TECHNISCHE UNIVERSITÄT MÜNCHEN

TUM School of Engineering and Design

A modular and efficient implementation of isogeometric analysis for the interactive CAD-integrated design of lightweight structures

Thomas Josef Oberbichler

Vollständiger Abdruck der von der TUM School of Engineering and Design der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitz:

Prof. Dr. Pierluigi D'Acunto

Prüfer:innen der Dissertation:

1. Prof. Dr.-Ing. Kai-Uwe Bletzinger
2. Prof. Dr. Helmut Pottmann
3. Prof. Dr. Fehmi Cirak

Diese Dissertation wurde am 7.11.2022 bei der Technischen Universität München eingereicht und durch die TUM School of Engineering and Design am 15.3.2023 angenommen.

Schriftenreihe des Lehrstuhls für Statik
TU München

Band 56

Thomas Josef Oberbichler

A modular and efficient implementation of isogeometric analysis for the interactive CAD-integrated design of lightweight structures

München 2023

Veröffentlicht durch

Kai-Uwe Bletzinger
Lehrstuhl für Statik
Technische Universität München
Arcisstr. 21
80333 München

Telefon:	+49 (0) 89 289 22422
Telefax:	+49 (0) 89 289 22421
E-Mail:	kub@tum.de
Internet:	www.cee.ed.tum.de/st

ISBN: 978-3-943683-69-1

Abstract

With isogeometric analysis (IGA), the range of interactive CAD-integrated parametric design tools can be extended to include mechanical behaviour. A structural model can be interpreted as a parametric model which describes the energy state as a function of the displacements at discrete nodes. A stable equilibrium state is defined as a state of minimum energy. From a mechanical point of view, the derivatives correspond to the residual force and stiffness of the system. The energy functional to be minimized is usually composed of elastic deformation, prestress and external loads. Depending on the chosen geometric discretization (e.g. discrete meshes, NURBS, subdivision surfaces, etc.), this energy functional can be defined as a function of different parameters.

In this thesis, modular and efficient methods for the implementation of finite isogeometric elements were developed, which allow interactive CAD-integrated analyses of lightweight structures. The selection of different discretizations is interpreted as a reparameterization of the energy functional. Techniques from algorithmic differentiation (AD) are used to compute the required derivatives by decomposing the parametric model of the energy functional into building blocks. It is shown that the use of the backward method, especially in combination with smooth geometries, leads to a dramatic increase in performance. The approach leads to a core-congruential element formulation, which makes it possible to define a core element that is initially independent of the geometry parameterization and can subsequently be combined with other parameterizations. The concept is presented for structural analysis and shape finding in combination with trimmed NURBS geometries and Catmull-Clark subdivision surfaces.

Computational efficiency and modularity make it possible to integrate isogeometric analysis for different geometric parameterizations as interactive analysis tools which are fully integrated into a CAD package.

Zusammenfassung

Isogeometrische Analyse (IGA) erweitert den Funktionsumfang CAD-integrierter parametrischer Entwurfswerkzeuge um mechanisches Verhalten. Das mechanische Modell eines Tragwerks kann als ein parametrisches Modell interpretiert werden, welches den Energiezustand als eine Funktion von Verschiebungen an diskreten Knoten beschreibt. Ein stabiler Gleichgewichtszustand entspricht einem Zustand minimaler Energie. Mechanisch werden die Ableitungen als Residualkraft und Steifigkeit des Systems interpretiert. Das zu minimierende Energiefunktional setzt sich u.a. aus elastischer Verformung, Vorspannung und externen Lasten zusammen. Entsprechend der gewählten geometrischen Diskretisierung (z.B. diskrete Netze, NURBS, Unterteilungsflächen usw.) kann dieses Energiefunktional in Abhängigkeit von unterschiedlichen Parametern definiert werden.
Im Rahmen dieser Arbeit wurde ein modulares und effizientes Vorgehen für die Implementierung von finiten isogeometrischen Elementen entwickelt, welchea interaktive CAD-integrierte Analysen von Leichtbautragwerken ermöglicht. Die Wahl unterschiedlicher Diskretisierungen wird als Umparametrisierung des Energiefunktionals interpretiert. Für die Berechnung der benötigten Ableitungen werden Techniken des algorithmischen Ableitens (AD) adaptiert und das parametrische Modell des Energiefunktionals in Bausteine zerlegt. Die Verwendung der adjungierten Methode führt in Kombination mit glatten Geometrien zu einer drastischen Leistungssteigerung. Das Vorgehen entspricht einer ”Kern-kongruenten”Elementformulierung, bei der Kernelement unabhängig von der Geometrieparametisierung definiert werden und anschließend mit unterschiedlichen Parametrisierungen kombiniert werden können. Das Konzept wird für die Strukturanalyse und Formfindung in Kombination mit getrimmten NURBS Geometrien und Catmull-Clark Unterteilungsflächen präsentiert. Aufgrund der Recheneffizienz und Modularität ist es auf diese Weise möglich, isogeometrische Analysetools für unterschiedliche Geometrieparametrisierungen in Form von interaktiven Analysewerkzeugen in CAD-Pakete zu integrieren.

Acknowledgments

This thesis was written from 2017 to 2022 during my time as a research associate at the Chair of Structural Analysis (Lehrstuhl für Statik) at the Technische Universität München.
First, I would like to thank Prof. Kai-Uwe Bletzinger for giving me the opportunity to work in the challenging and fascinating field of numerical form-finding and optimisation. With inspiring discussions and academic freedom, he provided me with a great working environment to grow academically and personally.
I also want to thank Prof. Helmut Pottmann and Prof. Fehmi Chirak for completing my board of examiners and Prof. Pierluigi D'Acunto for the organisation. Their interest in my work is gratefully acknowledged.
Furthermore, I would like to thank all my coworkers at the Chair of Structural Analysis for their friendly cooperation and the pleasant time I had. I would like to thank Benedikt Philipp, Michael Breitenberger, and Prof. Roland Wüchner, who supported me during my studies with discussions on- and off-topic. During my research, I met many colleagues who inspired me with their fascinating work and discussions. I especially want to mention Prof. Frank Petzold, Ole Ohlbrock, and Eike Schling.
My deepest gratitude goes to my family, who have supported me through all chapters of my life.

Thomas Oberbichler
Munich
April 2023

Contributions

This cumulative dissertation is based on two published peer-reviewed research papers, which are presented in Chapter 5 and Chapter 6.

Paper 1

Oberbichler, T., Wüchner, R., Bletzinger, K.-U. (2021). Efficient computation of nonlinear isogeometric elements using the adjoint method and algorithmic differentiation. *Computer Methods in Applied Mechanics and Engineering, 381*. https://doi.org/10.1016/j.cma.2021.113817

Contributions: Thomas Oberbichler develops a modular concept for the efficient and modular implementation of isogeometric finite elements for structural analysis. Roland Wüchner and Kai-Uwe Bletzinger supervised this study and reviewed the manuscript. All authors approved the final version.

Paper 2

Oberbichler, T., Bletzinger, K. U. (2022). CAD-Integrated Form-Finding of Structural Membranes Using Extended Catmull–Clark Subdivision Surfaces. *Computer-Aided Design, 151*. https://doi.org/10.1016/j.cad.2022.103360

Contributions: Thomas Oberbichler extends the concept of Paper 1 for the application in form-finding of lightweight structures and the support of subdivision surfaces. Kai-Uwe Bletzinger supervised this study and reviewed the manuscript. All authors approved the final version.

Contents

1

Introduction

The design of lightweight structures as an interaction of form and force is an exciting as well as challenging task. Pure geometry is enriched by mechanical properties such as stiffness and prestress. To obtain a state of equilibrium, computer-aided methods are usually required. The tight interaction of form and force results in multiple design iterations. In this context, interactive tools are particularly helpful as they provide a real-time feedback while changing specific parameters. The designer creates a parametric model and specifies goals for form and force interactively while the computer resolves the complex physical relations and determines the appropriate equilibrium state. It is a productive collaboration between man and machine. Ivan Sutherland demonstrated the potential of this concept in 1963 with his work 'Sketchpad' [1]. The focus is on the application of geometric constraints as shown in Figure 1.1. Similar techniques can be found in many modern *computer-aided design* (CAD) packages which allow the description of geometric relationships implicitly via constraints or goals.

Sutherland considers the application to mechanical structures in a brief chapter of his thesis. Indeed, there is a smooth transition from geometry to mechanics. Geometric constraints, e.g. lengths and angles, can be interpreted mechanically by elastic members or rotational springs. A complex problem can be decomposed

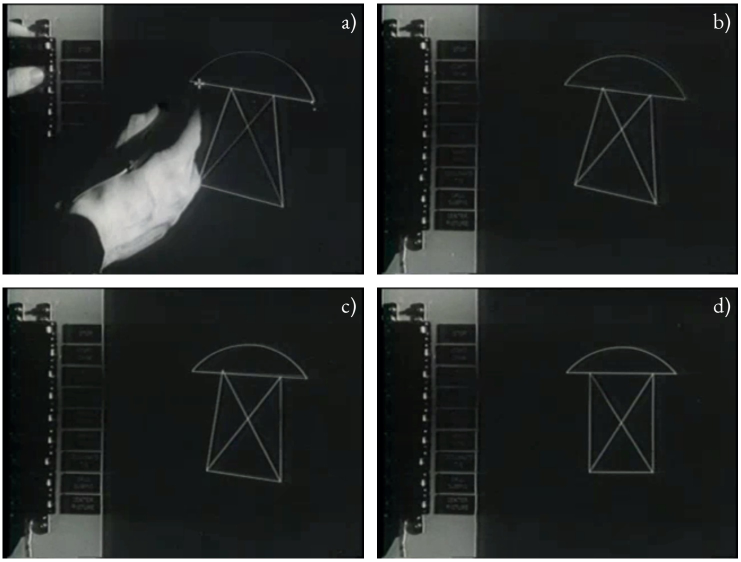


Figure 1.1: Constraint-driven design of a rivet with the interactive drawing tool *Sketchpad* by Ivan Sutherland (screenshots from [2]). The user sketches a solution where he prescribes the topology and initial geometry of the design. Then he imposes additional constraints such as angles and lengths (a). The computer tries to satisfy the implicit constraints by moving the design parameters - in this case the location of the vertices (b, c and d).

into a set of such *finite elements*. Each elastic element has the goal of minimizing its own deformation energy and contributes to the total energy of the entire system. A stable equilibrium corresponds to a state of minimal energy, which results in an optimization problem. Such problems can be solved by using gradient-based methods. From a mechanical point of view, the gradient of the energy functional corresponds to the residual force vector, and the Hessian matrix to the tangential stiffness matrix. The derivation, implementation and computation of these derivatives is usually a time-consuming task.

Isogeometric analysis (IGA) extends the range of finite elements to smooth free-form geometries. It adopts the geometric parameterization of *computer-aided design* (CAD) tools for the formulation of mechanical and geometrical constraints. Therefore, free-form geometries can be used directly for *computer-aided engineering*

(CAE) without converting the geometric parametrization. Boundaries between design and analysis are disappearing as the exchange of data between CAD and CAE is simplified. This is especially useful in the early design phases where multiple design iterations are necessary. However, CAD tools offer different possibilities to parameterize free-form geometries. Besides discrete meshes, smooth B-splines, NURBS and subdivision surfaces are also commonly used. Based on the geometric parameterizations, different mechanical elements can be formulated for different types of analysis. From the combinations, a large number of finite elements and coupling conditions can be generated. Each of them requires an individual finite element implementation.

The computation of the required derivatives usually becomes slower and requires more memory when smooth CAD geometries are involved. CAD-integrated FE tools are therefore usually divided into two components: a finite element analysis (FEA) kernel, which is optimized for numerical computations, and a plug-in for the CAD package, which has access to the data and functionalities of the CAD tool, acts as a classic pre- and post-processor and provides the graphical user interface. Additional interfaces must be designed, implemented and maintained to enable communication between these two components. As the sharing of functionalities and data structures through such interfaces is restricted, it is necessary to re-implement CAD functionalities within the FEA kernel and vice versa. This efforts makes the implementation of IGA within CAD-integrated interactive design tools difficult or even impossible.

This thesis addresses the following question:

> How can IGA be implemented in a modular and efficient way to enable the interactive CAD-integrated analysis of lightweight structures?

In this context, the term 'efficiency' includes three aspects: (i) a quick and easy implementation of new elements within an existing finite element framework, (ii) an economic use of computational resources (computational time and memory consumption) and (iii) an easy way to maintain, modify and reuse existing element formulations. A 'modular' implementation should make it possible to combine mechanical and geometrical finite elements from reusable building blocks.

For this purpose, *algorithmic differentiation* (AD) techniques are adapted and optimized for use in the field of IGA. The thesis is structured as follows:

Chapter 2 is an introduction to the concept of parametric modelling. It is shown how algorithms can be used to represent parametric relationships. The basic distinction between explicit and implicit parametrizations is clarified. The search for the parameters that produce a certain result leads to inverse problems where the solution can be obtained by numerical optimization with gradient-based methods.

Chapter 3 introduces algorithmic differentiation (AD) as a technique to systematically determine the sensitivities of a model with respect to the design parameters. The advantages over finite differences for functions with multiple parameters are elaborated. The chapter concludes with a description of the implementation of AD using hyperdual numbers.

Chapter 4 gives an overview of common parameterizations of free-form geometries and the corresponding subdivision schemes, which are visualized as graphic solutions. This should clarify the algorithmic concept behind these geometries.

Chapter 5 transfers the concept of adjoint sensitivity analysis to the formulation of isogeometric finite elements like cables, beams, membranes and shells. The procedure corresponds to the application of reverse-mode AD to the energy functional of the corresponding element. The use of different geometric parameterizations corresponds to a reparameterization of this energy functional. The element formulations are modularized by dividing them into reusable blocks. In combination with different geometric transformations the energy functional can be adapted to different geometric parameterizations.

Chapter 6 extends this technique to the area of form finding. Prestressed membrane and cable elements are used in combination with the updated reference strategy (URS) to determine the equilibrium of lightweight structures. The reference configuration is reduced to the simple weights of the energy functional. It is therefore sufficient to update the weights instead of the reference location of the nodes. This allows the combination of form-found and elastic elements in hybrid systems in a very elegant way and avoids modelling problems as descibed in [3].

Chapter 7 gives conclusions and provides an outlook on future research.

2 Parametric modelling

During the early design phase of a project, not all the details are yet known or finally defined. Changing the design parameters usually leads to a revision of individual parts or even the entire design.

Parametric modelling represents the relations between the design parameters and the resulting design in a transparent way. The relation is described by a blueprint, which defines how the solution is constructed based on a prescribed parameter configuration. The blueprint can be executed for different configurations. Therefore, parametric models represent not only one specific solution but the whole solution space, which is spanned by the chosen design parameters. Parametric models permit the modification of individual parameters, which allows changes to be applied quickly. By varying certain parameters, the solution space can be explored. As described in [4], the model can be automatically evaluated for a large number of randomly chosen parameter configurations. The designs are grouped computer-aided to present the designer with an overview of possible solutions. New solutions might be discovered that were not initially obvious to the designer.

The solution space of a parametric Lissajous curve is shown in Figure 2.1. This curve results from the superposition of two orthogonal oscillations. The blueprint of the curve is defined by a simple formula:

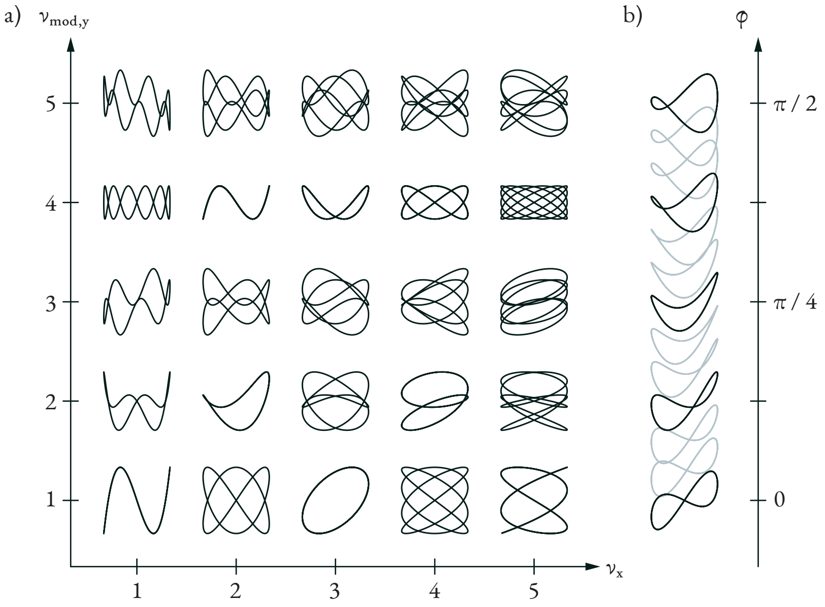
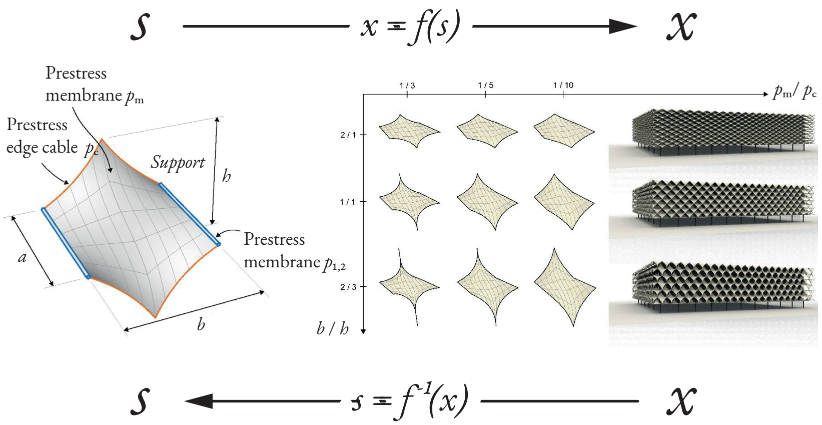


Figure 2.1: Lissajous curves result from the superposition of two oscillations. By changing the parameters of the individual oscillations, a variety of different shapes can be generated.

$$\mathbf{c}\,(\alpha) = \begin{bmatrix} \sin\left(\alpha\,\nu_x + \phi\right)\ \cos\left(\alpha\,\nu_{\mathrm{mod},x}\right) \\ \sin\alpha\,\nu_y\ \cos\alpha\,\nu_{\mathrm{mod},y} \end{bmatrix} \in \mathbb{R}^2 \qquad \text{with} \quad \alpha \in [0, 2\pi] \tag{2.1}$$

These formulas contain several parameters. The angle α is the *curve parameter* which passes through the parameter space of the curve and is used to plot the geometry. The shape of the curve can be controlled by the designer by changing the *design parameters* $\phi \in \mathbb{R}$ and $\nu_x, \nu_y, \nu_{\mathrm{mod},x}, \nu_{\mathrm{mod},y} \in \mathbb{Z}$. The variation of these parameters results in a large number of different shapes built according to the same principle. Figure 2.1 shows a subset of the solution space, which is spanned by the parameters ν_x, $\nu_{\mathrm{mod},x}$ and ϕ. ν_x, $\nu_{\mathrm{mod},x}$ are *discrete parameters*, which results in discontinuous transitions between the solutions. In contrast, the *continuous parameter* ϕ allows a smooth change of the shape. The influence of such a continuous parameter on the



$s \xrightarrow{x = f(s)} x$
Prestress membrane p_m
Prestress edge cable p_c
Support
h
a
b
Prestress membrane $p_{1,2}$
1 / 3
1 / 5
1 / 10
p_m / p_c
2 / 1
1 / 1
2 / 3
b / h
$s \xleftarrow{s = f^{-1}(x)} x$

Figure 2.2: A parametric model transforms a set of input parameters according to specified rules to a certain result. This results in a direct problem. Finding a parameter configuration which produces a certain result yields an inverse problem. (Adapted from [5])

solution can be estimated mathematically by computing the derivative of a property with respect to the parameter.

A parametric model can be extended arbitrarily. In Figure 2.2, a parametric model generates the input data for a form-finding analysis, which creates the shape of a façade module. The design parameters include dimensions and internal forces. The single module is then duplicated according to a specific pattern. The resulting model of the façade could then be used, e.g. for a sunlight study or wind analysis. When modelling the construction process of a structure, a structural analysis is performed for each building stage. Each analysis requires the results of the previous stage. The individual analyses are linked together sequentially, where the initial configuration of the design parameters has an influence on the final results. The data management when dividing the entire process into individual analyses is already a challenge that should not be underestimated. Determining the influence of the parameters on the final results within a sensitivity analysis makes the demands on data management and computational efficiency even higher. It is essential to use clear notation from which the interaction of parameters can be determined efficiently.

Even if a parametric model results in a wide range of the possible solutions, it is obvious that the choice of the design parameters and the definition of the model have a decisive influence on the set of possible solutions. For example, it is not possible to change the parameters of the Lissajous curve to create a square. Similarly, changing the parameters in Figure 2.2 will never turn the membrane structure into a grid shell. In both cases, the parametric model requires extensions which include new solutions in the solution space. At this point, it should be mentioned that the role of the designer in parametric modelling is very important. They decide according to which principles the solution is constructed and, thus also, which solutions can be contained in the solution space. It is necessary to be aware of the possibilities provided by the chosen parametrization. Constraints resulting from the parameterization should not be decisive for the design. Therefore, it must be possible to adapt the parameterization and the design efficiently to the requirements.

2.1 Algorithmic modelling

There are many possible notations to describe the relationship between the input and output of a parametric model. In the previous section, for example, it was a mathematical formula or a diagram. In general, the blueprint can be seen as a process which transforms the input parameters into a certain output according to specific rules. Algorithms are an elegant way to describe such processes in a formal way. On the computer, they can be represented with the help of programming languages. These languages have the advantage that they usually have a very clear structure as they are designed to be interpreted by machines, e.g. by a compiler. This makes it easier to analyze a process and extract additional information from the blueprint. The interactions between individual parameters, for example, can be determined automatically and leads to the subject of algorithmic differentiation.

In the field of geometric modelling, visual programming languages (VPL) are a common tool for defining geometric relations using algorithms. VPLs are often integrated directly into the CAD package. Due to their popularity in architecture, Grasshopper for Rhino, Dynamo for Autodesk Revit and GenerativeComponents for MicroStation should be highlighted. These tools are called node-based VPLs. Each node corresponds to a function, which has inputs and outputs. The function transforms the input parameters into the output. The outputs of one node can be connected to the inputs of another node. This allows complex processes to be described by combining basic building blocks.

