layer of a Jet.* Dissertation, University of Stuttgart, 2009.

[6] F. Bassi and S. Rebay. A high-order accurate discontinuous Finite Element method for the numerical solution of the compressible Navier-Stokes equations. *J. Comput. Phys.*, 131:267–279, 1997.

[7] F. Bassi and S. Rebay. High-order accurate discontinuous Finite Element solution of the 2D Euler equations. *J. Comput. Phys.*, 138(2):251–285, 1997.

[8] F. Bassi and S. Rebay. A high-order discontinuous Galerkin Finite Element method solution of the 2d Euler equations. *J. Comput. Phys.*, 138:251–285, 1997.

[9] F. Bassi and S. Rebay. Numerical evaluation of two discontinuous Galerkin methods for the compressible Navier-Stokes equations. *International Journal for Numerical Methods in Fluids*, 40:197–207, 2002.

[10] F. Bassi, S. Rebay, G. Mariotti, S. Pedinotti, and M. Savini. A high-order accurate discontinuous Finite Element method for inviscid an viscous turbomachinery flows. In R. Decuypere and G. Dibelius, editors, *Proceedings of 2nd European Conference on Turbomachinery, Fluid and Thermodynamics*, pages 99–108, Technologisch Instituut, Antwerpen, Belgium, 1997.

[11] K.-J. Bathe and H. Zhang. Finite Element developments for general fluid flows with structural interactions. *International Journal for Numerical Methods in Engineering*, 60(1):213–232, 2004.

[12] M. Bergot and M. Duruflé. High-order optimal edge elements for pyramids, prisms and hexahedra. *J. Comput. Physics*, 232(1):189–213, 2013.

[13] R. Biswas, K. Devine, and J. Flaherty. Parallel, adaptive Finite Element methods for conservation laws. *Appl. Numer. Math.*, 14:255–283, 1994.

[14] T. Bjøntegaard, E. M. Rønquist, and O. Tråsdahl. High order polynomial interpolation of parameterized curves. In J. S. Hesthaven and E. M. Rønquist, editors, *Spectral and High Order Methods for Partial Differential Equations*, volume 76 of *Lecture Notes in Computational Science and Engineering*, pages 365–372. Springer Berlin Heidelberg, 2011.

[15] K. Black. A conservative Spectral Element method for the approximation of compressible fluid flow. *KYBERNETIKA*, 35(1):133–146, 1999.

[16] K. Black. Spectral Element approximation of convection-diffusion type problems. *Applied Numerical Mathematics*, 33(1-4):373–379, May 2000.

[17] T. Blacker, W. Bohnhoff, and T. Edwards. *CUBIT mesh generation environment. Volume 1: Users manual.* SAND94-1100. May 1994.

[18] M. S. Bloor. STEP-standard for the exchange of product model data. In *Standards and Practices in Electronic Data Interchange, IEE Colloquium on*, pages 2/1–2/3, May 1991.

[19] T. Bolemann. Flächenrekonstruktion mit CAD-basierten Normalenvektoren zur Erstellung von Gittern für Verfahren hoher Ordnung. Diplomarbeit, IAG, Universität Stuttgart, 2010.

[20] J. Bonet and R. D. Wood. *Nonlinear Continuum Mechanics for Finite Element Analysis.* Cambridge University Press, 1997.

[21] D. L. Bonhaus. *A higher order accurate Finite Element method for viscous compressible flows.* PhD thesis, Virginia Polytechnic Institute and State University, 1998.

[22] L. Bos, M. A. Taylor, and B. A. Wingate. Tensor product gauss-lobatto points are fekete points for the cube. *Math. Comp*, 70:1543–1547, 2001.

[23] O. Brodersen and A. Stürmer. Drag prediction of engine–airframe interference effects using unstructured Navier-Stokes calculations. In *19th AIAA Applied Aerodynamics Conference*, number AIAA 2001-2414, Anaheim, California, 11-14 June 2001. AIAA.