Node-based VPLs share common features with functional programming languages. Unlike procedural programming, the algorithm is not composed of a set of instructions. Instead, the model results from a composition of functions. The result of a single function depends exclusively on the respective input parameters and only affects the direct output. Beyond that, no interaction between the functions takes place. Each function can be considered separately. There are advantages to the parallelization of programs written in a functional notation. Single nodes or branches of multiple nodes can be evaluated independently from each other as soon as all input parameters are available. The change of a parameter only influences the subsequent nodes. In this way, computational time can be saved by reevaluating only certain parts of the program. The functional notation of computation sequences offers a special advantage for algorithmic differentiation, which is described in more detail in Chapter 3.

In computer programs, parameters are intuitively transformed by functions and used as input for the following operations. This can be interpreted as a reparameterization. Figure 2.3 shows the reparameterization of a simple point. In the first case (a) the Cartesian coordinates (x, y) are prescribed explicitly. Similar to common VPLs, the design parameters which are controlled by the user are symbolized by sliders. In the second case (b), the point is described by polar coordinates by specifying an angle α and a radius r. The input parameters α and r are transformed by a series of functions to produce Cartesian coordinates. This corresponds to a reparameterization of the point from Cartesian to Polar coordinates. Altogether this results in a new function which generates a point from polar coordinates (c). This function corresponds to a composition of basic mathematical functions. The interaction between the input and output can be determined automatically if the structure of this composition is known.

2.2 Explicit and implicit parameterizations

A parametric model can be parameterized in different ways. The relationship between the input and output can be specified explicitly or implicitly. In an explicit parameterization, the input parameters are transformed directly into the desired solution. In an implicit parameterization, the solution is described by formulating a set of goals which the solution should satisfy.

Figure 2.4 shows how the geometry of a torus can be described via an implicit and an explicit parameterization. The angles α and β are the surface parameters, while

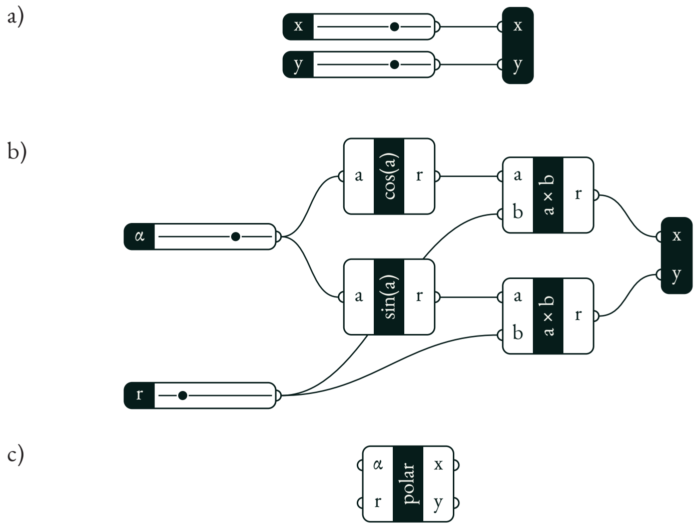


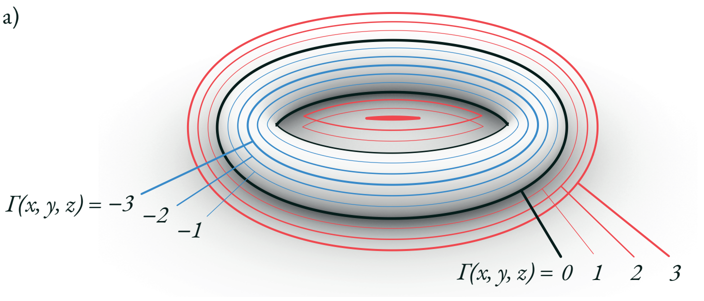
Figure 2.3: In node-based VPLs, each node corresponds to a function that transforms input parameters into output parameters. In (a), the node generates a point from Cartesian coordinates. In (b), polar coordinates are converted to Cartesian coordinates to generate a point. By prepending the additional functions, a reparameterization occurs. The resulting composition of functions results in a new function (c) which generates points by polar coordinates.

the radii R and r are the design parameters to control the size of the torus. The explicit parametrization transforms a pair α, β directly into a point x, y, z on the surface:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} (R + r\cos\beta)\cos\alpha \\ (R + r\cos\beta)\sin\alpha \\ r\sin\beta \end{bmatrix} \tag{2.2}$$

The evaluation of such an explicit expression can usually be visualized very well as a construction process. Therefore, one can also speak of a graphical solution.

An implicit parameterization describes the requirements for a spatial point x, y, z to lie on the surface. In (2.3), a level set function Γ is used to determine whether a



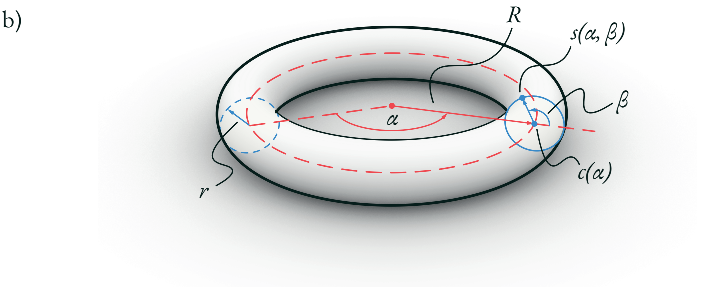


Figure 2.4: Implicit (a) and explicit (b) parameterization of a torus.

point is inside, on or outside the geometry:

$$\Gamma(x, y, z) = (R - \sqrt{x^2 + y^2})^2 + z^2 - r^2 = \begin{cases} < 0 & \text{inside} \\ = 0 & \text{on surface} \\ > 0 & \text{outside} \end{cases} \tag{2.3}$$

The implicit parametrization provides a kind of quality measure for a proposed solution. With an implicit parameterization, a compromise solution can be found when it is not possible to fulfil all goals. Such a solution fulfils the objectives in the sense of a *least-square solution*.

For simple cases, the implicit form can be transformed into an explicit one and vice versa. For more complex problems, this is usually not possible. One example is *combinatorial equilibrium modelling* (CEM) [6], where the nonlinear equation system that describes the equilibrium implicitly is transformed by substitution into an explicit parameterization which can be solved sequentially. For this, the topology of the structure must fulfil certain requirements. The solution process can be represented graphically and leads to a variety of application possibilities.

Depending on the application, an explicit or implicit parameterization can be more advantageous. An explicit parameterization is useful, e.g., if the solution can be determined constructively. The implicit parameterization is preferable if the solution can be described more easily by a set of goals. In practice, a combination of both variants is common. In Figure 1.1, for example, the geometries of the individual lines and arcs are explicitly parameterized. At the same time, additional constraints are applied between the individual geometries in an implicit way. The aim is to find the parameter configuration that best satisfies the constraints. This question leads to an inverse problem.

2.3 Inverse parametric modelling

A parametric model can be evaluated for different parameter configurations. For many practical applications, the parameter configuration that provides a certain result is required. This is the case, for example, if an implicit parametrization is used or if the desired variables cannot be controlled directly by an explicit parametrization. In this case, the problem is called an inverse problem because the parametric model has to be inverted (Figure 2.2). The inverted model would be able to compute the design parameters for a prescribed result. In practice, it is difficult or even impossible to invert a general problem. Instead, iterative methods are used, which modify the design parameters in each iteration in order to reduce the divergence from the desired result. This results in an optimization problem.

The general numerical optimization workflow is shown in Figure 2.5. Starting from an initial configuration of the design parameters, a parametric model is evaluated providing the actual state of the model. The result is then compared to a prescribed target or reference state. The deviation between the actual and the reference state is measured and provides an error. A key challenge is to capture the deviations numerically and weight them in a proper way. If this is done, different parameter configurations can be evaluated and compared. By simple trial and error, the best design can be determined from a limited number of parameter configurations. For simple problems, it might also be possible to plot the error over parts of the design space in order to find a solution. For practical problems, this is usually not

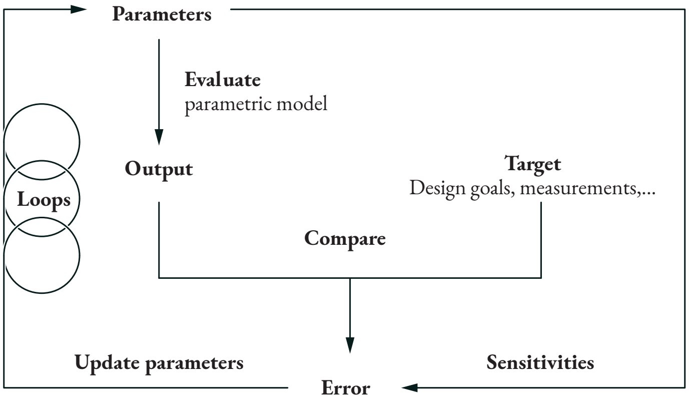


Figure 2.5: A classical optimization workflow: A parametric model is evaluated for a prescribed parameter configuration. The result is compared with a desired target, and the deviation is evaluated. Iteratively, the parameters are adjusted to minimize the error. If the influence of the parameters on the error is known in the form of sensitivities, gradient-based methods can accelerate the convergence of the solution.

possible because of the computational effort due to the number of design parameters and the complexity of the model. In this case, different optimization strategies, which approach an optimum in several iterations, can be applied to the problem. Optimization algorithms can be divided into derivative-free and gradient-based methods [7, 8].

Derivative-free strategies, such as evolutionary optimization strategies, generate a set of possible parameter configurations in each iteration and rate their quality according to the error function. Starting from a set of randomly chosen configurations, the most promising candidates are slightly modified for the following iteration while additional random candidates are added. In this way, the design space is searched until a suitable candidate is found or the best solution up to that point is selected after a certain number of iterations.

Gradient-based methods start from a given initial configuration. This configuration is then improved iteratively until a local minimum is found. To achieve an improvement, it is necessary to estimate the influence of the parameters on the error. This influence is described by the derivatives of the error function with respect to the design parameters, which results in the gradient. The gradient corresponds to a linearization of the error function at a certain point. It indicates the direction with the largest increase in the error. Therefore, a small modification of the design parameters in the opposite direction usually leads to an improved solution. This procedure is known as *gradient-descent strategy*. The search direction might be transformed to improve the modification, e.g. by applying the inverse Hessian as a transformation matrix for the gradient. This is called a *Newton-Raphson strategy*, as explained in Section 5.

Derivative-free strategies make no or no high requirements on the model to be optimized. Therefore, they can work with a black-box model where the relations between the input and output are unknown. Moreover, such algorithms can deal with discontinuities due to discrete parameters. Gradient-based algorithms usually expect an error function which is smooth and differentiable, at least in the search area. The computation of the derivatives represents additional computational effort, which might be compensated for a more targeted search and a faster convergence.

In this thesis, only gradient-based optimization algorithms for solving inverse problems are considered. The required gradients are determined automatically using algorithmic derivative techniques to deal with different geometry parameterizations.

2.4 Example

As an example, consider the Chebyshev lambda mechanism in Figure 2.6. This is a four-link coupling mechanism that converts a circular motion into a piecewise approximate straight-line motion with almost constant velocity. The mechanism was developed in the 19th century by *Pafnuty Lvovich Chebyshev* [9].

Nodes **A** and **B** are fixed, while nodes **C**, **D** and **E** are connected by four members. These members are assumed to be rigid, which is why **C**, for example, must move on a perfectly circular path around **A**. The motion path of nodes **C**, **D**, and **E** should be determined as a function of α. For the prescribed location of the supports and the lengths of the members $L_1 = 1, L_2 = L_3 = L_4 = 2.5$, the solution can be constructed graphically, which lead to an explicit construction rule:

1. The location of the fixed supports **A** and **B** are given.

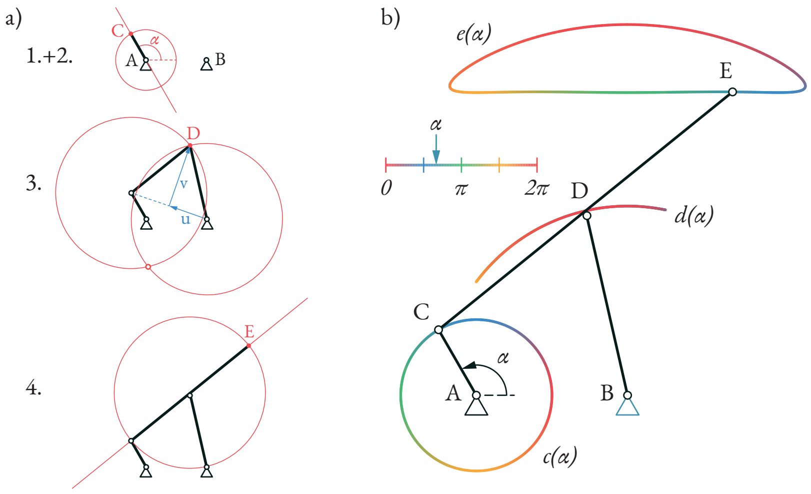


Figure 2.6: The mechanical system of the Chebyshev lambda mechanism. The motion of each node is parametrized by the angle α. The movement of each node builds on the movement of the previous nodes. This results in a composition of functions.

2. **C** can be constructed by intersecting a circle at **A** with $R = L_1$ with a line with angle α.
3. The intersection of two circles at **B** and **C** with $R = 2.5$ provides two possible solutions for **D**.
4. **D** is chosen as the intersection, which is on the right side of **BC**.
5. Extending the line **CD** by 2.5 units provides the location of node **E**.

Thus the position of all nodes is determined for a specific α. A parametric model can be generated that explicitly provides the positions depending on α by repeating the graphic construction process. It results in the curves $\mathbf{c}(\alpha)$, $\mathbf{d}(\alpha)$ and $\mathbf{e}(\alpha)$, which represent the motion of each node. This illustrates how a linear motion in parameter space α is transformed first into a circular motion **c**, an arc **d**, and finally into a sectional nearly linear motion **e**.

The graphic solution is a visualization of an algorithm which can also be represented as a simple computer program:

```
$\mathbf{A} \leftarrow [0, 0]$;
$\mathbf{B} \leftarrow [2, 0]$;
$\mathbf{C} \leftarrow \mathbf{A} + [\cos(\alpha), \sin(\alpha)]$;
$\mathbf{v} \leftarrow 0.5 \times (\mathbf{C} - \mathbf{B})$;
$h \leftarrow \sqrt{2.5^2 - \mathbf{v} \cdot \mathbf{v}}$;
$\mathbf{u} \leftarrow h/\sqrt{\mathbf{v} \cdot \mathbf{v}} \times [v_y, -v_x]$;
$\mathbf{D} \leftarrow \mathbf{B} + \mathbf{v} + \mathbf{u}$;
$\mathbf{E} \leftarrow 2 \times \mathbf{D} - \mathbf{C}$;
```

Likewise, the representation as a mathematical function is also possible:

$$\mathbf{c}(\alpha) = \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix} \tag{2.4}$$

$$\mathbf{d}(\alpha) = \frac{1}{2} \begin{bmatrix} 2 - \cos(\alpha) \\ \sin(\alpha) \end{bmatrix} + \frac{\sqrt{5 + \cos\alpha}}{\sqrt{5 - \cos\alpha}} \begin{bmatrix} \sin\alpha \\ -\cos\alpha \end{bmatrix} \tag{2.5}$$

$$\mathbf{e}(\alpha) = \begin{bmatrix} 2 \\ 0 \end{bmatrix} + 2\,\frac{\sqrt{5 + \cos\alpha}}{\sqrt{5 - \cos\alpha}} \begin{bmatrix} \sin\alpha \\ -\cos\alpha \end{bmatrix} \tag{2.6}$$

In any case, one obtains an explicit relationship between α and the node positions. The position can be calculated directly and unambiguously. Which kind of description is more suitable may depend on the specific problem.

In step 3 of the graphic solution, two intersection points are obtained. This illustrates that there are two solutions to position the nodes. The second solution, where nodes **D** and **E** are mirrored around **CD**, is explicitly excluded. This is done in the same way for the representation as a program or formula. To get the second solution, only the sign of u has to be changed.

In an implicit parameterization, the solution is not computed directly. Instead, it is described by a set of goals. The solution is expected to satisfy all the objectives. In the case of the lambda mechanism, these goals can be as follows:

$$g_1 = \frac{|\mathbf{C} - \mathbf{A}|^2}{1^2} - 1 \to 0 \tag{2.7}$$

$$g_2 = \frac{|\mathbf{E} - \mathbf{C}|^2}{5^2} - 1 \to 0 \tag{2.8}$$

$$g_3 = \frac{|0.5(\mathbf{E} - \mathbf{C})|^2}{2.5} - 1 \to 0 \tag{2.9}$$

$$g_4 = \angle(\mathbf{B}, \mathbf{A}, \mathbf{C}) - \alpha \to 0 \tag{2.10}$$

The constraints g_1, g_2 and g_3 describe the length of the lines **AC**, **CE** and **BD**, where g_3 includes the fact that **D** is located between **C** and **E**. This explicitly describes part of the structure, namely the connection between **C**, **D** and **E**. Instead, it would also be possible to describe the segments **CD** and **DE** separately and also to specify the angle between the two segments. The constraint g_4 specifies a certain angle α. This constraint can be exchanged to control other parameters. For example, the position of the node **E** can be specified instead. In this way, the objective can be assembled conveniently from individual elements.

In any case, one obtains a set of nonlinear equations where the nodal coordinates are the unknown quantities. Parameter configurations that describe a state compatible with the kinematics of the mechanism satisfy these equations. If one wants to determine the parameters such that the constraints are satisfied, an inverse problem arises, as described in the previous section.

The constrained problem can be transformed into an unconstrained optimization problem by using a penalty approach. Therefore, the constraints are accumulated into an objective function:

$$\Gamma = \frac{w_1}{2} {g_1}^2 + \frac{w_2}{2} {g_2}^2 + \frac{w_3}{2} {g_3}^2 + \frac{w_4}{2} {g_4}^2 \tag{2.11}$$

The influence of each constraint g_i on the objective Γ is weighted by a penalty factor w_i. The penalty factor can be interpreted mechanically as a stiffness. In the simple case, all constraints are weight equally.

Using gradient-based optimization, the unknown positions of the nodes **C**, **D** and **E** can be computed. An initial configuration is improved iteratively until all constraints are satisfied. Also, in this case, two solutions are possible. Since no explicit selection is made, it depends on the initial configuration and the optimization algorithm, which of the two solutions is found.

The graphical and analytical solution allows to determine the position of node **E** directly for a given angle α. However, if α should be determined for a given **E**, a gradient-based optimization might be used to solve the inverse problem. This makes it possible to control variables that are not explicitly available as design parameters. In the context of CEM, for example, additional constraints can be integrated into the problem as shown in [10] and [11]. For the analytical formula, the determination of the derivatives is straightforward. If the blueprint of the solution is available as an algorithm, the derivatives can be determined with the help of algorithmic differentiation as described in Chapter 3. This technique can also be used to compute the derivatives of the constraints g_i for the implicit solution. In this way, isogeometric finite elements are implemented in Chapter 5 and 6. Furthermore, this technique is used to determine the derivatives for freeform geometries. This is particularly helpful when dealing with extended subdivision surfaces as shown in Chapter 6.

3

Algorithmic differentiation

Solving inverse problems using gradient-based methods requires the derivatives of the objective with respect to the design parameters. To solve complex problems, efficient computation of these derivatives is necessary. Due to the complexity, it is assumed that these calculations are performed by a computer. In this context, the term 'efficiency' covers three aspects: (i) a simple and intuitive implementation, (ii) a fast computation that minimizes computation time and memory consumption, and (iii) a systematic approach that simplifies the maintenance and extension of an existing framework. In the context of this work, those requirements are implemented by applying algorithmic differentiation (AD).

The derivative of a function $f(x)$ with respect to a parameter x describes how the value of f is influenced by a change in x. The first-order derivative can be visualized as the slope or tangent of the function at a certain position and is therefore called the gradient. Figure 3.1 visualizes the gradient for a univariate and bivariate scalar function. The concept can be extended to an arbitrary number of input and output parameters.

The first-order derivative yields a linear approximation T_1 of f at $\hat{x}$. This approximation can be generalized to a higher polynomial order by using a Taylor series. The second-order Taylor polynomial T_2 includes not only the function value but also

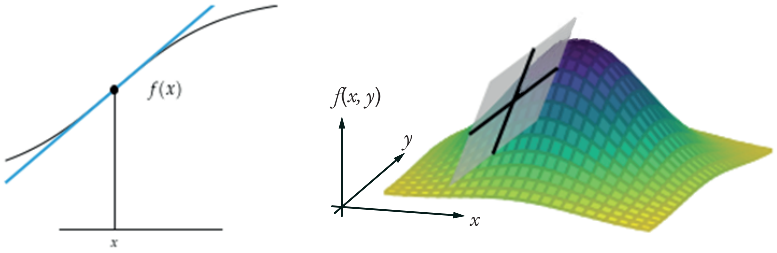


Figure 3.1: The first derivative of a function corresponds to the slope at a certain point. It can be visualized as a line for a scalar univariate function or as a plane for a scalar bivariate function (adapted from [8]).

the first two derivatives and leads to a quadratic approximation of f at $\hat{x}$.

$$T_n f(x, \hat{x}) = \sum_{k=0}^{n} \frac{f^{(k)}(\hat{x})}{k!} \Delta x^k \qquad \text{with} \quad \Delta x = (x - \hat{x}) \tag{3.1}$$

$$= \underbrace{\underbrace{f(\hat{x}) + f'(\hat{x})\,\Delta x}_{T_1} + \frac{1}{2} f''(\hat{x})\,\Delta x^2}_{T_2} + \dots \tag{3.2}$$

The extrema x_{ext} of T_2 is given by (3.2):

$$x_{\text{ext}} = \hat{x} - \frac{f'(\hat{x})}{f''(\hat{x})} \tag{3.3}$$

By iteratively determining the extrema of T_2, an inverse problem can be solved using the Newton-Raphson process, as explained in Chapter 5. This chapter clarifies the efficiency advantage of AD and provides details on the implementation used in Chapters 5 and 6.

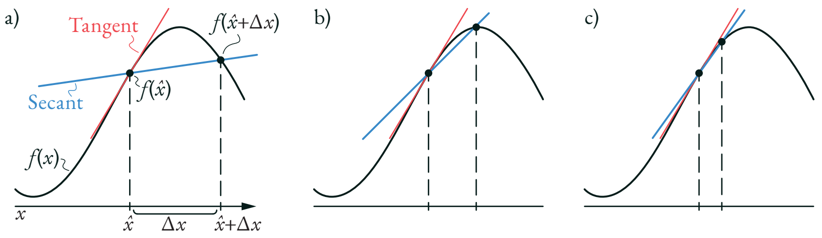


Figure 3.2: Finite differences approximate the derivative of a function with a secant (a). With decreasing step size Δx, the secant converges to the tangent (b and c) (adapted from [8]).