[24] P. Brown and Y. Saad. Hybrid Krylov methods for nonlinear systems of equations. *SIAM Journal on Scientific and Statistical Computing*, 11(3):450–481, 1990.

[25] C. Burstedde, L. C. Wilcox, and O. Ghattas. P4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comput.*, 33(3):1103–1133, May 2011.

[26] A. R. Butz. Alternative algorithm for hilbert's space-filling curve. *IEEE Trans. Comput.*, 20(4):424–426, Apr. 1971.

[27] C. Canuto, M. Hussaini, A. Quarteroni, and T. Zang. *Spectral Methods: Fundamentals in Single Domains.* Springer, 2006.

[28] N. Castel, G.Cohen, and M. Durufle. Application of discontinuous Galerkin Spectral method on hexahedral elements for aeroacoustic. *Journal of Computational Acoustics*, 17(2):175–196, 2009.

[29] B. Cockburn, S. Hou, and C.-W. Shu. The Runge-Kutta local projection discontinuous Galerkin Finite Element method for conservation laws IV: The multidimensional case. *Math. Comput.*, 54:545–581, 1990.

[30] B. Cockburn, G. E. Karniadakis, and C.-W. Shu. *Discontinuous Galerkin Methods.* Lecture Notes in Computational Science and Engineering. Springer, 2000.

[31] B. Cockburn, S. Y. Lin, and C.-W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin Finite Element method for conservation laws III: One dimensional systems. *J. Comput. Phys.*, 84:90–113, 1989.

[32] B. Cockburn and C.-W. Shu. TVB Runge-Kutta local projection discontinuous Galerkin Finite Element method for conservation laws II: General framework. *Math. Comput.*, 52:411–435, 1989.

[33] B. Cockburn and C.-W. Shu. The Runge-Kutta local projection p^1-discontinuous Galerkin method for scalar conservation laws. M^2AN, 25:337–361, 1991.

[34] B. Cockburn and C.-W. Shu. The local discontinuous Galerkin method for time-dependent convection diffusion systems. *SIAM Journal on Numerical Analysis*, 35:2440–2463, 1998.

[35] B. Cockburn and C.-W. Shu. The Runge-Kutta discontinuous Galerkin method for conservation laws V: Multidimensional systems. *J. Comput. Phys.*, 141:199–224, 1998.

[36] M. Connor and P. Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *Visualization and Computer Graphics, IEEE Transactions on*, 16(4):599–608, July 2010.

[37] A. de Boer, M. van der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers & Structures*, 85(11–14):784 – 795, 2007. Fourth MIT Conference on Computational Fluid and Solid Mechanics.

[38] C. Degand and C. Farhat. A three-dimensional torsional spring analogy method for unstructured dynamic meshes. *Computers & Structures*, 80(3–4):305 – 316, 2002.

[39] S. Fagherazzi, D. Furbish, P. Rasetarinera, and M. Y. Hussaini. Application of the discontinuous Spectral Galerkin method to groundwater flow. *Advances in Water Resourses*, 27:129–140, 2004.

[40] S. Fagherazzi, P. Rasetarinera, M. Y. Hussaini, and D. J. Furbish. Numerical solution of the dam-break problem with a discontinuous Galerkin method. *Journal of Hydraulic Engineering*, 130(6):532–539, June 2004.

[41] G. Farin. *Curves and surfaces for CAGD: a practical guide.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[42] G. Farin and D. Hansford. Discrete Coons patches. *Comput. Aided Geom. Des.*, 16(7):691–700, 1999.

[43] S. Fechter, F. Hindenlang, H. Frank, C.-D. Munz, and G. Gassner. Discontinuous Galerkin schemes for the direct numerical simulation of fluid flow and acoustics. In *Aeroacoustics Conferences*, pages –. American Institute of Aeronautics and Astronautics, June 2012.