3.1 Finite differences

Derivatives can be approximated numerically by finite differences (FD). FD evaluates f twice, where the step size Δx specifies the distance between the two evaluation points. This way, the tangent of f is approximated by a secant, as shown in Figure 3.2. There are forward, central and backward differences:

$$f'(x) \approx \underbrace{\frac{f(x+\Delta x) - f(x)}{\Delta x}}_{\text{Forward}} \approx \underbrace{\frac{f\left(x+\frac{\Delta x}{2}\right) - f\left(x-\frac{\Delta x}{2}\right)}{\Delta x}}_{\text{Central}} \approx \underbrace{\frac{f(x) - f(x-\Delta x)}{\Delta x}}_{\text{Backward}} \tag{3.4}$$

With a smaller Δx the secant converges to the tangent. However, too small a value might cause numerical problems, as explained in Chapter 5. Another problem is the computational cost of FD which increases drastically for multivariate functions since f must be evaluated for a distortion of each parameter. This issue will be illustrated in this section.

Consider the case with one parameter shown in Figure 3.3. It visualizes the process of FD for a scalar univariate function $f = g \circ h$, which is obtained by concatenating two functions, g and h. The evaluation of $f(x) = g(h(x))$, determines $u = h(x)$ (bottom right) and evaluates $g(u)$ (top left) to get the value of f at x (top right). Computing the derivative $f'(\hat{x})$ using forward finite differences requires a second evaluation of f at $\hat{x} + \Delta x$. A large step size Δx is chosen to illustrate the deviation between the secant and tangent. In practice, a small step size would be chosen

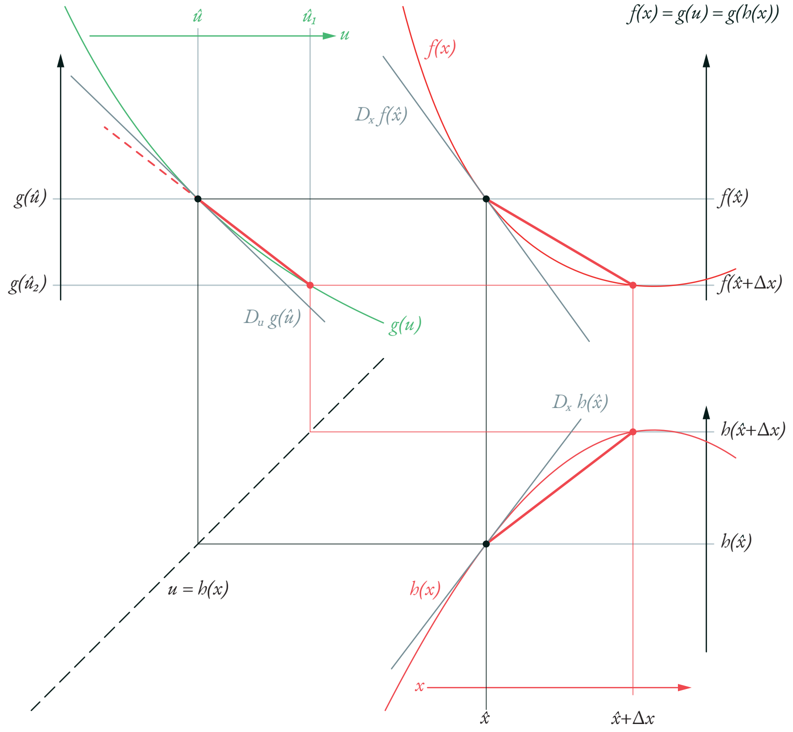
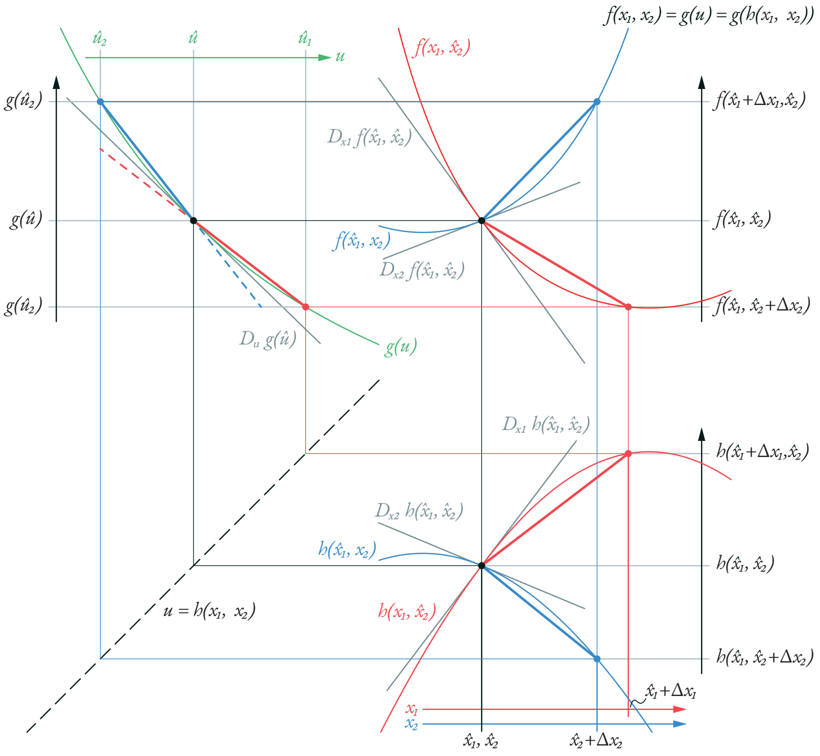


Figure 3.3: Forward finite differences for a composed function $f = g \circ h$ with a parameter x. g and h are each evaluated at two points to approximate the tangent with a secant.



$f(x_1, x_2) = g(u) = g(h(x_1, x_2))$
$\hat{u}_2$
$\hat{u}$
$\hat{u}_1$
u
$f(x_1, \hat{x}_2)$
$g(\hat{u}_2)$
$f(\hat{x}_1{+}\Delta x_1, \hat{x}_2)$
$D_{x1}\, f(\hat{x}_1, \hat{x}_2)$
$g(\hat{u})$
$f(\hat{x}_1, \hat{x}_2)$
$f(\hat{x}_1, x_2)$
$D_{x2}\, f(\hat{x}_1, \hat{x}_2)$
$g(\hat{u}_2)$
$f(\hat{x}_1, \hat{x}_2{+}\Delta x_2)$
$D_u\, g(\hat{u})$
$g(u)$
$D_{x1}\, h(\hat{x}_1, \hat{x}_2)$
$h(\hat{x}_1{+}\Delta x_1, \hat{x}_2)$
$D_{x2}\, h(\hat{x}_1, \hat{x}_2)$
$h(\hat{x}_1, x_2)$
$h(\hat{x}_1, \hat{x}_2)$
$u = h(x_1, x_2)$
$h(x_1, \hat{x}_2)$
$h(\hat{x}_1, \hat{x}_2{+}\Delta x_2)$
$\hat{x}_1{+}\Delta x_1$
x_1
x_2
$\hat{x}_1, \hat{x}_2$
$\hat{x}_2{+}\Delta x_2$

Figure 3.4: Forward finite differences for a composed function $f = g \circ h$ with two parameters, x and y. g and h are each evaluated at three points to approximate the tangent with a secant. For the function g, the same tangent is approximated in two different ways, resulting in unnecessary extra work.

to increase the quality of the approximation. To compute the secants of f, the functions g and h are each evaluated twice.

A second parameter extends the example. In this case, $f = g(h(x_1, x_2))$ is a scalar bivariate function which results from a concatenation. Now h and thus also f depend on two parameters, x_1 and x_2. Figure 3.4 visualizes the computation of the derivatives of f with respect to x_1 and x_2 using forward FD. Similar to the univariate example, the function is evaluated at the point $\hat{x}_1, \hat{x}_2$. The two parameters are deflected separately by Δx_1 and Δx_2, and the corresponding function values $\hat{u}_1$, $\hat{u}_2$ are determined. To compute the two required secants of f, the functions g and h are evaluated three times. During the computation, the tangent of g is approximated by two different secants (Figure 2 top left). A sufficiently small step size should lead to similar results for both secants. However, this redundant calculation could be avoided if the secant for the first parameter could be reused for the second run.

Such redundant computations might increase the computational effort for FD when it is applied to multivariate functions. The problem can be recognized and avoided if the structure of the composed function is known. FD is usually applied to functions whose structure is unknown (black box). For functions with a known structure, the computational efficiency might be improved by using AD.

3.2 Algorithmic differentiation

In the previous section, it was shown that the computational effort could be reduced if the structure of the composed function is known. In this case, the partial derivatives of each suboperation (e.g. g and h) can be calculated separately and accumulated according to the chain rule. This avoids redundant computations when dealing with multivariate functions. Computing the individual partial derivatives can take place numerically or symbolically. AD deals with the consistent implementation of this concept in arbitrary algorithms.

The calculation of the derivative follows a particular calculus. This set of rules is well-suited for systematic processing with the aid of a computer. AD encompasses the techniques that transform the algorithm that evaluates a parametric model into an algorithm that evaluates the derivatives of the model. The technique goes back to John McCarthy [12]. He introduces the basic features of the functional programming language LISP, which was designed to work with symbolic expressions. LISP represents algorithms and data in the same way: by chained lists. One speaks, therefore, of a homoiconic ('self-representable') programming language. Due to this property, it is particularly suitable for interpreting and manipulating algorithms (metaprogramming). In LISP, one can very elegantly transform a computer program according to the differentiation rules in such a way that a new program is generated, which computes the derivatives [13]. The concepts can also be applied to other programming languages. The essential step is to capture and transform the structure of an algorithm.

3.3 Computational graph

As mentioned in Chapter 2, the structure of a computer program can be visualized with a graph. Such a computational graph visualizes the computational process within a parametric model as a composition of functions. Figure 2.3 shows the computation of a point via polar coordinates using a node-based VPL. Due to the visual and functional nature of this language, this definition already corresponds to a visual representation of the algorithm as a composition of elementary suboperations.

For each suboperation, the interaction between the input and output can be determined by a linearization given by the first derivative of the function. These are represented in the graph with the help of weighted edges. The weighting corresponds to the partial derivative. For the example from 2.3, this results in the weighted graph shown in Figure 3.5. The function is decomposed in suboperations visualized by nodes. The analytical derivatives of each suboperation are stored and evaluated at the same as the functions. This results in numerical values for the weights of the edges.

The derivation of the whole composition is obtained from the graph by considering the paths connecting the input and output. For example, there is a path connecting the parameter α with the x-coordinate of the point (Figure 3.5a red). Multiplying the weights on this path results in the derivative $D_\alpha x = -\sin(\alpha)\, r$. The same technique can be used to compute the other derivatives, $D_r x$, $D_\alpha y$ and $D_r y$.

The procedure corresponds to the application of the chain rule and yields the evaluated analytical derivatives. There is no divergence as a result of an approximation, as is the case for FD. The process can also be considered a transformation of the computational graph. Two consecutive edges are replaced by a new edge. The weight of the new edge results from the product of the weights of the former edges. The corresponding transformation rule is shown in Figure 3.6a.

In the following sections, the computational graph is simplified. For this purpose, the edges connecting the output and input of different nodes are omitted since they always have a weight of one. In addition, the parameters are arranged in layers (Figure 3.5b). The outer layers correspond to the input and output parameters. By applying the transformation rule, the layers containing the intermediate results can be eliminated. This results in a new graph which contains the direct connections between the input and output.

The interaction between the parameters on the same levels can be specified by

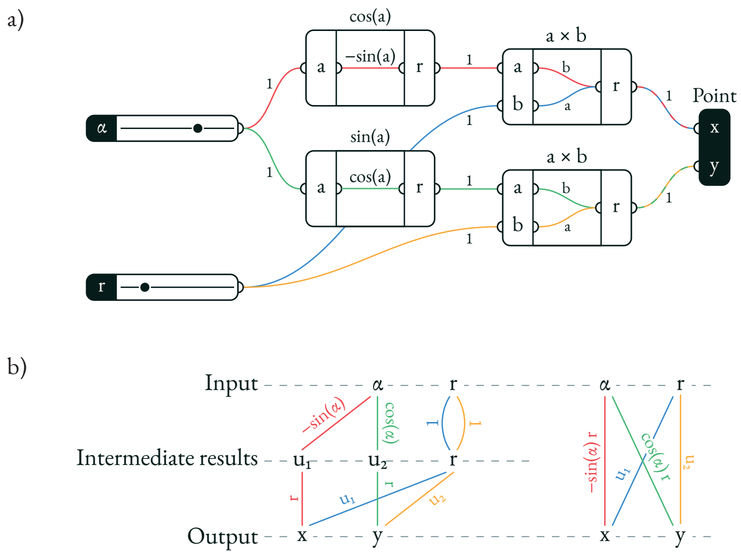


Figure 3.5: For the example from Figure 2.3, the relationships between the input and output of the individual operations were added (a). The representation can be simplified (b).

additional edges whose weights correspond to the second-order derivatives. This will be described in more detail in Chapter 5. Additional transformation rules can be defined to eliminate the layers with intermediate results for second-order derivatives (Figure 3.6b and c).

3.4 Direct and adjoint methods

When applying the chain rule, the transformation rules can be applied in different orders. This leads to the distinction between the direct and adjoint methods or the forward and backward mode for computing derivatives. Even if both variants lead to the same final result, the order might have a big impact on the computational effort. In Chapter 5, this technique is used to increase the efficiency in the calculation of IGA elements. The distinction between direct and adjoint methods will be discussed in Chapter 5 and Chapter 6 in more detail.

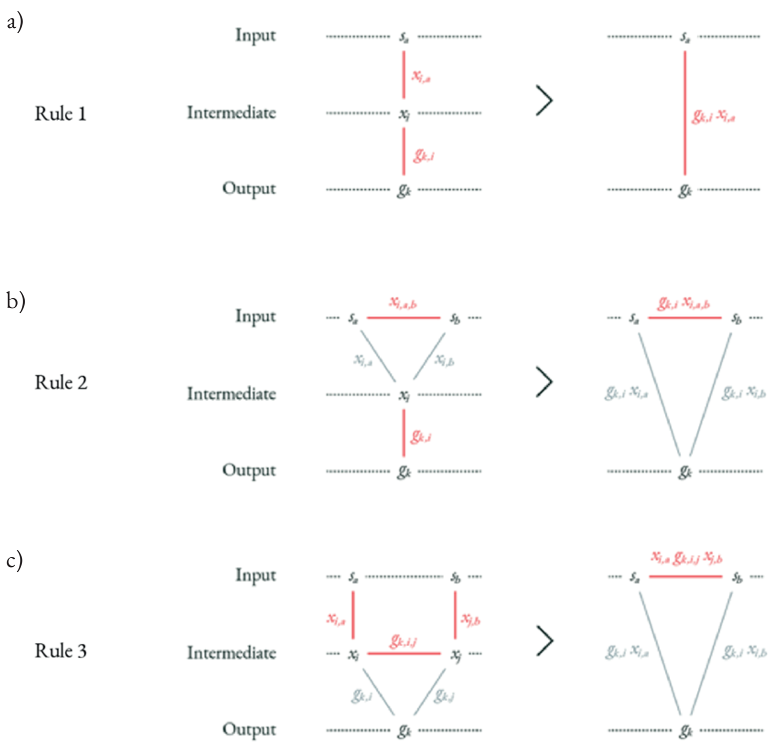


Figure 3.6: Transformation rules for calculating the derivative of a function. Rule 1 is needed for the calculation of the first derivative, rules 2 and 3 for the second-order derivative. Their application eliminates one level of intermediate results.

3.5 Implementing AD using hyperdual numbers

The forward mode of AD can be implemented in a simple but elegant way by using hyperdual numbers [14]. This approach has the advantage that it can be realized with simple commands and can therefore be integrated very well into an existing FE framework.

Similar to complex numbers, dual numbers extend a real number by additional components. For complex numbers, this is a complex unit; for dual numbers, it is an infinitesimal or dual unit. To perform AD, the dual component is used to store the derivative of the real value with respect to the design parameters. An additional hyperdual component stores the second-order derivative. This way, a hyperdual number represents the evaluated second-order Taylor polynomial of a function for a certain parameter configuration. Thus, a hyperdual number represents not only the value of a function but also the quadratic approximation at the evaluation point.

Mathematical operations with hyperdual arguments are defined in a way that result in hyperdual numbers. The result consists of the function value as well as the corresponding derivatives. A classic algorithm which transforms real parameters into a real result can be executed with hyperdual parameters to return hyperdual results. For this, all suboperations used within the algorithm must be defined for hyperdual numbers. The transformations from Figure 3.6 are applied synchronously with the execution of the algorithm. Therefore, this process corresponds to the forward or direct mode of AD.

For functions with several parameters, a dual number can be expanded to a jet. A jet has not only one but several dual components, which take the derivatives with respect to the individual parameters. Similarly, several hyperdual components can be introduced to represent the second-order derivatives. Thus, the second-order Taylor polynomial of a multivariable function can be represented.

The implementation is done using a simple data type, which stores the real and dual components. A simple vector is used to store all values, as shown in Figure 3.7. The first entry corresponds to the real component, followed by the dual and hyperdual components. Assuming that the Hessian is symmetric, only the entries of the upper triangular matrix are stored in row-major format to reduce memory consumption and avoid redundancies. For a function with s design parameters, this vector has a length of

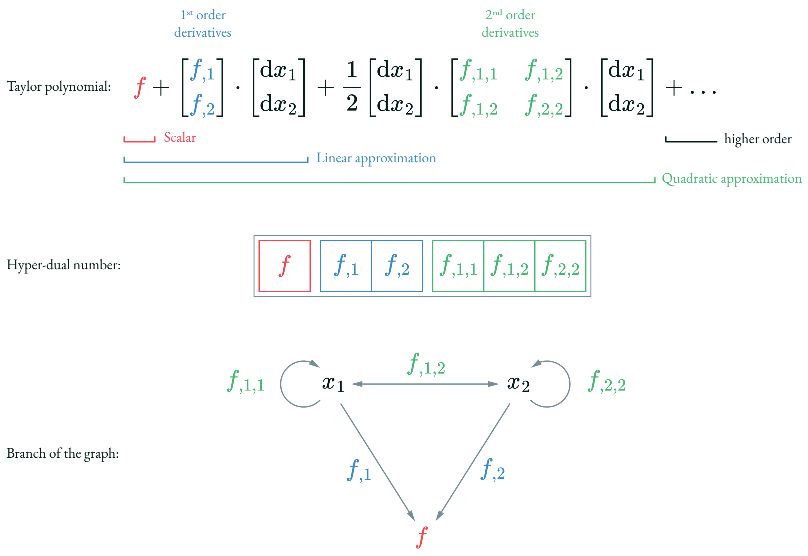

1st order derivatives
2nd order derivatives
Taylor polynomial:
$f + \begin{bmatrix} f_{,1} \\ f_{,2} \end{bmatrix} \cdot \begin{bmatrix} dx_1 \\ dx_2 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} dx_1 \\ dx_2 \end{bmatrix} \cdot \begin{bmatrix} f_{,1,1} & f_{,1,2} \\ f_{,1,2} & f_{,2,2} \end{bmatrix} \cdot \begin{bmatrix} dx_1 \\ dx_2 \end{bmatrix} + \dots$
Scalar
Linear approximation
Quadratic approximation
higher order
Hyper-dual number:
f $f_{,1}$ $f_{,2}$ $f_{,1,1}$ $f_{,1,2}$ $f_{,2,2}$
Branch of the graph:
$f_{,1,1}$ x_1 $f_{,1,2}$ x_2 $f_{,2,2}$
$f_{,1}$ $f_{,2}$
f

Figure 3.7: A hyperdual number represents the second-order Taylor polynomial of a function at the evaluation point.

$$n = 1 + s + \frac{s(s+1)}{2} \tag{3.5}$$

A constant value only consists of a real component. It is independent of the design parameters, and therefore dual, and hyperdual components are equal to zero. For example, π is represented with respect to two parameters, x_1 and x_2, as follows:

$$\pi \longrightarrow [3.141\ldots, 0, 0, 0, 0, 0] \tag{3.6}$$

A design parameter consists of a real component and depends only on itself. The corresponding dual component is, therefore, equal to 1, while all other entries are zero. For the design parameters x_1 and x_2, this leads to the following representation:

$$x_1 \longrightarrow [x_1, 1, 0, 0, 0, 0] \tag{3.7}$$

$$x_2 \longrightarrow [x_2, 0, 1, 0, 0, 0] \tag{3.8}$$

Operations are now defined according to the derivation rules. The partial derivatives of the operation with respect to its parameters are required. The number of arguments of a function is called its 'arity'. The cosine function is an example of a unary function which means that it depends on a single argument. For the first- and second-order derivatives, we get:

$$f(a) = \cos(a) \qquad D_a f = -\sin(a) \qquad D_a D_a f = -\cos(a) \tag{3.9}$$

The transformations can now be applied according to the following scheme:

```python
def unary(s, n, a, f, da, dada, r):
  r[0] = f
  for i in range(1, n):
    r[i] = da * a[i]
  k = 1 + s
  for i in range(s):
    ca = dada * a[1 + i]
    for j in range(i, s):
      r[k] += ca * a[1 + j]
      k += 1
```

The quotient of two numbers is an example of a binary operation. For the first- and second-order derivatives, we get:

$$f(a,b) = \frac{a}{b} \quad D_a f = \frac{1}{b} \quad D_b f = a \quad D_a D_a f = 0 \quad D_a D_b f = -\frac{1}{b^2} \quad D_b D_b f = 0 \tag{3.10}$$

The transformations can now be applied according to the following scheme:

```
def binary(s, n, a, b, f, da, db, dada, dadb, dbdb, r):
  r[0] = f
  for i in range(1, n):
    r[i] = da * a[i] + db * b[i]
  k = 1 + s
  for i in range(s):
    ca = dada * a[1 + i] + dadb * b[1 + i]
    cb = dadb * a[1 + i] + dbdb * b[1 + i]
    for j in range(i, s):
      r[k] += ca * a[1 + j] + cb * b[1 + j]
      k += 1
```

The principle can be extended to more parameters. A generalization to arbitrary arity and vectorial output is shown in Chapter 5.

This type of implementation of AD can be extended to higher-order derivatives. For a large number of design parameters, it may be more efficient to store the dual components in sparse rather than dense vectors. Both features are implemented, e.g. in [15]. For the application of element formulation described in this thesis, only the first and second derivatives are required. With FEA, a large sparse problem is divided into smaller finite elements where residual force vectors and stiffness matrices are populated fairly densely. At the element level, the use of hyperdual numbers can thus be very efficient.

In terms of computational efficiency, it is beneficial to explicitly define derivation rules for recurring operations. Even if a decomposition into basic operations would be possible and lead to the correct result, this is often associated with a considerable additional effort.

4

Freeform geometries

The form of basic geometric shapes such as lines, circles, prisms or spheres can be defined by a few parameters. They include lengths and radii, as well as transformations like rotations or scaling. The basic character of the shape is usually preserved when these parameters are varied (Figure 4.1a).

Freeforms offer greater design freedom by increasing the number of design parameters. This type of geometry includes polygons and discrete meshes. Their geometry is determined by the control points that are combined into a control mesh. The position of each control point in space is specified. The points can be connected linearly by lines and triangles. This results in polygonal lines and triangular meshes. The number of design parameters and the solution space increase with the number of control points. Complex geometries can be represented with a higher number of control points (Figure 4.1b).

The control mesh can be recursively refined according to certain subdivision schemes. This process converges to a smooth geometry described by polynomials of higher order (Figure 4.1c). Thus, smooth curves and surfaces can be described with fewer control points. This chapter explains the subdivision schemes of some essential refinement algorithms.

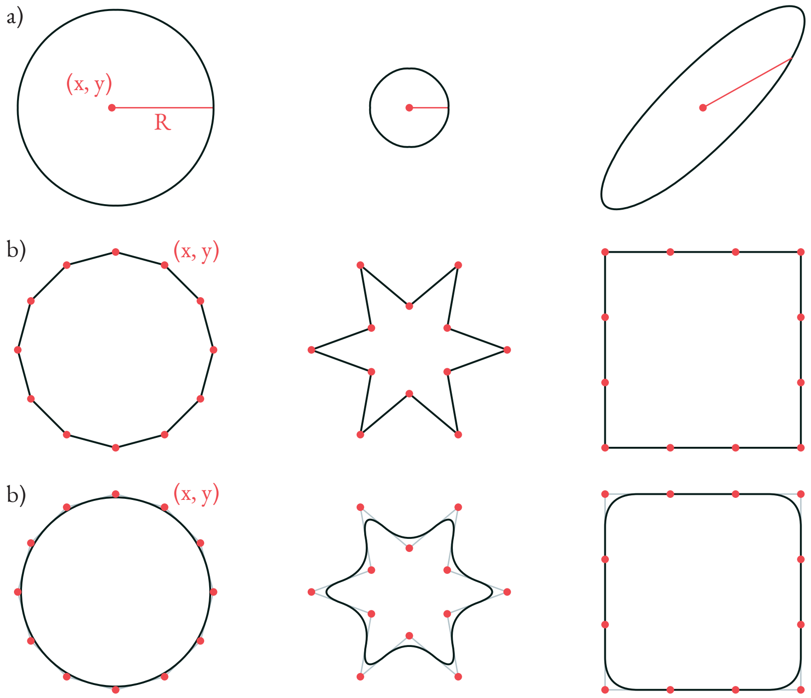


Figure 4.1: Different parametrizations of geometries. In (a), a circle is parametrized by a centre point and a radius. Changing the radius or scaling the geometry yields similar shapes. The circle is approximated in (b) with a polyline. The polyline is parametrized by the location of the control points. This offers more design possibilities. In (c), a quadratic B-Spline is used to smooth the control polygon. This simplifies the representation of smooth geometries but cannot produce sharp corners.

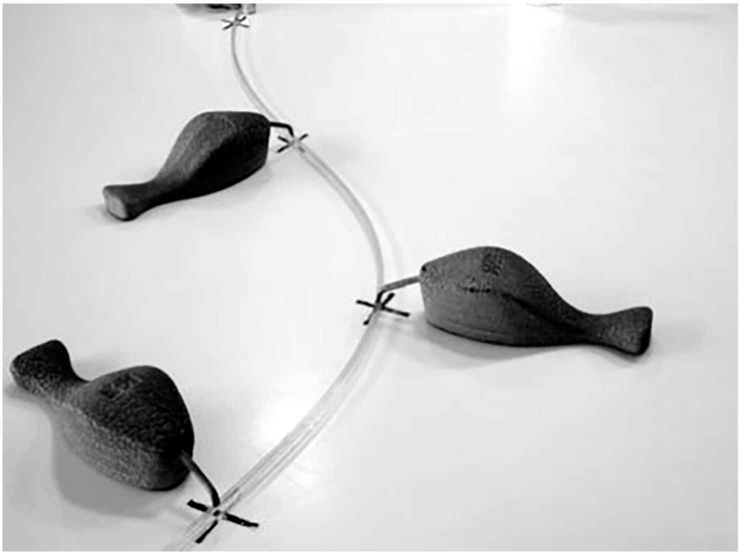


Figure 4.2: Traditional splines are bendable rods whose shape is controlled by some fixed points [16].

4.1 Freeform curves

The representation of smooth curves using a set of control points goes back to elastic rulers. The shape of these so-called splines was fixed at specific points with weights (Figure 4.2). With the advent of CAD, it became necessary to represent freeform geometries in an exact and reproducible form on the computer. Different algorithms have been developed. Most of them refine a coarse mesh according to a certain rule. The refinement process converges to a limit geometry, which has certain properties, e.g. smoothness and can be expressed by simple construction algorithms.

De Casteljau's algorithm

Bézier curves are widely used to define freeform curves in CAD [17]. Paul de Casteljau in 1959 and Pierre Bézier in 1962 developed this type of curve independently

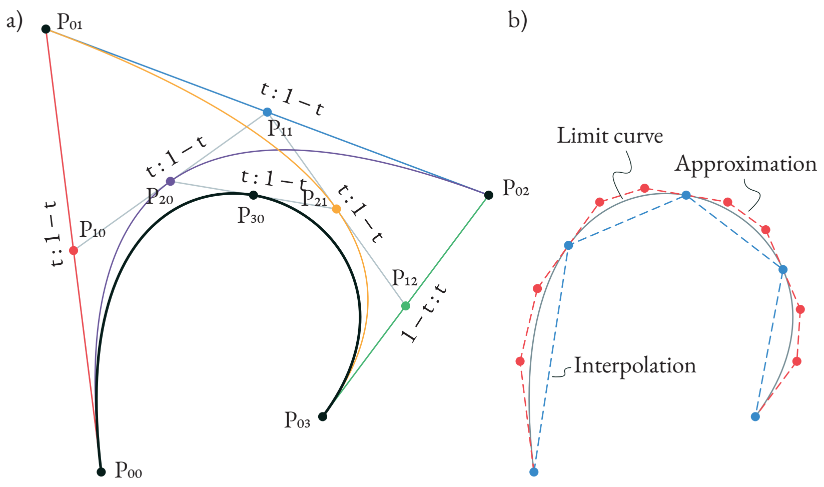


Figure 4.3: De Casteljau's algorithm generates a smooth Bézier curve of polynomial degree $p = n - 1$ from a coarse control polygon with n control points. (a) Smoothing the line segments of the control polygon (red, blue, green) results in two quadratic splines (purple and orange) and finally in a cubic spline (black). (b) Evaluating the Bézier curve for multiple parameters results in an interpolation (blue). From the intermediate results of the algorithm, the curve can also be approximated with a refined control polygon (red).

of each other for automotive design.

A Bézier curve of polynomial degree p is defined by a control polygon which consists of $n = p + 1$ control points $\mathbf{P}_{00}, \mathbf{P}_{01}, \mathbf{P}_{02}, \dots$ (Figure 4.3). For a curve parameter defined in the normalized domain $t \in [0, 1]$, the corresponding point on the Bézier curve can be determined by applying De Casteljau's algorithm. Therefore, each segment of the control polygon is divided in the ratio $(t : 1 - t)$. This results in $n - 1$ new points, $\mathbf{P}_{10}, \mathbf{P}_{11}, \mathbf{P}_{12}, \dots$, which are connected to a polygon. The segments of this polygon are subdivided in the same way. Repeating this process $(n - 1)$ times results in a single point, which represents the smooth Bézier curve at parameter t. Figure 4.3a shows the construction process for a cubic Bézier curve ($p = 3$) defined by $n = 4$ control points.

The process can be repeated for different curve parameters in order to represent the smooth curve. The resulting points are connected to form a polygon which

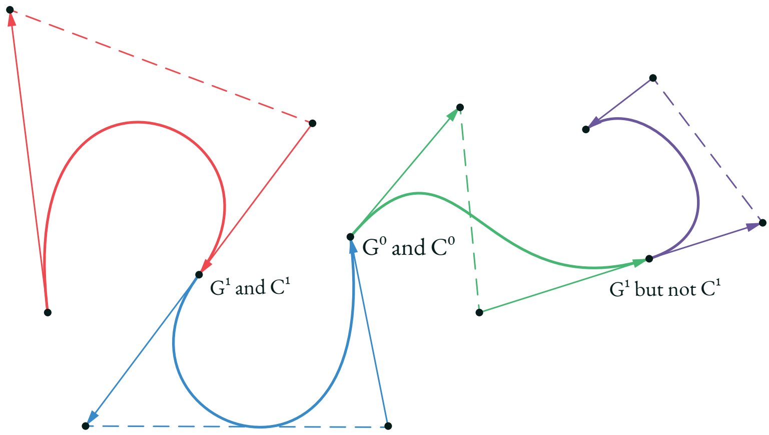


Figure 4.4: The continuity between individual cubic Bézier segments can be controlled by the alignment of the control points. By choosing identical start and end points G0 and C0 continuity is a given. If the direction of the adjacent segments of the control polygon coincides, a G1 continuity (no kink) is created. If the lengths of the segments also match, C1 continuity is created (no change in the traversing speed).

interpolates the smooth Bézier curve (Figure 4.3b blue). By increasing the number of evaluation points, the interpolation becomes more accurate. Alternatively, the Bézier curve can also be approximated by a polygon (Figure 4.3b red). Looking at the intermediate results of Casteljau's algorithm in Figure 4.3, we see that the point sequences $(P_{00}, P_{10}, P_{20}, P_{30})$ and $(P_{30}, P_{21}, P_{12}, P_{03})$ define two polygons that approximate the smooth Bézier curve. In fact, these are again control polygons, where each defines a subdomain, $[0, t]$ and $[t, 1]$, of the original Bézier curve. By applying De Casteljau's algorithm to these polygons, the approximation can be refined stepwise. In this sense, this algorithm is a subdivision scheme which refines a coarse polygon towards a smooth limit curve.

Cubic Bézier curves are particularly popular in geometric modelling because the influence of the four control points on the shape is very intuitive for the design. While the outer control points define the end points of the curve, the inner control points define the directions of the tangents. Several cubic Béziers can be connected, as shown in Figure 4.4. By aligning the control points, the derivatives at the transitions can be adjusted to create continuous joints without kinks. This alignment can be explicitly incorporated into the definition of the curve and results in *B-splines*.

De Rham-Chaikin's algorithm

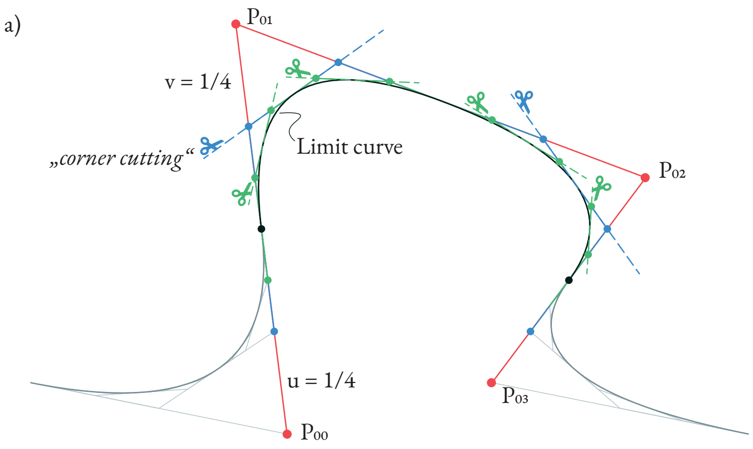
Georges De Rham described an important subdivision algorithm in 1947 [18]. Starting from a rough control polygon, each segment is divided into three sections of equal length (Figure 4.5). The two outer parts are discarded, while the once middle parts are joined to a new polygon. The algorithm is, thereforem also known as the 'cutting-edge' algorithm. By applying the algorithm recursively, the resulting polygon converges to a Co-continuous curve.
The subdivision ratio $(1/3 : 1/3 : 1/3)$ can be generalized to $(u : 1 - u - v : v)$. By choosing u and v, curves with different properties can be generated. De Rham recognized that his algorithm converges to a $C1$-continuous curve for $u = v = 1/4$ (Figure 4.4). This property was also recognized by George Chaikin when he rediscovered the algorithm in 1974 [19]. Moreover, he was able to prove that the resulting curves correspond to quadratic B-splines ($p = 2$). These are particularly suitable for computer-aided design. For a Bézier curve, the change of a control point influences the shape of the whole curve. B-spline have local support, which means that changing a control point only affects a subdomain of the curve. This makes deforming the curve by moving the control points more intuitive.

Lane-Riesenfeld's algorithm

Lane-Riesenfeld's algorithm [20] generates uniform B-splines of arbitrary polynomial degree p. The refinement of the control polygon with $n \geq p + 1$ control points is done in two stages. In the first stage, the control points are duplicated. In the second stage, the segments of the control polygon are subdivided in the ratio $(1 : 1)$ and connected p times. The two stages are repeated and converge to a B-spline of degree p, as shown in Figure 4.6.

De Boor's algorithm

De Boor's algorithm introduces the knot vector and results in a nonuniform B-spline (NUBS) [21]. This allows a distortion of the parameter space, which does not have to be normalized anymore. The knot vector contains a non-descending list of scalar values which are called knots. The distortion of the parameter space at



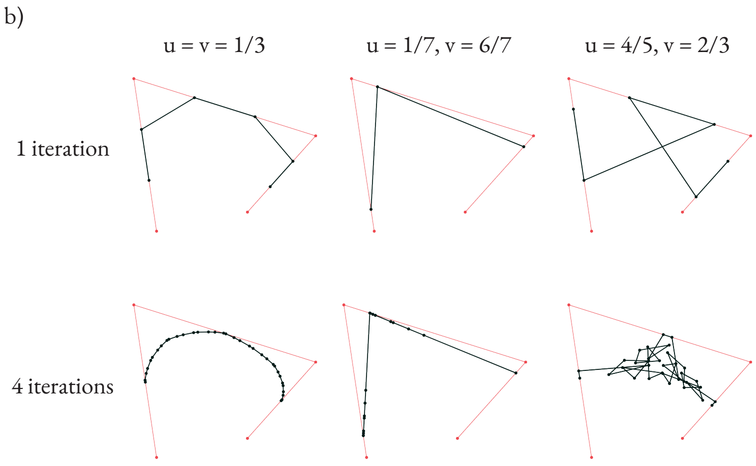


Figure 4.5: De Rham-Chaikin algorithm with a different subdivision ratio. The ratio in (a) results in a quadratic B-spline. Other ratios produce, e.g. C1 continuous curves, a shifted polyline or fractals (b).

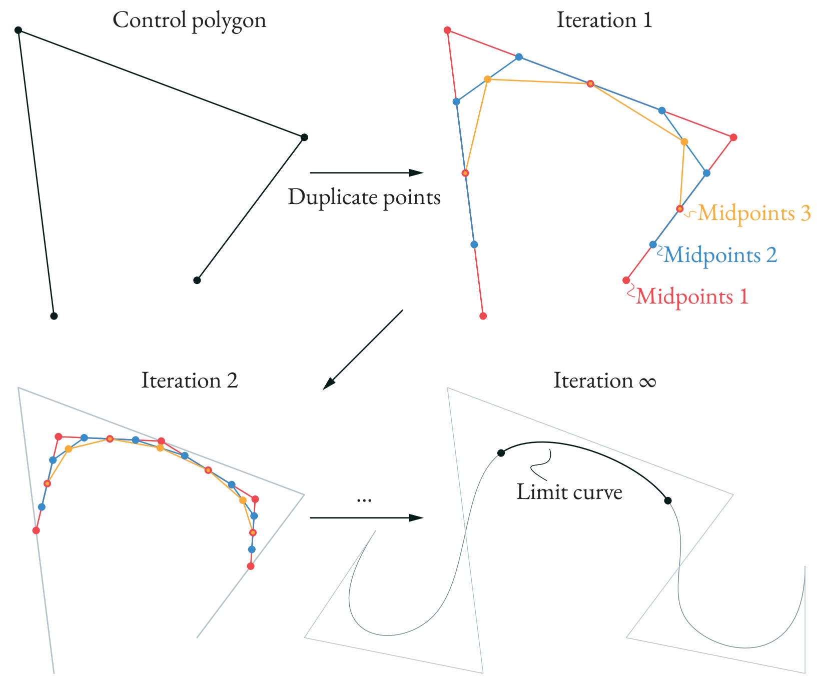


Figure 4.6: Lane-Riesenfeld's algorithm for a polynomial degree $p = 3$. At the beginning of each subdivision iteration, the control points are duplicated. Then the midpoints of the control polygon segments are determined and connected p times.

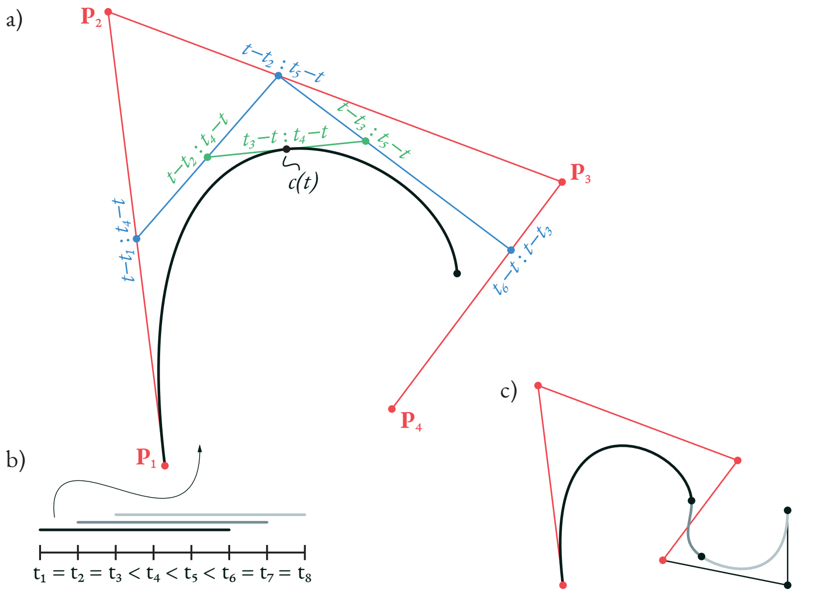


Figure 4.7: De Boor's algorithm for generating a nonuniform Bézier segment of degree $p = 3$ (a). An additional knot vector (b) defines the subdivision ratio. $2p$ knots define a span. Multiple spans are connected to a NUBS (c). Knots with a multiplicity of p result in the interpolation of the control points at both ends.

parameter t results from the $2p$ surrounding knots where p is again the polynomial degree of the spline.

Similar to De Casteljau's algorithm, the segments of a control polygon are divided. However, the subdivision ratio is now also influenced by the knots (Figure 4.7). Note that the outer control points are generally no longer interpolated. To achieve interpolation of a control point, a node in the knot vector must be repeated p times. Multiplicities generated in this way can lead to discontinuities within the curve.

Similar to Bézier curves, several such segments can be lined up. Nodes and control points must overlap accordingly. In this way, a nonuniform B-splines (NUBS) is created. Classic B-splines are a special form with evenly distributed nodes.

NURBS

Nonuniform rational B-splines (NURBS) extend NUBS by applying weights to the control points [22, 23]. The weights can be interpreted as abstract attracting forces pushing the curve towards the control points. The principle can be illustrated by considering the weights as an additional dimension. Figure 4.8 shows a planar NURBS curve with different weights. The same curve is shown in three-dimensional space where the weights define the third dimension. A NUBS can be constructed from the spatial control polygon. The corresponding planar NURBS curve results from the central projection of the NUBS. A change in the weights corresponds to the shifting of the control points in the higher dimensional space.

The visualization as a central projection also clarifies that the ratio of the individual weights determines the shape of the NURBS. Scaling all weights by the same factor has no impact on the form.

NURBS are widely used in CAD since they can represent cone sections exactly. This property is essential for technical drawings.

4.2 Freeform surfaces

Freeform surfaces can be defined as tensor products of freeform curves. The geometry is defined by a regular control mesh, which consists of $n \times m$ control points, which are connected by $(n-1) \times (m-1)$ quads, as shown in Figure 4.9. The n rows and m columns of the control mesh define control polygons which can be used, e.g. for the definition of Bézier, B-spline or NURBS curves. Evaluating all rows for the parameter u results in n points. These define a control polygon of a curve, which can be evaluated for the parameter v. The resulting point corresponds to the smooth limit surface at the parameter pair $(u, v) \in \mathbb{R}^2$.

Evaluating the surfaces for a rectangular domain $u, v \in [0, 1]$ results in a quadrilateral shape. To create more complex models, the surfaces might be evaluated only for a subset Ω of the parameter space. This is then called a trimmed surface, where the domain is usually defined by its boundaries in the parameter space. Multiple trimmed surface patches can be joined together as a trimmed multipatch model, as shown in Figure 4.10. It should be noted that in practice, the surfaces might not match perfectly at the edges (watertightness) but lie within a certain tolerance. During rendering, imperfections along coupling edges are usually hidden by the CAD.

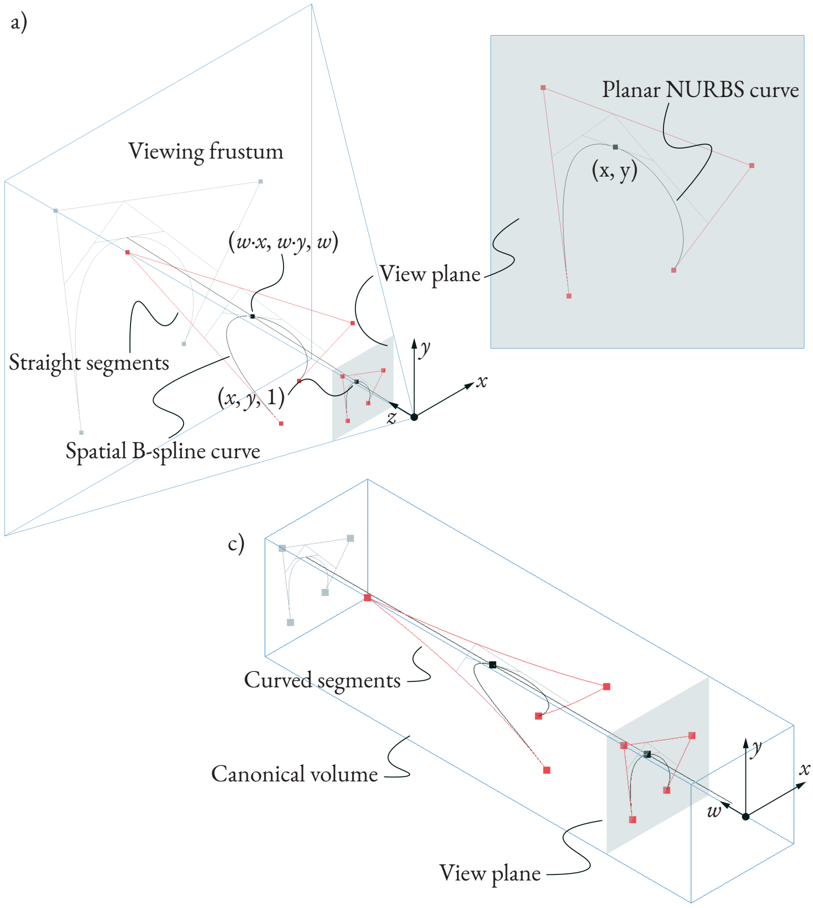


Figure 4.8: An n-dimensional NURBS can be interpreted as a central projection of an $(n+1)$-dimensional B-spline. The example shows a two-dimensional NURBS in the grey plane. The weights correspond to a shift of the control points in an additional third dimension. On the left, the situation is shown over the 'viewing frustum' (3D control polygon as linear segments), on the right in the canonical volume (3D control polygon as distorted segments).