[44] L. Fejér. Bestimmung derjenigen abszissen eines intervalles, für welche die quadratsumme der grundfunktionen der lagrangeschen interpolation im intervalle ein möglichst kleines maximum besitzt. *Annali della Scuola Normale Superiore di Pisa - Classe di Scienze*, 1(3):263–276, 1932.

[45] B. M. Froehle. *High-Order Discontinuous Galerkin Fluid-Structure Interaction Methods.* PhD thesis, University of California, Berkeley, 2014.

[46] G. Gassner. A skew-symmetric discontinuous Galerkin Spectral Element discretization and its relation to SBP-SAT Finite Difference methods. *SIAM Journal on Scientific Computing*, 35(3):A1233–A1253, 2013.

[47] G. Gassner and D. Kopriva. A comparison of the dispersion and dissipation errors of Gauss and Gauss-Lobatto discontinuous galerkin Spectral Element methods. *SIAM Journal on Scientific Computing*, 33(5):2560–2579, 2011.

[48] G. Gassner, F. Lörcher, and C.-D. Munz. A contribution to the construction of diffusion fluxes for Finite Volume and discontinuous Galerkin schemes. *Journal of Computational Physics*, 224(2):1049–1063, June 2007.

[49] G. J. Gassner, F. Lörcher, C.-D. Munz, and J. S. Hesthaven. Polymorphic nodal elements and their application in discontinuous Galerkin methods. *Journal of Computational Physics*, 228(5):1573–1590, Mar. 2009.

[50] C. Geuzaine and J.-F. Remacle. Curvilinear mesh generation for CFD. Presentation from 2009.

[51] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional Finite Element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79:1309–1331, 2009.

[52] F. Giraldo, J. Hesthaven, and T. Warburton. Nodal high-order discontinuous Galerkin methods for the spherical shallow water equations. *Journal of Computational Physics*, 181(2):499–525, september 2002.

[53] F. Giraldo and M. Restelli. A study of Spectral Element and discontinuous Galerkin methods for the Navier-Stokes equations in nonhydrostatic mesoscale atmospheric modeling: Equation sets and test cases. *J. Comp. Phys*, 227(3849-3877), 2008.

[54] M. Goldapp. Approximation of circular arcs by cubic polynomials. *Computer Aided Geometric Design*, 8(3):227 – 238, 1991.

[55] D. F. Harlacher, H. Klimach, S. Roller, C. Siebert, and F. Wolf. Dynamic load balancing for unstructured meshes on space-filling curves. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, IPDPSW '12, pages 1661–1669, Washington, DC, USA, 2012. IEEE Computer Society.

[56] R. Hartmann and P. Houston. Symmetric interior penalty DG methods for the compressible Navier–Stokes equations I: Method formulation. *Int. J. Num. Anal. Model.*, 3(1):1–20, 2006.

[57] B. T. Helenbrook. Mesh deformation using the biharmonic operator. *International Journal for Numerical Methods in Engineering*, 56(7):1007–1021, 2003.

[58] J. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods. Algorithms, Analysis, and Applications.* Springer, Berlin Heidelberg New York, 2008.

[59] D. Hilbert. Über die stetige abbildung einer linie auf ein flächenstück. In *Dritter Band: Analysis · Grundlagen der Mathematik · Physik Verschiedenes*, pages 1–2. Springer Berlin Heidelberg, 1935.

[60] F. Hindenlang, G. Gassner, C. Altmann, A. Beck, M. Staudenmaier, and C.-D. Munz. Explicit discontinuous galerkin methods for unsteady problems. *Computers & Fluids*, 61:86–93, 2012.

[61] F. Hindenlang, G. Gassner, and C.-D. Munz. Improving the accuracy of discontinuous Galerkin schemes at boundary layers. *International Journal for Numerical Methods in Fluids*, 2014.

[62] T. Hughes, J. Cottrell, and Y. Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39–41):4135–4195, Oct. 2005.