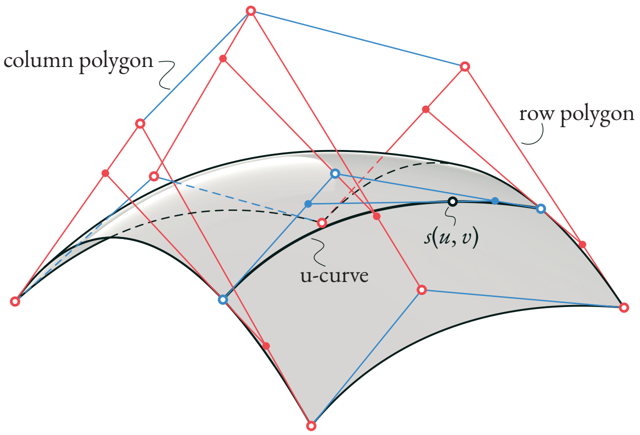


Figure 4.9: A Bézier surface is defined by a regular control mesh. Each row and column of the mesh defines the control polygon of a Bézier curve (adapted from [17]).

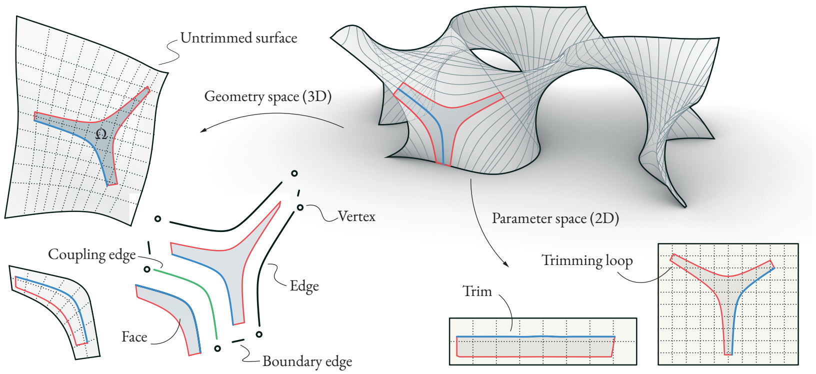


Figure 4.10: A trimmed multipatch model consists of several faces. The geometry of each face is defined by an untrimmed surface and a trimming domain Ω. The untrimmed surface might be a classic NURBS surface. The domain is defined by a trimming loop in the parameter space. Each segment of this loop belongs to an edge of the face. Boundary edges only have single trims. Edges with several trims are defined as coupling edges (adapted from [24]).

Subdivision surfaces are another type of freeform surface. The control mesh of these surfaces can have an arbitrary topology. This avoids the need for trimming. At the same time, sharp edges and corners can be defined specifically. This type of surface description is described in more detail in Chapter 6.

4.3 Sensitivities of freeform geometries

The freeform geometries described in the previous sections correspond to an explicit parameterization. Based on the given control points, the smooth geometry can be evaluated for a specific curve or surface parameter. To solve inverse problems based on such geometries, the derivatives of this point with respect to the design parameters are required.

This section explains the systematic computation of the derivatives for a Bézier curve by using AD. The procedure can be transferred to other types of free-form geometries. For this purpose, the procedure of the algorithm is shown in a graph. Fig 15 shows the decomposition of the algorithm into basic mathematical functions. The derivatives can then be determined from this graph. Since linear interpolation between two points is frequently required during the calculation, it is recommended that a dedicated operation is defined for this purpose, including derivation rules. This simplifies the graph and speeds up the calculation. For this purpose, a function

$$\operatorname{lerp}(a, b, t) = a + t\,(b - a) \tag{4.1}$$

is introduced, which interpolates linearly between two values, a and b, where $t \in [0, 1]$ is the interpolation parameter. The partial derivatives of this interpolation with respect to the three arguments are given by:

$$D_a \operatorname{lerp} = 1 - t \tag{4.2}$$

$$D_b \operatorname{lerp} = t \tag{4.3}$$

$$D_t \operatorname{lerp} = b - a \tag{4.4}$$

These are the required coefficients for the algorithms in Chapter 3.

A point on the Bézier curve can be constructed by three stages of linear interpolation, as shown in Figure 4.11. According to Chapter 3, the partial derivatives of the linear interpolation are used to create the weighted computational graph in Figure 4.12.

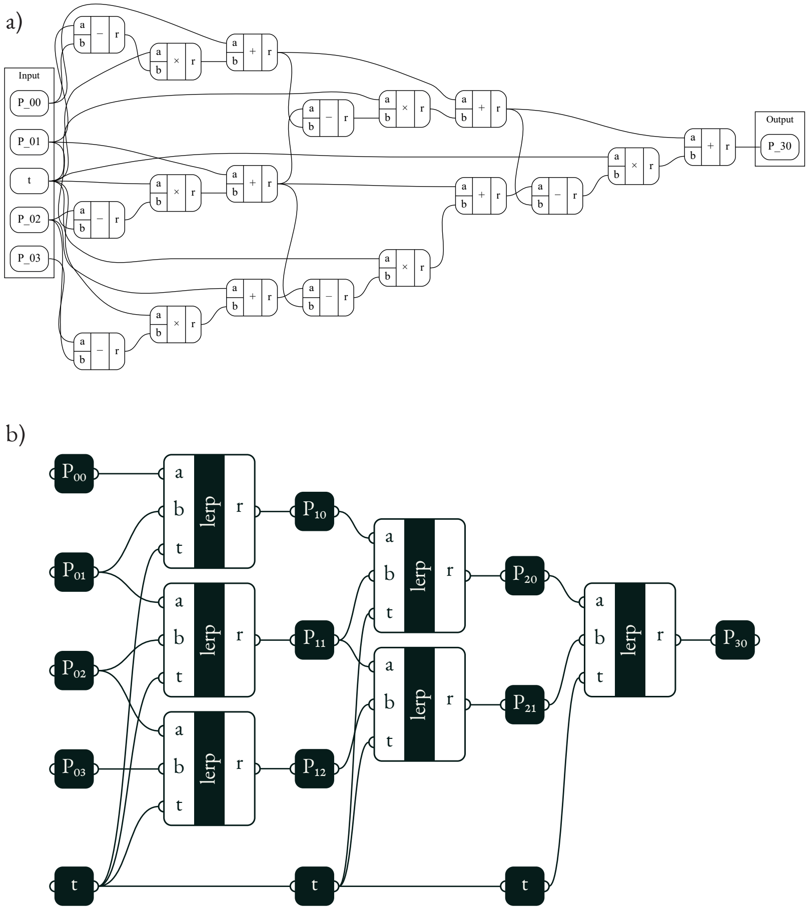


Figure 4.11: Computational graph for the computation of a cubic Bézier curve. In (a), the computation is decomposed into basic mathematical operations. By introducing a dedicated operation for linear interpolation in (b), the decomposition is simplified.

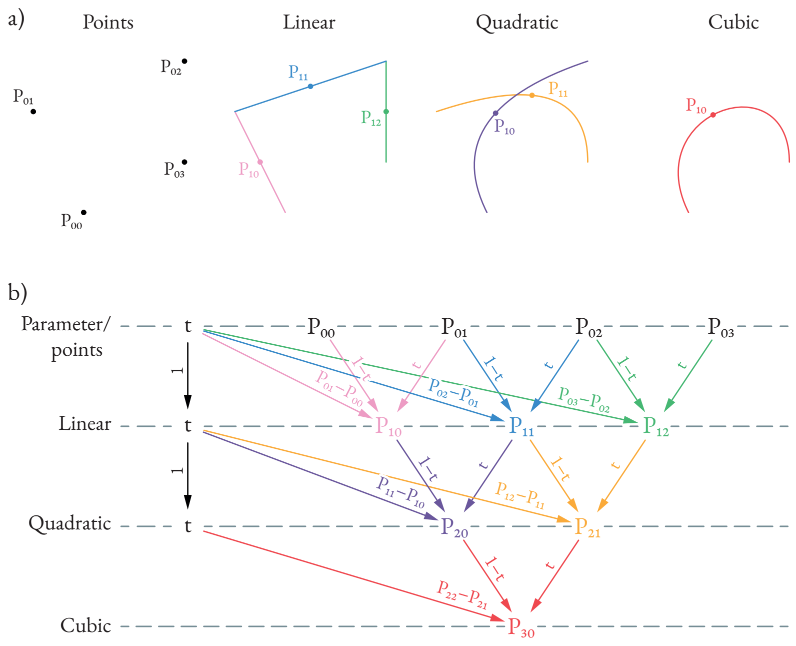


Figure 4.12: (a) Steps to construct a point on a cubic Bézier curve. (b) Weighted computational graph for the computation.

The derivatives of $X = P_{30}$ with respect to the positions of the control points result from the weights of the corresponding paths. They result in the shape functions N_i:

$$D_{P00}\,\mathbf{X} = t^3 = N_0(t) \tag{4.5}$$

$$D_{P01}\,\mathbf{X} = 3t^2\,(1-t) = N_1(t) \tag{4.6}$$

$$D_{P02}\,\mathbf{X} = 3t^2\,(1-t) = N_2(t) \tag{4.7}$$

$$D_{P03}\,\mathbf{X} = t^3 = N_3(t) \tag{4.8}$$

The point $\mathbf{X}$ of the Bézier curve can also be formed from the linear combination of the shape functions:

$$\mathbf{X} = \sum_{i}^{n} N_i \, \mathbf{P}_{0i} \tag{4.9}$$

The derivative of X with respect to the curve parameter t also results from the corresponding weights and provides the prescription for the basis vector $\mathbf{a}_1$:

$$D_t \, \mathbf{X} = (\mathbf{P}_{20} - \mathbf{P}_{21}) + t\,(\mathbf{P}_{10} - \mathbf{P}_{11}) + (1-t)\,(\mathbf{P}_{11} - \mathbf{P}_{12}) + \tag{4.10}$$

$$t^2\,(\mathbf{P}_{00} - \mathbf{P}_{01}) + 2t\,(1-t)(\mathbf{P}_{01} - \mathbf{P}_{02}) + \tag{4.11}$$

$$(1-t)^2\,(\mathbf{P}_{02} - \mathbf{P}_{03}) = \mathbf{a}_1(t) \tag{4.12}$$

The computation of the derivatives is done according to the principle of AD, as described in Chapter 3. AD is applied to the algorithm, which evaluates the Bézier curve by tracking the construction process in a computational graph. The structure of this graph and the calculation of the weights for this type of curve follows a certain scheme. An extensive AD library can optimize the computation of the derivatives by applying the transformation rules in a smart way. In a perfect scenario, this would yield the very efficient algorithms from [22]. Since these efficient algorithms are already known, they can also be implemented directly to determine the weights of the paths. This results in a new function which evaluates the curve and its derivative and can be combined as a new building block within AD.

In the same way, building blocks are defined in Chapter 5 for isogeometric analysis to apply the principles of AD in an efficient way. In Chapter 6, AD is used to compute the shape functions of SubD surfaces. In this case, the computation turns out to be more complex since the control mesh can have an arbitrary topology. AD is a useful tool for simplifying the data management in such situations.

5

Application in isogeometric analysis

The content of this chapter was previously published as: Oberbichler, T., Wüchner, R., Bletzinger, K.-U.: Efficient computation of nonlinear isogeometric elements using the adjoint method and algorithmic differentiation. *Computer Methods in Applied Mechanics and Engineering*, 2021, https://doi.org/10.1016/j.cma.2021.113817

Acknowledgments

We present a consistent and efficient approach to the formulation of geometric nonlinear finite elements for *isogeometric analysis* (*IGA*) and *isogeometric B-Rep analysis* (*IBRA*) based on the adjoint method. IGA elements are computationally expensive, especially for high polynomial degrees. Using the method presented here enables us to reduce this disadvantage and develop a methodical framework for the efficient implementation of IGA elements. The elements are consistently derived from energy functionals. The load vector and stiffness matrix are obtained from the first and second order derivatives of the energy. Starting from the functional, we apply the concept of *algorithmic* or *automatic differentiation* to compute the precise derivatives. Here, we compare the direct (forward) and adjoint (reversed) methods. Analysis of the computational graph allows us to optimize the computation and identify recurring modules. It turns out that using the adjoint method leads to a *core-congruential formulation*, which enables a clean separation between the mechanical behavior and the geometric description. This is particularly useful in CAD-integrated analysis, where mechanical properties are applied to different geometry types. The adjoint method produces the same results but requires significally fewer operations and fewer intermediate results. Moreover, the number of intermediate results is no longer dependent on the polynomial degree of the NURBS. This is important for implementation efficiency and computation speed.

The procedure can be applied to arbitrary element formulations and coupling conditions based on energy functionals. For demonstration purposes, we present the proposed approach specifically for use with geometrically nonlinear trusses, beams, membranes, shells, and coupling conditions based on the penalty method.

5.1 Introduction

Ever since *isogeometric analysis* (*IGA*) was first introduced by Hughes in 2005 [25, 26], the method has been continuously expanded. Today, there are a wide range of isogeometric mechanical elements available. These include cables, trusses, and membranes [27] as well as beams [28] and shells [29, 30, 31]. *Isogeometric B-Rep analysis* (*IBRA*) extends the methodology to trimmed multipatches. The introduction of weak forms of geometric coupling conditions addresses the problem of non-interpolating control points. Various types of coupling conditions have been

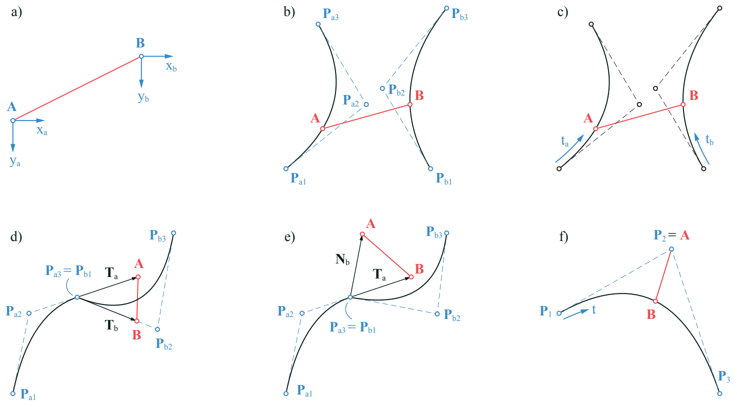


Figure 5.1: Different scenarios in which the aim is to minimize the distance between two points **A** and **B** by a coupling constraint (red). The difference lies in the parameterization of the two points. This results in different control parameters (blue): (a) The locations are explicitly given by the coordinates x and y; (b) The points are embedded in NURBS curves. The locations of **A** and **B** are controlled by the location of the control points $\mathbf{P}_i$; (c) **A** and **B** can slide along the NURBS curves. The location is controlled by the curve parameters t_a and t_b; (d) The distance between the tangents $\mathbf{T}_a$ and $\mathbf{T}_b$ is minimized to achieve G1 continuity. The geometry is controlled by $\mathbf{P}_i$; (e) The distance between the tangent $\mathbf{T}_a$ and the normal $\mathbf{N}_b$ is minimized by controlling $\mathbf{P}_i$; (f) **A** corresponds to a control point, while **B** slides on the curve. The distance is controlled by $\mathbf{P}_i$ and t.

formulated, including the Penalty [32, 33, 34], Lagrange [35, 36], Nitsche [37, 38, 39, 40, 41, 42, 43], and Mortar methods [44, 35, 45]. In addition to classic NURBS, embedded geometries can also be used to describe the shape of a structural entity. They can be used to describe edge cables for membrane structures [27], to enforce shells with beam elements [46] or to realize sliding conditions [47].

Manual implementation of all these elements requires considerable effort. Embedding an element does not change its basic mechanical behavior but results in a new element formulation. A similar problem arises with coupling conditions. Figure 5.1 shows different coupling scenarios. In all the examples, the aim is to minimize the distance between two points. However, in each case, the points are described differently, which results in varying formulations. The same applies when using different coupling methods.

We introduce a modular and computationally efficient method of formulating isogeometric elements and coupling conditions. We do this using an energy functional, which can be formulated as a simple scalar function $\Pi(\mathbf{x})$. The load vector $\mathbf{F}$ and stiffness matrix $\mathbf{K}$ are obtained from the derivatives of the energy with respect to the degrees of freedom $\mathbf{x}$. To compute these derivatives, we decompose the functional into basic operations and consistently apply the chain rule. The chain rule can be applied in two different ways, resulting in the *direct method* and the *adjoint method* [48]. Both produce identical results but may require a different number of operations. The direct method is the one usually used for implementing IGA. In Section 5.5, we compare the two methods and show that the adjoint method significantly reduces computational effort and memory consumption. Moreover, the adjoint method leads to a *core-congruential formulation* (CCF) [49, 50], which enables the mechanical behavior $\hat{\mathbf{\Pi}}$ to be separated from the geometrical description $\mathbf{g}$ by decomposing Π into:

$$\Pi(\mathbf{x}) = (\hat{\Pi} \circ \mathbf{g})(\mathbf{x}).$$

The derivatives of $\hat{\mathbf{\Pi}}$ are interpreted mechanically as the core load $\hat{\mathbf{F}}$ and core stiffness $\hat{\mathbf{K}}$. Both are independent of the geometrical description. Therefore, they are not influenced by the polynomial degree of the NURBS and can be computed very efficiently. By applying the chain rule, we transform $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ to $\mathbf{F}$ and $\mathbf{K}$ according to the geometric description given by $\mathbf{g}$. This separation is a key concept of CCF and has special advantages in the context of IGA. If the geometry is described using NURBS, the complexity of this operation depends on the polynomial degree. With the direct method, this affects all intermediate results. Since the adjoint method leads to CCF, the NURBS description is only used for this final transformation. This minimizes the influence of the higher polynomial degrees. Furthermore, different element types can be created by transforming the same $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ for different geometric descriptions $\mathbf{g}$. For example, it is shown in Section 5.8 that all coupling scenarios represented in Figure 5.1 can be realized by transforming the same $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ according to the specific geometric representation.

As mentioned above, we decompose the functional into simpler suboperations to determine the derivatives using the chain rule. This process can be generalized and automated using *algorithmic differentiation* (AD), also called *automatic differentiation*. Using this technique, decomposition can be done down to the basic mathematical operations. It can easily be applied to an arbitrary energy functional and is therefore suitable for rapid implementation and testing of new element types. A common way to implement AD is by using *hyper-dual* numbers [14]. This leads to the so-called *forward mode* of AD. The authors presented this approach for implementing isogeometric shell and beam elements based on NURBS geometries[1]. A similar approach was used in [51] for the implementation of classic solid finite elements. The forward mode of AD represents an automation of the direct method. It is less efficient for implementing isogeometric finite elements then the adjoint method. For an even more efficient implementation, we analyze the computational graph to calculate the derivatives. This allows us to identify recurring operations and simplifies the decomposition of the functional. It also enables a mechanical interpretation of individual components and decomposes the energy functional down to the basic mechanical operations. This manual approach can be combined with *algorithmic differentiation* using hyper-dual numbers to calculate branches of the graph that are too complex for manual optimization.

This article is divided into ten sections. Section 5.2 describes the relationship between the derivatives of the energy functional and the equilibrium of a structure. It is in this context that the equality of the energy-based approach and the principle of virtual work are presented. The relevant principles of NURBS, which are used to describe the geometry of IGA elements, are recalled in Section 5.3. In Section 5.4, we explain the methodical procedure of applying the chain rule to complex functions. Section 5.5 compares the direct and adjoint methods of formulating IGA elements. In Section 5.6, 5.7, and 5.8, we formulate the structural elements, geometric transformations, and coupling conditions according to the adjoint approach. Section 5.9 presents a comparison with the established research software *Carat*++ [52]. Finally, we discuss the results in Section 5.10 and provide an outlook on future research.

5.2 Finding Equilibrium — The Role of Derivatives

A structure is in stable equilibrium when it reaches a state of minimum energy. A structural analysis using the *finite element method* (FEM) requires us to formulate the energy Π of the system as a function of the nodal coordinates $\mathbf{x}$. By determining $\mathbf{x}$ such that the energy is minimized, we obtain a state of equilibrium.

[1]T. Oberbichler, R. Wüchner, K.-U. Bletzinger, Benefits of Automatic differentiation in Finite Element Analysis (2019). The 5th ECCOMAS Young Investigators Conference. Krakow, Poland.

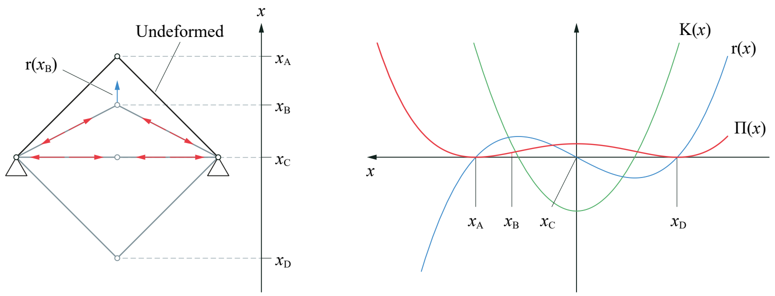


Figure 5.2: Internal energy Π of a simple two-bar truss in relation to x. The residual force r and tangential stiffness K are used to indentify points of equilibrium.

$$\min_{\mathbf{x}} \ \Pi(\mathbf{x}) \xrightarrow{\text{yields}} \text{stable equilibium} \tag{5.1}$$

Figure 5.2 shows the internal energy in a simple two-bar truss. To identify a local minimum, the derivatives must be considered. The first two derivatives of the energy functional can be interpreted in a mechanical sense as the *residual forces* $\mathbf{r}(\mathbf{x})$ and the *tangential stiffness* $\mathbf{K}(\mathbf{x})$.

Residual Forces

To obtain the residual forces, we compute the total differential of the energy functional $\mathrm{d}\Pi$. We use *Einstein notation* to simplify sums. The components x_r of a vector $\mathbf{x} = (x_r) = (x_1, x_2 \ldots x_{|x|-1}, x_{|x|})$ are numbered with an index r, while its dimension is denoted by $|x|$. The displacements $\mathbf{u} = \mathbf{x} - \mathbf{X}$ of the nodes are determined by their locations in the deformed configuration $\mathbf{x}$ and the undeformed configuration $\mathbf{X}$. Since $\mathbf{X}$ is constant, the derivative of Π with respect to a coordinate x_r is equal to the derivative with respect to the corresponding displacement u_r.

$$\mathrm{d}\Pi = \sum_{r=1}^{|x|} \frac{\partial \Pi}{\partial x_r}\,\mathrm{d}x_r = \frac{\partial \Pi}{\partial u_r}\,\mathrm{d}u_r = \frac{\partial \Pi}{\partial x_r}\,\mathrm{d}x_r \tag{5.2}$$

It turns out that $\mathrm{d}\Pi$ is equivalent to the virtual work δW, which gives us the analogy of the minimum energy state, and the *principle of virtual work* and shows that $\partial\Pi/\partial x_r$ is equal to the force F_r.

$$\mathrm{d}\Pi = \frac{\partial\Pi}{\partial x_r}\,\mathrm{d}x_r \equiv F_r\,\delta u_r = \delta W \tag{5.3}$$

The partial derivative of Π with respect to a coordinate x_r indicates the residual force F_r in the corresponding direction. The vector $\mathbf{r}$ contains all residual forces and is therefore called the *residual force vector*.

$$\mathbf{r} = \left(\frac{\partial\Pi}{\partial x_r}\right) = \left(\frac{\partial\,\Pi}{\partial\,x_1}, \frac{\partial\,\Pi}{\partial\,x_2} \cdots \frac{\partial\,\Pi}{\partial\,x_{|x|}}\right) \tag{5.4}$$

A local extremum is characterized by $\mathbf{r}(\mathbf{x}) = \mathbf{0}$, which means that there are no residual forces. For the example in Figure 5.2, we can identify the extrema x_A, x_C and x_D.

Tangential Stiffness Matrix

By deriving $\mathbf{r}$ with respect to the nodal coordinates x_r, we obtain the *tangential stiffness matrix* $\mathbf{K}$ of the structural system. This is equivalent to the second order derivatives of Π.

$$\mathbf{K} = \left(\frac{\partial\Pi}{\partial x_r\,\partial x_s}\right) = \begin{pmatrix} \frac{\partial\Pi}{\partial x_1\,\partial x_1} & \frac{\partial\Pi}{\partial x_1\,\partial x_2} & \cdots & \frac{\partial\Pi}{\partial x_1\,\partial x_{|x|}} \\ & \frac{\partial\Pi}{\partial x_2\,\partial x_2} & \cdots & \frac{\partial\Pi}{\partial x_2\,\partial x_{|x|}} \\ & & \ddots & \vdots \\ \text{sym.} & & & \frac{\partial\Pi}{\partial x_{|x|}\,\partial x_{|x|}} \end{pmatrix} \tag{5.5}$$

The definiteness of $\mathbf{K}$ supplies information about the stability of the structure. A positive definite stiffness matrix characterizes a point of stable equilibrium. Considering this relation for the example shown in Figure 5.2, we can identify stable equilibrium points at x_A and x_D while x_C is a state of unstable equilibrium.

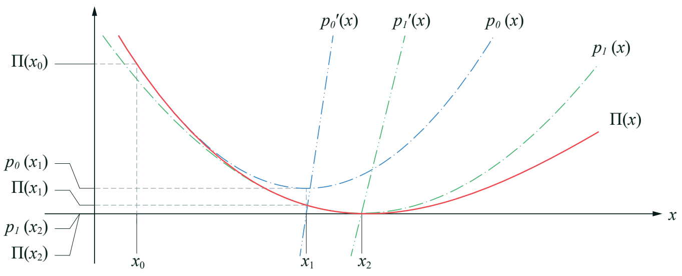


Figure 5.3: The Newton-Raphson algorithm is used to calculate the minimum of $\Pi(x)$. In each iteration i, the function Π is approximated by a parabola p_i at x_i. The minimum of p_i yields to x_{i+1}, which is used for the next iteration.

Newton-Raphson Solver

Line-search algorithms have been established to locate a solution close to a given initial state. The Newton-Raphson algorithm is widely used in structural mechanics and can be applied to a function with multiple parameters. For simplicity, we assume a single parameter x. The method can be used to determine the roots of $r(x) = 0$ iteratively, by applying the following iteration scheme:

$$x_{i+1} = x_i - \frac{r(x_i)}{r'(x_i)} \tag{5.6}$$

Similarly, the Newton-Raphson algorithm can be used to determine the minimum of $\Pi(x)$. In each iteration i, the function $\Pi(x)$ is approximated by a parabola

$$p_i(x) = \Pi(x_i) + \Pi'(x_i)\,(x - x_i) + \frac{1}{2}\,\Pi''(x_i)\,(x - x_i)^2. \tag{5.7}$$

The extrema of p_i can be found at the intersection of p_i' and the abscissa and yields to x_{i+1} (Figure 5.3). For this, we need the first two derivatives of Π.

$$x_{i+1} = x_i - \frac{\Pi'(x_i)}{\Pi''(x_i)} \tag{5.8}$$

The sequence $(x_0, x_1, \dots \hat{x})$ should converge to a local extremum $\Pi(\hat{x})$. The procedure in (5.6) is usually illustrated by the repeated intersection of tangents of r with the abscissa. Similarly, (5.8) can be shown as the repeated minimization of a parabola (Figure 5.3). Note that (5.6) and (5.8) are equal if $\Pi' = r$ and $\Pi'' = r' = K$. This confirms the analogy between minimum energy and equilibrium of forces.

Any other optimization or root-finding algorithm can be used to determine a state of equilibrium. An overview of numerical approaches can be found in [7]. Some of these algorithms omit the stiffness matrix (e.g. *steepest-descent*, *conjugate gradient*) or use an approximation (quasi-Newton methods, e.g., *L-BFGS*). Another method of determining equilibrium is to use the mass matrix in place of stiffness, which results into *dynamic relaxation* (DR). The mass matrix is generally approximated by a diagonal matrix. Details of the relation between DR and the Newton-Raphson method can be found in [53]. For structural analysis, it is necessary to use the precise stiffness matrix, as it generally improves the convergence rate of the solution and provides essential information about the stability of the structure.

Finite Elements

The total energy of the system results from the sum of the energies of the single elements. It allows element-wise calculation of the residual force and tangential stiffness. The contribution of an element solely depends on a subset $\mathbf{x}^{(e)}$ of $\mathbf{x}$, which enables efficient and parallel computation.

The energy functional is chosen with respect to the element type. The energy contribution $\Pi^{(e)}$ of a single element is obtained by integrating this functional over the domain of the element. This domain results from the shape of the element, e.g. the area of a trimmed NURBS patch. To perform a numerical calculation, the integral is approximated with a weighted sum using an n-point Gaussian quadrature rule.

$$\Pi^{(e)} = \int_{\Omega} \Pi^{(e)}(\omega)\, \mathrm{d}\Omega \approx \sum_{i=1}^{n} \Pi^{(e)}(\omega_i)\, \alpha_i \tag{5.9}$$

Here ω is a point within the domain Ω of the element, ω_i is the position of an integration point, and α_i is the corresponding weight in the geometry space. The energy of the element results from the sum of the contributions of the individual integration points. Therefore, it is necessary to calculate the internal energy at specific points of the structure and evaluate the derivatives with respect to the degrees of freedom **x**.

Penalty Constraints

For an analysis of structures modeled by trimmed multipatches, the optimization problem is extended by nonlinear equality constraints g_i. This yields to the constrained optimization problem:

$$\min_{\mathbf{x}} \Pi(\mathbf{x}) \qquad \text{s.t. } \mathbf{g}(\mathbf{x}) = \mathbf{0} \tag{5.10}$$

To solve this problem, we use the penalty method and add the constraints to the objective function by using a quadratic penalty function

$$P_i(\mathbf{x}) = \frac{w_i}{2} g_i(\mathbf{x})^2, \tag{5.11}$$

where w_i denotes the weight of the constraint. This converts the constrained optimization problem (5.10) into an unconstrained one (5.12), which can be solved using the classical Newton-Raphson algorithm, for example.

$$\min_{\mathbf{x}} \left(\Pi(\mathbf{x}) + \frac{1}{2} \sum_{i=1}^{|g|} w_i \cdot g_i(\mathbf{x})^2 \right) \tag{5.12}$$

If not all conditions can be fulfilled, the procedure results in a least-square solution according to the weights ratio.

5.3 Fundamentals of NURBS

A NURBS curve in d-dimensional space is defined by the *polynomial degree* p, the *knot vector* $\boldsymbol{\Xi}$ and the *control points* $\mathbf{P}_i$, where each point is associated with a weight w_i. The parametric representation of the curve is given by

$$\mathbf{c}(\xi) = \sum_{i=1}^{|P|} R_i(\xi)\,\mathbf{P}_i \qquad \in \mathbb{R}^d, \tag{5.13}$$

where ξ is the curve parameter and R_i is the weighted basis function, which depends on p, $\boldsymbol{\Xi}$ and $\mathbf{w}$. The knots Ξ_p and $\Xi_{|P|}$ define the domain of the curve and constrain the curve parameter $\Xi_p \leq \xi \leq \Xi_{|P|}$. The knot vector defines the parameter space of the NURBS and divides it into spans. It contains the knots in non-descending order.

$$\boldsymbol{\Xi} = \begin{pmatrix} \Xi_0 & \Xi_1 & \Xi_2 & \dots & \Xi_{|P|+p-1} & \Xi_{|P|+p} \end{pmatrix} \tag{5.14}$$

Note that, in this case, we start with an index 0. This is because the knots Ξ_0 and $\Xi_{|P|+p}$ are never used to describe a continuous geometry. Omitting them yields the reduced knot vector $(\Xi_1, \dots \Xi_{|P|+p-1})$, which facilitates consistent handling of the knot spans. The location of $\mathbf{c}(\xi)$ only depends on $p+1$ control points because

$$R_i(\xi) \begin{cases} \neq 0 & s \leq i \leq s+p \qquad \text{where } \Xi_{s+p-1} \leq \xi < \Xi_{s+p} \\ = 0 & \text{otherwise} \end{cases}. \tag{5.15}$$

Here, s denotes the knot span of ξ. Within the same knot span, all points are influenced by the same control points $\bar{\mathbf{P}}$. Furthermore, only the knots $\bar{\boldsymbol{\Xi}} = (\Xi_s \dots \Xi_{s+2p-1})$ are involved (Figure 5.4). Therefore, isogeometric finite elements are usually defined according to the knot spans:

$$\mathbf{c}(\xi) = \sum_{i=1}^{p+1} \bar{R}_i(\xi)\,\bar{\mathbf{P}}_i \qquad \text{with } \bar{R}_i = R_{s+i-1} \text{ and } \bar{\mathbf{P}}_i = \mathbf{P}_{s+i-1} \tag{5.16}$$

When it comes to the continuum mechanical derivation of finite elements, the basis vectors of the geometry are of interest. These are defined as derivatives of (5.16) with respect to the parameter ξ. It is therefore sufficient to compute the derivatives of the basis functions denoted as $\bar{R}_{i,1}$ and $\bar{R}_{i,1,1}$ according to [22]:

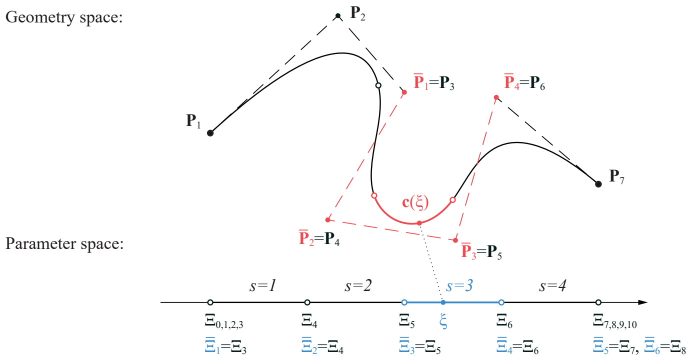


Figure 5.4: A NURBS curve defined by the polynomial degree $p = 3$, the control points $\mathbf{P}_i$ and the knots Ξ_i. Within the third knot span ($s = 3$), only the control points $\bar{\mathbf{P}}_i$ (red) and knots $\bar{\Xi}_i$ (blue) have an influence on the geometry.

$$\frac{\mathrm{d}}{\mathrm{d}\xi}\,\mathbf{c}(\xi) = \mathbf{a}_1(\xi) = \sum_{i=1}^{p+1} \bar{R}_{i,1}(\xi)\,\bar{\mathbf{P}}_i \tag{5.17}$$

$$\frac{\mathrm{d}^2}{\mathrm{d}\xi^2}\,\mathbf{c}(\xi) = \mathbf{a}_{1,1}(\xi) = \sum_{i=1}^{p+1} \bar{R}_{i,1,1}(\xi)\,\bar{\mathbf{P}}_i \tag{5.18}$$

These concepts can be applied directly to NURBS surfaces. In this case, we have two surface parameters ξ and η with two knot vectors $\boldsymbol{\Xi}$ and $\mathbf{H}$. A knot span corresponds to a rectangle in the parameter space. Also, for surfaces, the geometry within a span is only influenced by a subset of the control points. Extension to volume is performed analogously. Further details on NURBS can be found in [22] and [23].

5.4 Methods of Computing Derivatives

Derivatives describe the effect of variations in *design parameters* (input) on the *response* (output) of a mathematical model. We assume that the responses are collected within the vector $\mathbf{f} = (f_k)$ and are described by smooth functions of the design parameters collected in $\mathbf{s} = (s_i)$. Mathematically, the derivatives are defined as a limit of difference quotients:

$$\frac{\mathrm{d} f_k}{\mathrm{d} s_i} = \lim_{\varepsilon \to 0} \frac{f_k(s_i + \varepsilon) - f_k(s_i)}{\varepsilon} \tag{5.19}$$

There are three methods of calculating such derivatives: (i) Numeric differentiation or finite differences, (ii) symbolic or analytic differentiation, and (iii) algorithmic or automatic differentiation. We now present a short overview of the different methods. More detailed descriptions can be found in [7].

Numeric Differentiation (ND) An approximation of the derivatives is obtained using difference quotients. For *forward differences*, ε is replaced by a small but finite step size h. A too-small step size leads to errors in floating-point precision, while a too-high value results in an inaccurate approximation. One way of improving this problem is to use *central differences* or *Ridders' method* [54]. ND is easy to implement since it only necessitates evaluating the function for different input parameters. This makes the computation very slow, with an increasing number of parameters. ND is therefore not suitable for an efficient and exact calculation of derivatives in the context of this study.

Symbolic Differentiation (SD) The derivatives are obtained by applying the basic rules of derivation on symbolic expressions. This can be done manually or with the aid of a *computer algebra system* (CAS) e.g. *Mathematica*, *SymPy* or *Maple*. The result is an analytical expression of the derivatives, which is then implemented in the FE code. Evaluating the expression for specific parameters provides exact results. By representing the derivation as a single expression, SD can inflate the final analytic solution. This makes the analytical expressions so complex that they can barely be interpreted or processed.

Algorithmic Differentiation (AD) A complex function is decomposed into simpler suboperations. The partial derivatives of the suboperation are computed for specific input parameters. The derivatives of the complex function are obtained by applying the chain rule. AD is heavily oriented towards computer implementation and automation. It is based on the fact that a complex algorithm is a composition of simple operations, and the computer is fed the structure of the composition from the program code. In contrast to SD, the results are numeric rather than symbolic. This approach gives precise results and works efficiently, even with complex functions.

It should be noted that AD is often mistakenly equated with SD and the use of a CAS. Unlike ND, AD and SD do not provide approximations but precise results. In this sense, the two methods are similar. The main difference is that SD aims to generate a symbolic expression of the derivative. In contrast, the goal of AD is to immediately evaluate the derivations at a specific position. Therefore, the essential derivation rules are implemented so that the computer can evaluate the derivatives of an arbitrary function. In this sense, algorithmic differentiation is "Symbolic differentiation performed by a compiler" [55]. Both methods lead to the same result. However, AD does not need to generate a human-readable analytical expression for the derivatives. This allows AD to work very efficiently.

In this paper, we use AD for the computation of derivatives. Detailed descriptions of AD can be found in [56, 57, 58, 55, 59, 60]. AD offers two distinct methods: the *forward mode* (FAD) and the *reverse mode* (RAD). They directly correspond to the *direct method* (DM) and the *adjoint method* (AM) of computing derivatives. FAD/DM and RAD/AM are based on the consistent application of the chain rule. The difference lies only in the order in which the rule is applied. Both methods can also be combined into a *mixed mode*. In this paper we focus on the basic methodology of AD in a systematic application of the chain rule to implement isogeometric elements. Note that there are a wide range of advanced implementations [61, 62, 63] that might also be suitable for other applications e.g. gradient based shape optimization of IGA models, as shown in [64]. In the following sections, we describe the principles of differentiating compositions of functions and how complex functions can be decomposed.

Derivatives of Composed Functions

By applying a function $\mathbf{g}$ to the result of another function $\mathbf{h}$, we obtain a *composed function* $\mathbf{g} \circ \mathbf{h}$. The symbol $\circ$ indicates the composition. Writing $\mathbf{f} = \mathbf{g}(\mathbf{u})$ and $\mathbf{u} = \mathbf{h}(\mathbf{s})$ allows the following two interpretations of the composition:

$$\mathbf{f} = (\mathbf{g} \circ \mathbf{h})(\mathbf{s}) = \mathbf{g}(\mathbf{h}(\mathbf{s})) \tag{5.20}$$

(i) A transformation of the output parameters of **h** from **u** to **f** by prepending **g**

(ii) A transformation of the input parameters of **g** from **u** to **s** by appending **h**

In both cases, we obtain the same result. The significance of the different interpretations will become clear in Section 5.5, when several functions will be composed. Using the chain rule, we can apply this reparametrization to the derivatives. The formula for the second order derivative is the multivariate form of *Faà di Bruno's formula* [65, 66].

$$\frac{\mathrm{d} f_k}{\mathrm{d} s_i} = \frac{\partial g_k}{\partial u_a} \cdot \frac{\partial u_a}{\partial s_i} \tag{5.21}$$

$$\frac{\mathrm{d}^2 f_k}{\mathrm{d} s_i \, \mathrm{d} s_j} = \frac{\partial^2 g_k}{\partial u_a \, \partial u_b} \cdot \frac{\partial u_a}{\partial s_i} \cdot \frac{\partial u_b}{\partial s_j} + \frac{\partial g_k}{\partial u_a} \cdot \frac{\partial^2 u_a}{\partial s_i \, \partial s_j} \tag{5.22}$$

We now introduce *Spivak's notation* for writing derivatives [67]. In this functional notation, $D_i f$ denotes the derivative of f with respect to its i-th argument. The evaluation of this derivative for $\hat{x}$ is written as $D_i f(\hat{x})$. Leaving the index out, Df denotes the *gradient* and $D^2 f$ the *hessian* of f. The *jacobian* of a vector function **f** is denoted by $D\mathbf{f}$. Spivak notation allows (5.21) and (5.22) to be written without having to define the auxiliary variables **f** and **u**:

$$D_i \,(g_k \circ h) = D_a \, g_k \cdot D_i \, h_a \tag{5.23}$$

$$D_i D_j \,(g_k \circ h) = D_a D_b \, g_k \cdot D_i \, h_a \cdot D_j \, h_b + D_a \, g_k \cdot D_i D_j \, h_a \tag{5.24}$$

A basic implementation of (5.23) is shown in Algorithm 1. It results in the matrix product of the Jacobians of **g** and **h**. The time complexity of the operation results from the three loops. It depends on the length of the vectors **f**, **u**, and **s**.

$$O(|f| \cdot |u| \cdot |s|) \tag{5.25}$$

To compute (5.24) we make use of the symmetry of the second order derivative, e.g. $D_a D_b g_k = D_b D_a g_k$ to obtain the time complexity of the operation (Algorithm 2).

$$O\left(|f| \cdot \frac{|u|^2 + |u|}{2} \cdot \frac{|s|^2 + |s|}{2}\right) \tag{5.26}$$

Here, we assume that all partial derivatives are not zero. This means we have to consider all contributions. In fact, many of the partial derivatives disappear, which can reduce the number of operations. The sparse structure of the partial derivatives can be illustrated with a computational graph, introduced in Section 5.4.

```
Input: First order derivatives Dg[k,a] = $\partial g_k / \partial u_a$ and Dh[a,i] = $\partial u_a / \partial s_i$
Output: First order derivatives Df[k,i] = $\partial f_k / \partial s_i$
for $k \leftarrow 1$ to $|f|$ do
    for $i \leftarrow 1$ to $|s|$ do
        for $a \leftarrow 1$ to $|u|$ do
            Df[k,i] = Dg[k,a] * Dh[a,i];
        end
    end
end
```
Algorithm 1: Computation of first order derivatives according to (5.23).

Decomposition of a Function and its Derivatives

For systematic differentiation using the chain rule, a complex function is decomposed into simpler suboperations for which the partial derivatives can be calculated. With AD, this decomposition is performed down to the basic mathematical operations. This results in the automatic differentiation of an arbitrary functional. In Sections 5.5, 5.6, 5.7 and 5.8, we optimize the decomposition to identify recurring operations on IGA elements.

We now demonstrate the decomposition with a simple example. The function (5.27) is decomposed into basic mathematical operations for which we already

```
Input: First order derivatives Dg and Dh. Second order derivatives
       DDg[k,a,b] = $\partial^2 g_k / (\partial u_a \, \partial u_b)$ and DDh[a,i,j] = $\partial^2 u_a / (\partial s_i \, \partial s_j)$
Output: Second order derivatives DDf[k,i,j] = $\mathrm{d}^2 f_k / (\mathrm{d}s_i \, \mathrm{d}s_j)$ with $i \leq j$
for $k \leftarrow 1$ to $|f|$ do
    for $i \leftarrow 1$ to $|s|$ do
        for $j \leftarrow i$ to $|s|$ do
            DDf[k,i,j] = 0;
            for $a \leftarrow 1$ to $|u|$ do
                DDf[k,i,j] += Dg[k,a] * DDh[a,i,j] + DDg[k,a,a] *
                  Dh[a,i] * Dh[a,j];
                for $b \leftarrow a + 1$ to $|u|$ do
                    DDf[k,i,j] += DDg[k,a,b] * (Dh[a,i] * Dh[b,j] +
                      Dh[b,i] * Dh[a,j]);
                end
            end
        end
    end
end
```

Algorithm 2: Computation of second order derivatives by taking advantages of the symmetries according to (5.24)[3].

know the derivation rules. In Section 5.5, we will use the same procedure for the decomposition of the energy functional.

$$\mathbf{f}(\mathbf{s}) = (f_1) = \left(\frac{s_1 - s_2}{s_2 \, s_3} \right) \qquad \text{with } \mathbf{s} = (s_1, s_2, s_3) \tag{5.27}$$

Input **s** and output **f** are vectors. For simplicity, the result consists of a single component f_1, while the input consists of the three parameters s_1, s_2 and s_3. To calculate the derivatives of f_1, we decompose it into the simpler suboperations g_1, h_1 and h_2.

$$f_1 = g_1(\mathbf{u}) = \frac{u_1}{u_2} \qquad u_1 = h_1(\mathbf{s}) = s_1 - s_2 \qquad u_2 = h_2(\mathbf{s}) = s_2 \, s_3 \tag{5.28}$$

[3]A mistake in line 6 from the original publication has been corrected for this thesis.