[63] Mechanical design using boundary representation, April 2002. ISO (International Organization for Standardization).

[64] G. E. Karniadakis and S. Sherwin. *Spectral/hp Element Methods for Computational Fluid Dynamics.* Numerical Mathematics and Scientific Computation. Oxford University Press, USA, 2005.

[65] G. Karypis. Metis and parmetis. In D. Padua, editor, *Encyclopedia of Parallel Computing*, pages 1117–1124. Springer US, 2011.

[66] C. Klaij, J. J. W. van der Vegt, and H. van der Ven. Spacetime discontinuous Galerkin method for the compressible NavierStokes equations. *J. Comput. Phys.*, 217(2):589–611, 2006.

[67] D. Kopriva. Metric identities and the discontinuous Spectral Element method on curvilinear meshes. *Journal of Scientific Computing*, 26(3):301–327, Mar. 2006.

[68] D. Kopriva and G. Gassner. On the quadrature and weak form choices in collocation type discontinuous Galerkin Spectral Element methods. *Journal of Scientific Computing*, 44(2):136–155, 2010-08-01.

[69] D. Kopriva, S. Woodruff, and M. Hussaini. Discontinuous Spectral Element approximation of Maxwell's equations. In B. Cockburn, G. Karniadakis, and C.-W. Shu, editors, *Proceedings of the International Symposium on Discontinuous Galerkin Methods*, New York, May 2000. Springer-Verlag.

[70] D. Kopriva, S. Woodruff, and M. Hussaini. Computation of electromagnetic scattering with a non-conforming discontinuous Spectral Element method. *International Journal for Numerical Methods in Engineering*, 53, 2002.

[71] D. A. Kopriva. *Implementing Spectral Methods for Partial Differential Equations.* Springer, 2009.

[72] K. Kovalev. *Unstructured Hexahedral Non-conformal Mesh Generation.* Phd thesis, Vrije Universiteit Brussel, December 2005.

[73] K. Kovalev, M. Delanaye, and C. Hirsch. Untangling and optimization of unstructured hexahedral meshes. In *In Proc. Workshop on Grid Generation: Theory and Applications*, pages 847–855, 2002.

[74] N. Kroll. Final report - ADIGMA (adaptive higher-order variational methods for aerodynamic applications in industry). Technical report, European Commission, CORDIS, Record number 11297, 2009.

[75] C. Ladson and C. Brooks. Development of a computer program to obtain ordinates for naca 4-digit, 4-digit modified, 5-digit, and 16-series airfoils. Technical report, NASA Technical Memorandum X-3284, 1975.

[76] F. Lörcher, G. Gassner, and C.-D. Munz. An explicit discontinuous Galerkin scheme with local time-stepping for general unsteady diffusion equations. *J. Comput. Phys.*, 227(11):5649–5670, 2008.

[77] Message Passing Interface Forum. MPI: A message-passing interface standard, version 2.2. Specification, September 2009.

[78] R. J. Meyers and T. J. Tautges. The Hex-Tet hex-dominant meshing algorithm as implemented in cubit. In *IMR*, pages 151–158, 1998.

[79] S. A. Mitchell. The all-hex geode-template for conforming a diced tetrahedral mesh to any diced hexahedral mesh. In *Proceedings of 7th international meshing roundtable*, pages 295–305, 1998.

[80] D. Moore. Fast hilbert curve generation, sorting, and range queries, May 2014.

[81] G. M. Morton. *A computer oriented geodetic data base and a new technique in file sequencing*. International Business Machines Co., Ottawa, 1966.

[82] C. D. Munz, P. Ommes, and R. Schneider. A three-dimensional finite-volume solver for the Maxwell equations with divergence cleaning on unstructured meshes. *Computer Physics Communications*, 130(1-2):83–117, July 2000.

[83] J. Peraire and P. Persson. The compact discontinuous Galerkin (CDG) method for elliptic problems. *SIAM J. Sci. Comput.*, 30(4):1806–1824, 2008.

[84] P.-O. Persson and J. Peraire. Curved mesh generation and mesh refinement using Lagrangian solid mechanics. *Proc. of the 47th AIAA Aerospace Sciences Meeting and Exhibit*, 2009.

[85] A. Ralston and P. Rabinowitz. *A First Course in Numerical Analysis.* McGraw-Hill, New York, 2nd edition, 1978.

[86] W. Reed and T. Hill. Triangular mesh methods for the neutron transport equation. Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.

[87] J.-F. Remacle, T. Toulorge, and J. Lambrechts. Robust untangling of curvilinear meshes. In X. Jiao and J.-C. Weill, editors, *Proceedings of the 21st International Meshing Roundtable*, pages 71–83. Springer Berlin Heidelberg, 2013.

[88] M. Restelli and F. Giraldo. A conservative discontinuous Galerkin semi-implicit formulation for the Navier-Stokes equations in nonhydrostatic mesoscale modeling. *SIAM J. Sci. Comp.*, 31(3):2231–2257, 2009.

[89] E. Riuz-Gironés. *Automatic hexahedral meshing algorithms: from structured to unstructured meshes.* Phd thesis, UPC, Barcelona, January 2011.

[90] E. Ruiz-Gironés, X. Roca, and J. Sarrate. The receding front method applied to hexahedral mesh generation of exterior domains. *Engineering with Computers*, 28(4):391–408, 2012.

[91] C. Rumsey, B. Wedan, T. Hauser, and M. Poinot. Recent updates to the CFD General Notation System (cgns). In *Aerospace Sciences Meetings*, pages –. American Institute of Aeronautics and Astronautics, Jan 2012.

[92] J. Stiller. Point-normal interpolation schemes reproducing spheres, cylinders and cones. *Computer Aided Geometric Design*, 24(5):286 – 301, 2007.

[93] A. Tomboulides and S. Orszag. Numerical investigation of transitional and weak turbulent flow past a sphere. *J. Fluid Mech.*, 416:45–73, 2000.

[94] L. Trefethen. Is Gauss Quadrature better than Clenshaw–Curtis? *SIAM Review*, 50(1):67–87, 2008.

[95] M. Vinokur and H. Yee. *Extension of Efficient Low Dissipation High Order Schemes for 3-D Curvilinear Moving Grids*, chapter 8, pages 129–164.

[96] H. Wang, J. Kearney, and K. Atkinson. Arc-length parameterized spline curves for real-time simulation. In *In in Proc. 5th International Conference on Curves and Surfaces*, pages 387–396, 2002.

[97] Z. Wang. High-order methods for the Euler and Navier-Stokes equations on unstructured grids. *Progress in Aerospace Sciences*, 43(1–3):1 – 41, 2007.

List of Tables

3.1. Number of interpolation points n_{IP} for six standard element types. 22

3.2. Condition number, C, of the Vandermonde matrix and Lebesque constant, Λ, for $N = 8$ for six standard element types and three choices of basis functions and nodal sets. 26

4.1. Maximum growth rate $f^{\max}$ for increasing mapping quality of the scaled Jacobian J_s, normalized by $J_s^{exact} = f^{-N_g}$ and corresponding growth rate between DG elements is f_{DG}. 66

4.2. Difference of the scaled Jacobian distributions between the final curved and the initial linear mesh, for different deformation models and three meshes $n_e = 4, 8, 16$. 77

4.3. Total number of iterations of the CG solver, $||R|| < 1.0E-04$. Non-linear system solved with 40 load increments and ≤ 3 Newton steps per increment. 78

4.4. Minimum distance error norms for NACA0012 mesh, using interpolation points from a mesh generator. The distance error $L_\infty(\Delta x)$ of the interpolation points is given as well as the error of the interpolation polynomial. 98

4.5. Interpolation error norms for the radius, normal vector and curvature, for a circular arc of 120° for polynomial degrees $N_g = 2, \ldots, 5$, taken from Section 4.6.1. 100