This gives us the intermediate results u_1 and u_2, which are collected in the vector $\mathbf{u}$. The first order and second order partial derivatives can be determined using the basic derivation rules:

$$D_i\, h_1 = \left(\frac{\partial h_1}{\partial s_i}\right) = \begin{pmatrix}1 & -1 & 0\end{pmatrix} \qquad D_iD_j\, h_1 = \left(\frac{\partial h_1}{\partial s_i\, \partial s_j}\right) = \begin{pmatrix}0 & 0 & 0\\ & 0 & 0\\ \text{sym.} & & 0\end{pmatrix} \tag{5.29a}$$

$$D_i\, h_2 = \left(\frac{\partial h_2}{\partial s_i}\right) = \begin{pmatrix}0 & s_3 & s_2\end{pmatrix} \qquad D_iD_j\, h_2 = \left(\frac{\partial h_2}{\partial s_i\, \partial s_j}\right) = \begin{pmatrix}0 & 0 & 0\\ & 0 & 1\\ \text{sym.} & & 0\end{pmatrix} \tag{5.29b}$$

$$D_a\, g_1 = \left(\frac{\partial g_1}{\partial u_a}\right) = \begin{pmatrix}\frac{1}{u_2} & -\frac{u_1}{u_2{}^2}\end{pmatrix} \qquad D_aD_b\, g_1 = \left(\frac{\partial g_1}{\partial u_a\, \partial u_b}\right) = \begin{pmatrix}0 & -\frac{1}{u_2{}^2}\\ \text{sym.} & \frac{2\,u_1}{u_2{}^3}\end{pmatrix} \tag{5.29c}$$

We can compute the derivatives of $\mathbf{f}$ with respect to $\mathbf{s}$ from the partial derivatives of the suboperations by applying (5.23) and (5.24). Using SD, we perform this transformation with symbolic expressions. This gives us an analytic solution for the derivatives:

$$D_i\, (g \circ h) = \left(\frac{\mathrm{d}f_1}{\mathrm{d}s_i}\right) = \begin{pmatrix}\frac{1}{s_2\, s_3} & -\frac{s_1}{s_2{}^2\, s_3} & \frac{s_2 - s_1}{s_2\, s_3{}^2}\end{pmatrix} \tag{5.30a}$$

$$D_iD_j\, (g \circ h) = \left(\frac{\mathrm{d}^2 f_1}{\mathrm{d}s_i\, \mathrm{d}s_j}\right) = \begin{pmatrix}0 & -\frac{1}{s_2{}^2\, s_3} & -\frac{1}{s_2\, s_3{}^2}\\ & \frac{2\, s_1}{s_2{}^3\, s_3} & \frac{s_1}{s_2{}^2\, s_3{}^2}\\ \text{sym.} & & \frac{2\,(s_1 - s_2)}{s_2\, s_3{}^3}\end{pmatrix} \tag{5.30b}$$

With AD, the partial derivatives are directly evaluated for a specific set of parameters e.g. $\hat{\mathbf{s}} = (3, 2, 0.5)$. The transformation is applied to the numerical results.

$$u_1 = h_1(\hat{\mathbf{s}}) = 1 \qquad D_i\, h_1(\hat{\mathbf{s}}) = \begin{pmatrix}1 & -1 & 0\end{pmatrix} \qquad D_iD_j\, h_1(\hat{\mathbf{s}}) = \begin{pmatrix}0 & 0 & 0\\ & 0 & 0\\ \text{sym.} & & 0\end{pmatrix} \tag{5.31a}$$

$$u_2 = h_2(\hat{\mathbf{s}}) = 1 \qquad D_i\, h_2(\hat{\mathbf{s}}) = \begin{pmatrix}0 & 0.5 & 2\end{pmatrix} \qquad D_iD_j\, h_2(\hat{\mathbf{s}}) = \begin{pmatrix}0 & 0 & 0\\ & 0 & 1\\ \text{sym.} & & 0\end{pmatrix} \tag{5.31b}$$

$$f_1 = g_1(\mathbf{u}) = 1 \qquad D_a\, g_1(\mathbf{u}) = \begin{pmatrix}1 & -1\end{pmatrix} \qquad D_aD_b\, g_1(\mathbf{u}) = \begin{pmatrix}0 & -1\\ \text{sym.} & 2\end{pmatrix} \tag{5.31c}$$

By applying the chain rule on the partial derivatives, we obtain

$$D_i\,(g \circ h) = \begin{pmatrix}1 & \underline{-1.5} & -2\end{pmatrix} \quad D_iD_j\,(g \circ h) = \begin{pmatrix}0 & -0.5 & -2\\ & 1.5 & \underline{3}\\ \text{sym.} & & 8\end{pmatrix}. \tag{5.32}$$

Evaluating (5.30) for $\mathbf{s} = \hat{\mathbf{s}}$ obviously yields to the same results. This shows the strong relation between AD and SD. The computation of the underscored values is visualized in Section 5.4.

Hyper-dual numbers

A convenient way to implement AD is by using hyper-dual numbers [14]. A dual number consists of a real and an infinitesimal component. The real component is used to store the value of a function, while the infinitesimal component stores its derivative. For multivariable functions, an n-dimensional dual number, also called *Jet*, with n infinitesimal components can be used. Hyper-dual numbers are an extension of dual numbers, which are also able to store the second order derivatives. They are used in many applications such as optimization and rendering systems for automatic derivation of multivariable functions. We write the evaluated function f at s as a hyper-dual number:

$$< \quad \underset{\substack{\uparrow \\ \text{real component}}}{f(s)} \quad , \quad \underset{\substack{\uparrow \\ \text{infinitesimal component} \\ \text{first order}}}{Df(s)} \quad , \quad \underset{\substack{\uparrow \\ \text{infinitesimal component} \\ \text{second order}}}{D^2 f(d)} \quad > \tag{5.33}$$

Mathematical operations with hyper-dual numbers apply the chain rule to the infinitesimal components. This gives us another hyper-dual number, which contains the result of the operation as the real component and the derivatives as infinitesimal components. Therefore, if a calculation is performed with hyper-dual numbers, the derivatives are computed 'automatically'. For the example in Section 5.4, we write **s** as a function of **ŝ** to obtain the input parameters as hyper-dual numbers:

$$s_1(\hat{\mathbf{s}}) = \left\langle 3, \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \mathbf{0} \right\rangle \quad s_2(\hat{\mathbf{s}}) = \left\langle 2, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \mathbf{0} \right\rangle \quad s_3(\hat{\mathbf{s}}) = \left\langle 0.5, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}, \mathbf{0} \right\rangle \tag{5.34}$$

Subtraction and multiplication of these hyper-dual numbers leads to the results in (5.31a) and (5.31b):

$$\begin{aligned} u_1(\hat{\mathbf{s}}) &= s_1(\hat{\mathbf{s}}) - s_2(\hat{\mathbf{s}}) = \left\langle 1, \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}, \mathbf{0} \right\rangle \\ u_2(\hat{\mathbf{s}}) &= s_2(\hat{\mathbf{s}})\, s_3(\hat{\mathbf{s}}) = \left\langle 1, \begin{pmatrix} 0 \\ 0.5 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 & 0 & 0 \\ & 0 & 1 \\ \text{sym.} & & 0 \end{pmatrix} \right\rangle \end{aligned} \tag{5.35}$$

The result of the division of these intermediate values is equal to (5.32):

$$f_1(\hat{\mathbf{s}}) = \frac{u_1(\hat{\mathbf{s}})}{u_2(\hat{\mathbf{s}})} = \left\langle 1, \begin{pmatrix} 1 \\ -1.5 \\ -2 \end{pmatrix}, \begin{pmatrix} 0 & -0.5 & -2 \\ & 1.5 & 3 \\ \text{sym.} & & 8 \end{pmatrix} \right\rangle \tag{5.36}$$

Since the chain rule is applied with each operation, this is known as forward mode AD. Hyper-dual numbers are easy to implement and flexible to use. They can be combined with other methods. To express the base vectors of a NURBS, for example, the infinitesimal component can be computed directly using the algorithms of Section 5.3.

In the context of this paper, we use the open source library *HyperJet* [68] to deal with hyper-dual numbers in C++ and Python. With this implementation, functions are decomposed into basic mathematical operations. For an efficient and modular implementation of isogeometric elements, decomposition into basic mechanical functions, as shown in Section 5.5, is more suitable. We therefore investigate the computational graphs of the functions. Nevertheless, we use hyper-dual numbers for computing derivatives of individual parts of the calculation to create a good balance of performance and usability.

The Computational Graph

The application of the chain rule on a composed function can be visualized in a *computational graph*. This is a visual representation of the operations performed in (5.23) and (5.24). It helps in understanding the interaction between the individual parameters and allows the procedure to be optimized. The computational graph for the function (5.27) is shown in Figure 5.5. The graph consists of nodes, branches, edges, weights and paths.

Nodes The *nodes* of the graph correspond to the parameters of the function. We arrange the parameters into *layers*. The first layer contains the *input* parameters (s_1, s_2, s_3), the second the *intermediate results* (u_1, u_2) and the third the *output* (f_1).

Branches For each suboperation, we obtain a set of edges, which we call a *branch*. In our example, we obtain three branches: g_1 (red), h_1 (blue) and h_2 (green).

Edges The parameters of two layers are connected by the suboperations g_1, h_1, h_2. This connection is illustrated by *edges*. We use colors to distinguish the edges of each suboperation. We denote an edge between u_a and u_b as $(u_a \leftrightarrow u_b)$. For an edge that results from a certain suboperation g_k we write $(g_k : u_a \leftrightarrow u_b)$. For each suboperation we can add two types of edges to the graph:

(i) *First order edges* connecting nodes of different layers
e.g. the suboperation h_2 connects u_2 to s_3 by the edge $(h_2 : u_2 \leftrightarrow s_3)$

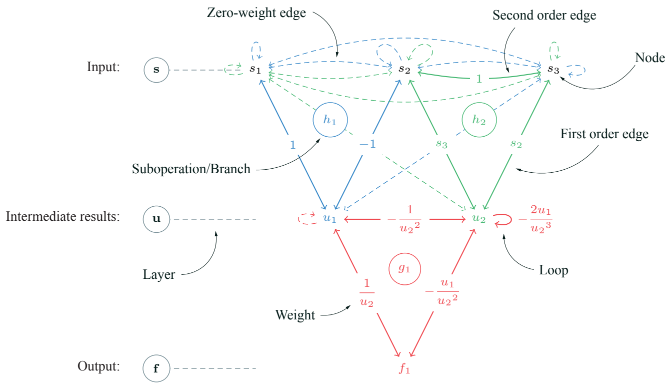


Figure 5.5: Structure of the computational graph using the function (5.27).

(ii) *Second order edges* connecting nodes of the same layer
e.g. the suboperation h_2 connects s_2 to s_3 by the edge $(h_2 : s_2 \leftrightarrow s_3)$

Note that a second order edge can also connect a node to itself, forming a *loop* e.g. $(g_1 : u_2 \leftrightarrow u_2)$.

Weights The edges are weighted according to the partial derivatives of the suboperations (5.29).

(i) For the first order edges, we use the first order derivatives
e.g. the weight of $(h_2 : u_2 \leftrightarrow s_3)$ is $\partial h_2 / \partial s_3 = s_2$.

(ii) For the second order edges, we use the second order derivatives
e.g. the weight of $(h_2 : s_2 \leftrightarrow s_3)$ is $\partial h_2 / (\partial s_2 \, \partial s_3) = 1)$:

The weighted computational graph shows the interaction between the parameters. Edges with zero-weight indicate that two parameters do not interact owing to the corresponding suboperation. In Figure 5.5, the *zero-weight edges* are indicated by dashed lines. For a clearer overview, these edges can be hidden.

Paths The components of operations (5.23) and (5.24) correspond to the partial derivatives and, in turn, the weights of the first order edges (i) and second order edges (ii).

$$\frac{\mathrm{d} f_k}{\mathrm{d} s_i} = \underbrace{\overbrace{\frac{\partial g_k}{\partial u_a}}^{\text{(i)}} \cdot \overbrace{\frac{\partial u_a}{\partial s_i}}^{\text{(i)}}}_{\text{(I)}}$$

$$\frac{\mathrm{d}^2 f_k}{\mathrm{d} s_i \, \mathrm{d} s_j} = \underbrace{\overbrace{\frac{\partial^2 g_k}{\partial u_a \, \partial u_b}}^{\text{(ii)}} \cdot \overbrace{\frac{\partial u_a}{\partial s_i}}^{\text{(i)}} \cdot \overbrace{\frac{\partial u_b}{\partial s_j}}^{\text{(i)}}}_{\text{(II)}} + \underbrace{\overbrace{\frac{\partial g_k}{\partial u_a}}^{\text{(i)}} \cdot \overbrace{\frac{\partial^2 u_a}{\partial s_i \, \partial s_j}}^{\text{(ii)}}}_{\text{(III)}}$$

The products (I), (II) and (III) are visualized in the graph by introducing three types of weighted *path*:

(I) *First order paths:* $f_k \longrightarrow u_a \longrightarrow s_i$
From f_k to s_i, passing two first order edges e.g. Figure 5.6b.
The weight of the path corresponds to the product of the weights of the traversed edges.

(II) *Indirect second order paths:* $s_i \longrightarrow u_a \longrightarrow u_b \longrightarrow s_j$
From s_i to s_j, passing a first order edge ($u_a \leftrightarrow s_i$), a second order edge ($g_k : u_a \leftrightarrow u_b$), and again a first order edge ($u_b \leftrightarrow s_j$), e.g. Figure 5.7b.
The weight of the path corresponds to the product of the weights of the traversed edges.

(III) *Direct second order paths:* $s_i \longrightarrow s_j$
From s_i to s_j, passing one second order edge ($h_a : s_i \leftrightarrow s_j$), e.g. Figure 5.7a.
The weight of the path corresponds to the product of the weights of ($h_a : s_i \leftrightarrow s_j$) and the corresponding first order edge ($g_k : f_k \leftrightarrow u_a$).

According to (5.23) we total the weights of all first order paths from f_k to s_i when we apply the chain rule. For instance, to compute $\mathrm{d}f_1/\mathrm{d}s_2$, we total the weights of all paths from f_1 to s_2 shown in Figure 5.6:

$$\frac{\mathrm{d}f_1}{\mathrm{d}s_2} = \underset{\substack{\uparrow \\ \text{Fig. 5.6b}}}{-1} \underset{\substack{\uparrow \\ \text{5.6c}}}{-\,0.5} = -1.5 \tag{5.37}$$

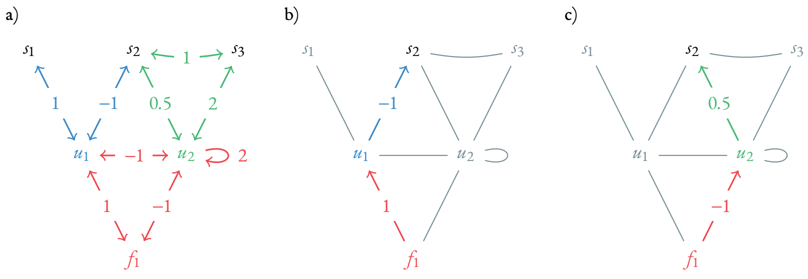
This corresponds with the results in Section 5.4. In the same way, we total the weights of all second order paths from s_i to s_j, when we apply (5.24). To compute $\mathrm{d}f_1/(\mathrm{d}s_2\,\mathrm{d}s_3)$, we use the weights of all possible second order paths from s_2 and s_3 shown in Figure 5.7 and 5.8.

$$\frac{\mathrm{d}^2 f_1}{\mathrm{d}s_2\,\mathrm{d}s_3} = \underset{\substack{\uparrow \\ \text{5.7a}}}{-\,1} + \underset{\substack{\uparrow \\ \text{5.7b}}}{2} + \underset{\substack{\uparrow \\ \text{5.7c}}}{2} + \underset{\substack{\uparrow \\ \text{5.8a}}}{0} + \underset{\substack{\uparrow \\ \text{5.8b}}}{0} + \underset{\substack{\uparrow \\ \text{5.8c}}}{0} = 3 \tag{5.38}$$

Paths with zero-weight do not contribute to the final result and can be ignored. In this example, only three of the six possible paths need be evaluated. The graph helps to simplify the computation of the chain rule by identifying such zero-weight paths. We will use the computational graph in Sections 5.6, 5.7 and 5.8 to visualize the values and structure of the partial derivatives of $\hat{\Pi}$ and $\mathbf{g}$, which will be discussed in the next section. In Figure 5.9, we show two more examples, in which the derivatives of the dot product are calculated. These two graphs will frequently appear in Section 5.6 as branches of the graphs for the element formulations.

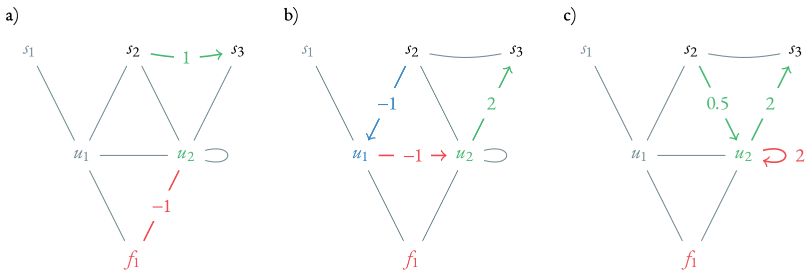
5.5 Element Formulation using the Direct and Adjoint Methods

Following the classical approach, an isogeometric element is formulated by deriving the load vector $\mathbf{F}$. The stiffness matrix $\mathbf{K}$ results from the derivatives of $\mathbf{F}$ with respect to $\mathbf{x}$. We, however, derive the elements directly from the energy functional $\Pi^{(e)}$ according to (5.9). For the sake of simplicity, we write $\Pi^{(e)} = \Pi$, where Π describes the internal energy at a certain point ω of the element as a function of



$f_1 \xrightarrow{1} u_1 \xrightarrow{-1} s_2 : -1$ $\quad$ $f_1 \xrightarrow{-1} u_2 \xrightarrow{0.5} s_2 : -0.5$

Figure 5.6: (a) The weights are evaluated for $\mathbf{s} = \hat{\mathbf{s}}$. Zero-weight edges are hidden. To compute the derivative of f_1 w.r.t. s_2, we total the weights of all first order paths from f_1 to s_2. There are two possible paths: (b) over u_1 and (c) over u_2. The weight of the paths corresponds to the product of the weights of the edges.



$(f_1 \xrightarrow{-1} u_2) \times (s_2 \xrightarrow{1} s_3) : -1$ $\quad$ $s_2 \xrightarrow{-1} u_1 \xrightarrow{-1} u_2 \xrightarrow{2} s_3 : 2$ $\quad$ $s_2 \xrightarrow{0.5} u_2 \xrightarrow{2} u_2 \xrightarrow{2} s_3 : 2$

Figure 5.7: To compute the second order derivative of f_1 w.r.t. s_2 and s_3, we total the weights of all possible second order paths from s_2 to s_3. The diagram shows all paths with a non-zero weight. The weight of the direct path (a) corresponds to the weight of the connecting edge ($h_2 : s_2 \longrightarrow s_3$) multiplied by the weight of the corresponding first order edge ($g_1 : f_1 \longrightarrow u_2$). The weight of the indirect paths (b and c) corresponds to the product of the weights of the edges.

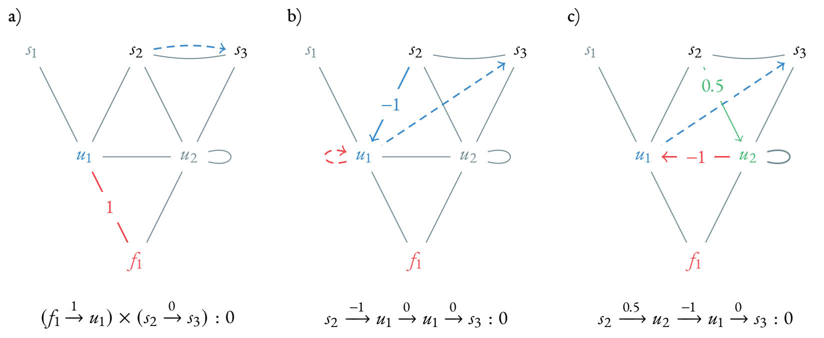

a)
b)
c)
s_1
s_2
s_3
u_1
u_2
f_1
1
−1
0.5
−1
$(f_1 \xrightarrow{1} u_1) \times (s_2 \xrightarrow{0} s_3) : 0$
$s_2 \xrightarrow{-1} u_1 \xrightarrow{0} u_1 \xrightarrow{0} s_3 : 0$
$s_2 \xrightarrow{0.5} u_2 \xrightarrow{-1} u_1 \xrightarrow{0} s_3 : 0$

Figure 5.8: All zero-weight second order paths from s_2 to s_3: (a) shows a direct path while (b) and (c) show indirect paths. Since at least one edge has a weight of zero (dashed lines), they do not contribute to the final result.