5.1. Comparison of the Metis and space-filling curve (SFC) mesh partitioning. 114

5.2. Estimated operation counts for DG-SEM for one hexahedron. . 120

5.3. Operation count and operation count per DOF for one hexahedra for Navier-Stokes DG-SEM operator with gradient evaluation. . 120

5.4. Number of REAL variables (8 Byte) per element for a given polynomial degree N and number of conserved variables n_{var}. . 121

5.5. Estimated and measured memory consumption per element for $n_{\text{var}} = 5$ and $N = 2 - 10$, as well as the number of elements that fit into 2MB cache. 122
5.6. Comparison of the performance of explicit DG-SEM and cFD codes for the simulation of a turbulent shear layer, from [43]. . 129
5.7. Statistics of the domain decomposition for 128, 1024 and 4096 domains, with the average, root mean square, minimum and maximum of all domains. 134

B.1. Jacobi-polynomial weight exponents in (a, b, c) direction for Gauss quadrature rules for all element types. 154

C.1. Relationship between element shape and polynomial degree of the 2D element metrics 156
C.2. Relationship between element shape and polynomial degree of the 3D element metrics for linear element mappings 157
C.3. Relationship between element shape and polynomial degree of the 3D element metrics for non-linear element mappings . . . 158

List of Figures

2.1. Mapping of a hexahedral element from physical to reference space . . . 8

3.1. Reference elements for triangles, quadrilaterals, tetrahedra, pyramids, prisms and hexahedra . . . 21
3.2. Condition number of the Vandermonde matrix (left) and Lebesgue constant (right) for six standard element types and $N = 1, \ldots, 8$. Three choices of basis functions and interpolation nodes, from top to down: monomial basis and uniformly spaced nodes, orthogonal basis and uniformly spaced nodes and orthogonal basis and optimized nodes. . . . 25
3.3. Location of Gauss points ■ and boundary fluxes □ in 2D. . . . 41

4.1. Strategies for the generation of curved element sides. . . . 47
4.2. Flowchart of the curved mesh generation process. . . . 49
4.3. Mapping from parameter space (ξ_1, ξ_2) to physical space using (4.1) . . . 50
4.4. Construction of tangential vectors (a) by projection and (a) by cross product. . . . 53
4.5. Sequence of constructing curved element edges from point-normals. 53
4.6. Special case of a circular arc, where optimal tangential vectors can be derived. . . . 54
4.7. Flowchart of the CAD tool to assign normal vectors from CAD. 55
4.8. Construction of mappings: from curved element edges to surface and volume mappings. . . . 56
4.9. Blending of curved element edges to a curved element side. . . 57
4.10. Additional CL points on curved edges for a very coarse mesh of a cylinder and a sphere surface, generated with ICEM©. . . . 58
4.11. Surface mesh of the DLR-F6 body-wing intersection, after two steps of isotropic refinement. . . . 60

4.12. Sketch of two stages of the algorithm to associate refined surface elements and coarse boundary faces, using the surface mesh connectivity, for a triangle and a quadrilateral. 61
4.13. Example of an agglomerated structured grid ($N_g = 4$). 62
4.14. Curved mesh of the periodic hill test case from an agglomerated structured grid ($N_g = 4$), 3D view (left) and front view (right) with initial grid lines shown in gray. 62
4.15. Example of a stretched DG grid derived from a finer grid (left), where the sub-grid is used for the element mapping (right). . . 64
4.16. Quality of the interpolated element mapping for polynomial degrees $N_g = 2, ..., 11$, over the fine grid growth rate f_{fine}. 65
4.17. The curved surface intersects the inner edges of a linear boundary layer mesh, and inner elements must be curved for a valid mesh. 67
4.18. Displacement vector for the Dirichlet boundary condition, being the difference between the linear and the curved boundary interpolation points. 69
4.19. Sequence of linear boundary layer meshes for the NACA0012 airfoil, curved boundary ($N_g = 4$) in red. 72
4.20. Curved boundary layer meshes (red lines) after deformation with Laplace equation and free corner nodes. Initial linear mesh in gray. 73
4.21. Curved boundary layer meshes (red lines) after deformation with Laplace equation and clamped corner nodes. Linear mesh shown in gray. 73
4.22. Comparison of the mesh deformation using the Laplace equation (black lines) and the neo-Hookean material law with increasing Poisson ratio $\nu = 0.3, 0.4, 0.45$ (red lines), with clamped corner nodes. 74
4.23. Distribution of the scaled Jacobian, before and after the mesh deformation with clamped corner nodes, using the Laplace equation (top) and the neo-Hookean material with $\nu = 0.45$ (bottom). 75
4.24. L_2 error norm of the radius for different approximations of a circular arc with angular extent $\varphi \in [1^\circ; 120^\circ]$. 79
4.25. L_2 error norm of the normal vector and curvature for different approximations of a circular arc with angular extent $\varphi \in [1^\circ; 120^\circ]$. 81

4.26. L_2 error norm of the radius with a random noise disturbance of 10^{-8}, for different approximations of a circular arc with angular extent $\varphi \in [1^\circ; 120^\circ]$. 82
4.27. Nose of a NACA0012 airfoil with 4 elements and a uniform and Chebychev-Lobatto point distribution and an arc length parametrization. 83
4.28. L_2 error norm of the distance to the NACA0012 profile, over the number of elements, for different polynomial approximations. . 86
4.29. L_2 error norm of the *minimum distance* to the NACA0012 profile, over the number of elements, for different polynomial approximations. 87
4.30. Comparison of the error of the distance to a uniformly spaced point set and the minimum distance, for Chebychev-Lobatto interpolation and even (left) and odd (right) polynomial degrees N_g. 88
4.31. Error norm of the normal vector (top) and curvature (bottom) for even (left) and odd (right) polynomial degrees. 89
4.32. Sketch of the spherical quadrilateral surface with parametric lines. 90
4.33. L_2 error norm of the radius for different approximations of the spherical quadrilateral. 91
4.34. NACA0012 mesh with 4 curved elements ($N_g = 8$) along the chord, generated from a structured base mesh (in gray). 92
4.35. Polynomial approximation for $N_g = 4$, the distance to exact geometry is magnified by factor 50. Initial mesh (top) and using projected interpolation points (bottom). 94
4.36. Polynomial approximation for $N_g = 8$, the distance to exact geometry is magnified by factor 50. Initial mesh (top) and using projected interpolation points (bottom). 95
4.37. Comparison of minimum distance error norms for NACA0012 mesh, using interpolation points from a mesh generator, the cases with interpolation point projection are marked with *. . 96
4.38. Entropy error and pressure coefficient along the front of the airfoil, $n_e = 4, N_g = 8$, for an inviscid flow of $Ma = 0.3$. 97
4.39. Geometry and initial solution of the coaxial waveguide (slice at $x = 0$, extraction line at $y = 0.2$). 99
4.40. 3D view of meshes for $N_g = 1, 2, 5$ and cross section for $N_g = 1, \dots, 5$. 100
4.41. Comparison of the electric field for $N = 5$ after 2 and 10 cycles for different geometry approximations. 101

4.42. Electric field for $N = 8$ after 10 cycles for different geometry approximations. 102
4.43. Hybrid volume mesh of the DLR-F6 configuration (tetrahedra, pyramids,prisms and hexahedra). 103
4.44. DLR-F6 configuration: scaled Jacobian statistics for curved tetrahedra and prisms. 104
4.45. Visualizations of the curved mesh. 105

5.1. Examples of space-filling curves, a 2D Hilbert curve with different refinement levels (left), a 3D Morton or Z-curve (middle) and 3D Hilbert curve (right), source: Wikipedia. 110
5.2. Four levels of the two-dimensional Morton space filling curve (z-curve) and the one-dimensional curve below. Corresponding volumes and one-dimensional ranges are highlighted. 111
5.3. Domain decomposition on 160 cores using the space-filling curve. 113
5.4. Comparison of the number of inter-domain faces (nMPI_faces) for the Hilbert and Morton curve, on 512 (left) and 1024 domains (right), plotted over all domains. 114
5.5. Comparison of the number of domain neighbors (nNeighbors) for the Hilbert and Morton curve, on 512 (left) and 1024 domains (right), plotted over all domains. 115
5.6. Communication pattern for inter-processor flux computation with non-blocking communication. 116
5.7. Flow chart of the parallel DG operator for hyperbolic equations, with **buffer** and *communication* routines. 117
5.8. Flow chart of the parallel DG operator for parabolic equations with lifting (BR1 scheme), with **buffer** and *communication* routines. 118
5.9. Load balancing of the inter-processor communication data, by assignment of master (m) and slave (s) sides. 119
5.10. Domain decomposition of a $16 \times 16 \times 8$ mesh into 32 and 64 domains, using the Hilbert space filling curve. Domains are scaled by a factor 0.5 for visualization. 123
5.11. Strong scaling for $N = 5$ and three mesh resolutions (top) and on the same mesh with an increasing polynomial degree (bottom), starting from $n_c = 8$ cores. 124
5.12. Performance index of the strong scaling simulations, over the load per core. 126

5.13. Performance index for an increasing load and a fixed number of cores, ranging from 8−1024, for polynomial degrees of $N = 3, 4, 5$ and 8. 127
5.14. Behavior of the performance index over the load per core. . . . 128
5.15. Performance index over the polynomial degree N at constant load per core. 129
5.16. Weak scaling for the limiting case of one element per core and different polynomial degrees. 130
5.17. Weak scaling for different polynomial degrees and a similar load per core, left $729 - 1024\,\mathrm{DOF}/c$ and right $2048 - 4000\,\mathrm{DOF}/c$. 131
5.18. Unstructured mesh of the sphere, 3D views, front view (left), slice and back view. 132
5.19. Domain decomposition of the unstructured mesh for the sphere (left) and communication graph between the domains (right), with 128, 1024 and 4096 domains. 133
5.20. Isosurface of $\lambda_2 = -0.001$ of the sphere flow at $\mathrm{Re} = 1000$ and velocity magnitude contours (levels $[0, 1]u_\infty$) in the x-y plane. . 135
5.21. Comparison of the strong scaling of the unstructured mesh for the sphere to the cartesian meshes for $N = 4$, speedup (top) and PID (bottom). 136

C.1. Affine transformation to unit length 155

D.1. Blending of faces edges and corners for tetrahedra, pyramids and prisms. 163

E.1. Sketch of the problem to find neighboring points within a given distance. 166
E.2. Two stages of the node matching algorithm. 167

Lebenslauf

08.02.1981	Geboren in Stuttgart
1987 - 1991	Grundschule, Leinfelden-Echterdingen
1991 - 2000	Immanuel-Kant-Gynmasium, Leinfelden
2000	Allgemeine Hochschulreife
2000 - 2001	Zivildienst in einer Behindertenwerkstatt, Garmisch
2001 - 2008	Studium der Luft- und Raumfahrttechnik an der Universität Stuttgart, Vertiefungsrichtungen: Strömungslehre, Raumfahrttechnik
2005 - 2006	Auslandsaufenthalt in Toulouse, Frankreich Teilnahme am Deutsch-Französischen Integrierten Studium Absolvierung der Strömungslehre-Vertiefung an der SUPAERO und Studienarbeit an der ONERA
2008 - 2014	Wissenschaftlicher Mitarbeiter am Institut für Aerodynamik und Gasdynamik der Universität Stuttgart

Stuttgart, den 30.09.2014 Florian Hindenlang