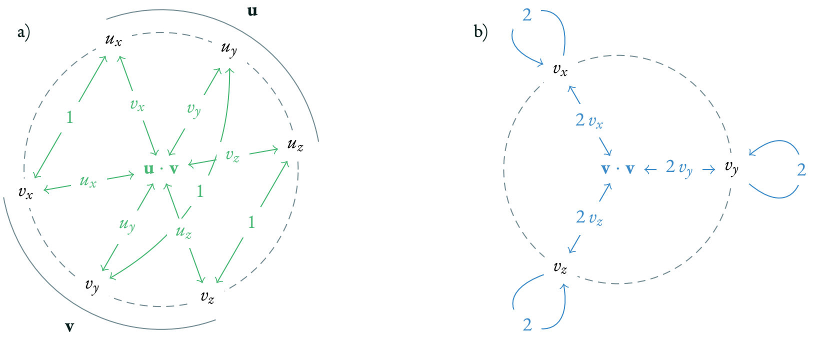


Figure 5.9: Computational graph for the derivatives of (a) the dot product of two vectors $\mathbf{u} = (u_x, u_y, u_z)$ and $\mathbf{v} = (v_x, v_y, v_z)$ and (b) the squared length of $\mathbf{v}$. For a clearer overview, we use a radial alignment of layers and parameters.

discrete nodal positions **x**. Load vector and stiffness are obtained from the first and second order derivatives of Π, as in Section 5.2. To compute the derivatives, we decompose Π into a sequence of suboperations by applying the techniques described in Section 5.4.

$$\Pi(\mathbf{x}) = (p \circ \mathbf{e} \circ \mathbf{g})(\mathbf{x}) \tag{5.39}$$

Here, **g** evaluates the geometry in the deformed configuration (e.g. the components of the base vectors), **e** provides the corresponding strains and p returns the energy according to the material law. Depending on the element type, the vectors **g** and **e** consist of $|g|$ and $|e|$ components, respectively. For an IGA truss, we calculate $|e| = 1$ strain ε and $|g| = 3$ components of the base vector $\mathbf{a}_1 = (a_{1x}, a_{1y}, a_{1z})$. Section 5.6 gives more details about the various element types. According to Section 5.3, the dimension $|x|$ of **x** depends on the polynomial degree of the NURBS geometry. The energy p is always a scalar. We apply (5.23) and (5.24) to compute the derivatives of (5.39) with respect to **x**. This results in the force vector **F** and the stiffness matrix **K**.

$$\begin{aligned} F_r &= D_r\, \Pi \\ &= D_r\, (p \circ \mathbf{e} \circ \mathbf{g}) \\ &= D_a\, p \cdot D_i\, e_a \cdot D_r\, g_i \end{aligned} \tag{5.40}$$

$$\begin{aligned} K_{rs} &= D_r D_s\, \Pi \\ &= D_r D_s\, (p \circ \mathbf{e} \circ \mathbf{g}) \end{aligned}\;. \tag{5.41}$$

Both expressions can be resolved in two ways: (i) using the direct method (DM), which evaluates from right to left, or (ii) using the adjoint method (AM), which evaluates from left to right. Since the operations are associative, the final result is identical in both cases. However, the two methods might produce different intermediate results and therefore change the computational effort. The classic method of formulating isogeometric elements uses DM. We perform the calculation for DM and AM to make a comparison.

The Direct Method

DM evaluates (5.40) from right to left. This implies that we first compute the derivatives of $\mathbf{e} \circ \mathbf{g}$, which corresponds to the derivatives of the strains $\mathbf{e}$ with respect to the nodal locations $\mathbf{x}$. The intermediate result is the so-called *B-matrix* of size $|e| \times |x|$. To facilitate familiarity with the Spivak notation, we also provide the results in a simplified classical notation.

$$\begin{aligned} F_r &= \frac{\partial p}{\partial e_a} B_{ar} \\ &= D_a p \cdot \underbrace{D_r (e_a \circ \mathbf{g})}_{B_{ar}} \end{aligned} \tag{5.42}$$

$$\begin{aligned} B_{ar} &= \frac{\partial e_a}{\partial g_i} \frac{\partial g_i}{\partial x_r} \\ &= D_i e_a \cdot D_r g_i \end{aligned} \tag{5.43}$$

In the same way, we use (5.24) to calculate the stiffness matrix $\mathbf{K}$. As an additional intermediate result, we obtain the third order tensor $\mathbf{B}^{(2)}$ of size $|e| \times |x| \times |x|$.

$$\begin{aligned} K_{rs} &= \frac{\partial^2 p}{\partial e_a \, \partial e_b} B_{ar} B_{bs} + \frac{\partial p}{\partial e_a} B^{(2)}_{ars} \\ &= D_a D_b p \cdot \underbrace{D_r (e_a \circ \mathbf{g})}_{B_{ar}} \cdot \underbrace{D_s (e_b \circ \mathbf{g})}_{B_{bs}} + D_a p \cdot \underbrace{D_r D_s (e_a \circ \mathbf{g})}_{B^{(2)}_{ars}} \end{aligned} \tag{5.44}$$

$$\begin{aligned} B^{(2)}_{ars} &= \frac{\partial^2 e_a}{\partial g_i \, \partial g_j} \frac{\partial g_i}{\partial x_r} \frac{\partial g_j}{\partial x_s} + \frac{\partial e_a}{\partial g_i} \frac{\partial^2 g_i}{\partial x_r \, \partial x_s} \\ &= D_i D_j e_a \cdot D_r g_i \cdot D_s g_j + D_i e_a \cdot D_r D_s g_i \end{aligned} \tag{5.45}$$

The Adjoint Method

AM evaluates (5.40) from left to right. In this case, we first calculate the derivative of $\hat{\Pi} = p \circ \mathbf{e}$ which corresponds with the derivatives of the energy with respect to the components of the geometry $\mathbf{g}$. Since we are computing the derivatives of an energy functional, we interpret this intermediate result mechanically as the core force vector $\hat{\mathbf{F}}$ of size $|g|$.

$$\begin{aligned} F_r &= \hat{F}_i \frac{\partial g_i}{\partial x_r} \\ &= \underbrace{D_i\,(p \circ \mathbf{e})}_{\hat{F}_i} \cdot D_r\, g_i \end{aligned} \tag{5.46}$$

$$\begin{aligned} \hat{F}_i &= \frac{\partial p}{\partial e_a} \frac{\partial e_a}{\partial g_i} \\ &= D_a\, p \cdot D_i\, e_a \end{aligned} \tag{5.47}$$

We compute the stiffness matrix $\mathbf{K}$ by applying (5.24) from left to right. In this case, we interpret the intermediate result as the core stiffness matrix $\hat{\mathbf{K}}$ of size $|g| \times |g|$.

$$\begin{aligned} K_{rs} &= \hat{K}_{ij} \frac{\partial g_i}{\partial x_r} \frac{\partial g_j}{\partial x_s} + \hat{F}_i \frac{\partial^2 g_i}{\partial x_r\, \partial x_s} \\ &= \underbrace{D_i D_j\,(p \circ \mathbf{e})}_{\hat{K}_{ij}} \cdot D_r\, g_i \cdot D_s\, g_j + \underbrace{D_i\,(p \circ \mathbf{e})}_{\hat{F}_i} \cdot D_r D_s\, g_i \end{aligned} \tag{5.48}$$

$$\begin{aligned} \hat{K}_{ij} &= \frac{\partial^2 p}{\partial e_a\, \partial e_b} \frac{\partial e_a}{\partial g_i} \frac{\partial e_b}{\partial g_j} + \frac{\partial p}{\partial e_a} \frac{\partial^2 e_a}{\partial g_i\, \partial g_j} \\ &= \underbrace{D_a D_b\, p \cdot D_i\, e_a \cdot D_j\, e_b}_{\hat{\mathbf{K}}_\mathrm{m}} + \underbrace{D_a\, p \cdot D_i D_j\, e_a}_{\hat{\mathbf{K}}_\mathrm{g}} \end{aligned} \tag{5.49}$$

Note that $\hat{\mathbf{K}}$ can be decomposed into $\hat{\mathbf{K}}_\mathrm{m}$ and $\hat{\mathbf{K}}_\mathrm{g}$, where $\hat{\mathbf{K}}_\mathrm{m}$ is the material stiffness matrix and $\hat{\mathbf{K}}_\mathrm{g}$ the geometric stiffness matrix of the element. The two components can be transformed separately to $\mathbf{K}_\mathrm{m}$ and $\mathbf{K}_\mathrm{g}$ according to the geometric description [49, 50]. In stability analysis, this can represent an easy way of extracting the geometric stiffness of isogeometric elements. However, this aspect will not be discussed any further in this paper.

A Comparison

In DM and AM, the operations are performed in opposite directions. In both cases, the derivation rules are applied correctly. Therefore the results are the same and equal to the analytic solution. The difference between DM and AM results from the intermediate steps shown in Figure 5.10. Using DM, we compute the derivatives of different outputs with respect to the same input parameters **x**. Each step transforms the output by prepending an additional function. AM computes the derivatives of the same output Π with respect to different input parameters. In each step, we transform the input by appending an additional function. This approach allows a modular structure and reduces the computational effort and memory consumption.

Modularization DM applies the relation of the nodal coordinates **x** and the geometry **g** as the first step of the computation. It affects all intermediate results. It is not possible to separate mechanics and geometry. AM applies this relation in the final step. As $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ are in fact independent of the chosen geometry description, it is possible to make a clean separation between mechanics and geometry.

Computational Time Computation of the partial derivatives of the suboperations takes the same time for DM as it does for AM. However, the effort required to apply the chain rule differs. According to (5.25) and (5.26), we can compute the complexity of the computation of the force $O^{(\bullet)}$ and the stiffness $O^{(\bullet 2)}$.

$$O^{(\mathrm{DM})}(|e| \cdot |g| \cdot |x| + 1 \cdot |e| \cdot |x|) \tag{5.50}$$
$$O^{(\mathrm{AM})}(1 \cdot |e| \cdot |g| + 1 \cdot |g| \cdot |x|) \tag{5.51}$$

$$O^{(\mathrm{DM2})}\left(|e| \cdot \frac{|g|^2 + |g|}{2} \cdot \frac{|x|^2 + |x|}{2} + 1 \cdot \frac{|e|^2 + |e|}{2} \cdot \frac{|x|^2 + |x|}{2}\right) \tag{5.52}$$
$$O^{(\mathrm{AM2})}\left(1 \cdot \frac{|e|^2 + |e|}{2} \cdot \frac{|g|^2 + |g|}{2} + 1 \cdot \frac{|g|^2 + |g|}{2} \cdot \frac{|x|^2 + |x|}{2}\right) \tag{5.53}$$

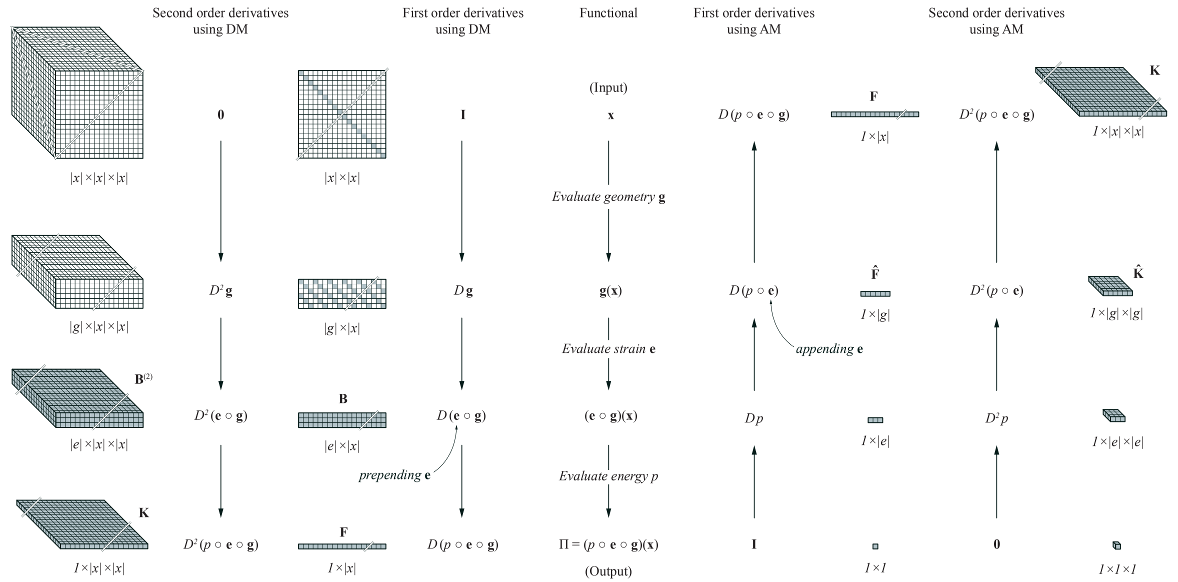


Figure 5.10: Comparison of the computation of an IGA element using DM (left) and AM (right). The evaluation steps of the energy functional are shown in the middle. With DM, the derivatives are evaluated in the same order (top to bottom) while with AM, they are evaluated in the opposite direction (bottom to top). The final results **F** and **K** are the same. The size of the intermediate results are shown for a membrane element (number of strain components $|e| = 3$, number of base vector components $|g| = 6$). Grey cells indicate the sparse structure of the intermediate results. This means that only the grey cells are assigned non-zero values. Variable sizes of vectors and matrices are indicated by cutting lines.

p	$\|x\|$	$O^{(DM)}$	$O^{(AM)}$		$O^{(DM2)}$	$O^{(AM2)}$		$\mathscr{M}^{(DM2)}$	$\mathscr{M}^{(AM2)}$	
1	6	24	21	87.5%	147	132	89.8%	42	14	33.3%
2	9	36	30	83.3%	315	276	87.6%	90	14	15.6%
3	12	48	39	81.2%	546	474	86.8%	156	14	9.0%
4	15	60	48	80.0%	840	726	86.4%	240	14	5.8%
5	18	72	57	79.2%	1197	1032	86.2%	342	14	4.1%
6	21	84	66	78.6%	1617	1392	86.1%	462	14	3.0%
7	24	96	75	78.1%	2100	1806	86.0%	600	14	2.3%

Table 5.1: Time complexity O and memory consumption $\mathscr{M}$ of a truss element ($|e| = 1$, $|g| = 3$) w.r.t. the polynomial degree p

p	$\|x\|$	$O^{(DM)}$	$O^{(AM)}$		$O^{(DM2)}$	$O^{(AM2)}$		$\mathscr{M}^{(DM2)}$	$\mathscr{M}^{(AM2)}$	
1	12	252	90	35.7%	5382	1764	32.8%	468	54	11.5%
2	27	567	180	31.7%	26 082	8064	30.9%	2268	54	2.4%
3	48	1008	306	30.4%	81 144	24 822	30.6%	7056	54	0.8%
4	75	1575	468	29.7%	196 650	59 976	30.5%	17 100	54	0.3%
5	108	2268	666	29.4%	406 134	123 732	30.5%	35 316	54	0.2%
6	147	3087	900	29.2%	750 582	228 564	30.5%	65 268	54	0.1%
7	192	4032	1170	29.0%	1 278 432	389 214	30.4%	111 168	54	0.05%

Table 5.2: Time complexity O and memory consumption $\mathscr{M}$ of a membrane element ($|e| = 3$, $|g| = 6$) w.r.t. the polynomial degree p in both parameter directions.

p	$\|x\|$	$O^{(DM)}$	$O^{(AM)}$		$O^{(DM2)}$	$O^{(AM2)}$		$\mathscr{M}^{(DM2)}$	$\mathscr{M}^{(AM2)}$	
1	24	1440	270	18.8%	87 300	14 445	16.5%	3600	132	3.7%
2	81	4860	783	16.1%	966 411	150 390	15.6%	39 852	132	0.3%
3	192	11 520	1782	15.5%	5 391 648	834 705	15.5%	222 336	132	0.06%
4	375	22 500	3429	15.2%	20 515 500	3 173 445	15.5%	846 000	132	0.016%
5	648	38 880	5886	15.1%	61 190 316	9 463 365	15.5%	2 523 312	132	0.005%
6	1029	61 740	9315	15.1%	154 211 085	23 848 020	15.5%	6 359 220	132	0.002%
7	1536	92 160	13 878	15.1%	343 501 056	53 119 665	15.5%	14 164 992	132	0.001%

Table 5.3: Time complexity O and memory consumption $\mathscr{M}$ of a volume element ($|e| = 6$, $|g| = 9$) w.r.t. the polynomial degree p in all three parameter directions.

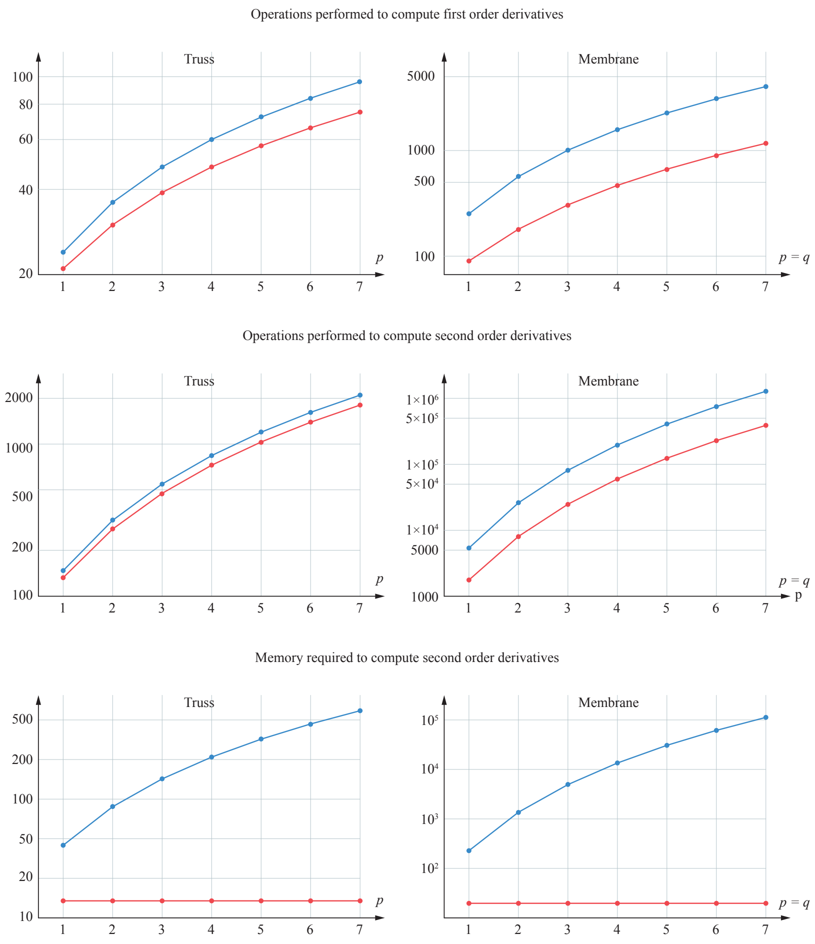


Figure 5.11: Time complexity and memory consumption for the direct method (blue) and the adjoint method (red)

Memory Consumption We now compare the number of intermediate results for DM and AM. In both cases, we assume that the memory for the final results is buffered by the FE kernel and should not be considered. An efficient implementation of DM usually takes advantage of the sparse pattern of $D\,\mathbf{g}$ or $D^2\,\mathbf{g}$ respectively (gray cells in Figure 5.11). Therefore, we only need to store $D\,(\mathbf{e}\circ\mathbf{g})(x)$ and $D^2\,(\mathbf{e}\circ\mathbf{g})(x)$, which are $\mathscr{M}^{(\mathrm{DM2})} = |e| \times |x| + |e| \times |x|^2$ values. For AM, we need to store $\mathscr{M}^{(\mathrm{AM2})} = |e| + |g| + |e|^2 + |g|^2$ values. Note that $\mathscr{M}^{(\mathrm{AM2})}$ is independent of the polynomial degree. This enables more efficient implementation without any dynamic allocation of memory. By using the symmetry of the second order derivative, the memory demand can be further reduced. In practice, however, the entire matrix is often created to simplify calculation using classic matrix indices.

The time complexity and memory consumption for the computation of a truss and a membrane element with respect to the polynomial degree are shown in Table 5.1, Table 5.2 and Figure 5.11. Table 5.3 shows that the benefits of the adjoint method are even greater for a volume element. However, the scope of this paper does not extend to investigating volume elements.

5.6 Structural Elements

In this section, we present the computation of $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ for the following types of isogeometric elements: a truss [27], a Bernoulli beam [28], a membrane [27] and a Kirchhoff shell [29]. We focus on the formulation using the adjoint method described in Section 5.5. For a detailed description of the mechanical behavior, see the relevant publications. We compute $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ by evaluating the derivatives of $\hat{\Pi}$ with respect to $\mathbf{g}$. For the purpose of generalization, we recompose $\hat{\Pi}$ as

$$\hat{\Pi}(\mathbf{g}) = (p\circ\mathbf{e})(\mathbf{g}) = (P\circ\mathbf{d})(\mathbf{g}), \tag{5.54}$$

$$\text{where} \quad P = \frac{1}{2}\,W_{ab}\,d_a\,d_b. \tag{5.55}$$

This recomposition has to be equivalent to the original formulation in the sense that it has the same input $\mathbf{g}$ and output Π. Therefore, we choose $\mathbf{W}$ and $\mathbf{d}$ in the appropriate manner. As long as $|e| = |d|$, the observations in Section 5.5 remain valid. We accumulate all constant terms as weights W_{ab}, where $W_{ab} = W_{ba}$. $\mathbf{W}$

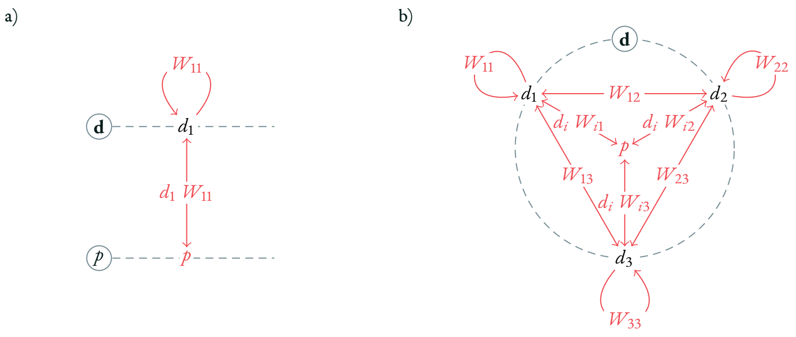


Figure 5.12: Computational graph of (5.55) for $|d| = 1$ (a) and $|d| = 3$ (b).

includes the constant integration weight α from (5.9). The partial derivatives of P are then simplified to:

$$D\,P = \frac{\partial P}{\partial d_a} = W_{ab}\,d_b \qquad\qquad D^2\,P = \frac{\partial P}{\partial d_a\,\partial d_b} = W_{ab} \tag{5.56}$$

Figure 5.12 shows the partial derivatives for $|d| = 1$ and $|d| = 3$, respectively. For each type of element, we only need to define $\mathbf{W}$ and $\mathbf{d}$ to compute the internal energy from a given geometry $\mathbf{g}$ using (5.54). Two different approaches are used to compute $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$: for the truss and membrane elements, we perform a complete manual analysis of the computational graph. For the beam and the shell element, we analyze parts of the graph manually and use algorithmic differentiation with hyper-dual numbers for branches which are too complex for manual optimization. In the following sections, $\mathbf{a}_\alpha$ denotes a base vector in the actual configuration (deformed), while $\mathbf{A}_\alpha$ denotes the same base vector in the reference configuration (undeformed). The dot product can be simplified using $\mathbf{a}_\gamma \cdot \mathbf{a}_\delta = a_{\gamma\delta}$ and $\mathbf{A}_\gamma \cdot \mathbf{A}_\delta = A_{\gamma\delta}$. We assume that the reference configuration is constant.

Truss

To describe the mechanics of a simple truss or cable, we only need to evaluate the geometry for the basis vector $\mathbf{a}_1$. In three-dimensional space, $\mathbf{a}_1$ consists of three components, a_{1x}, a_{1y} and a_{1z}, which are collected in $\mathbf{g}$:

$$\mathbf{g} = \underbrace{(a_{1x}, a_{1y}, a_{1z})}_{\mathbf{a}_1} \tag{5.57}$$

We will now apply the Green-Lagrange strain measurement $\varepsilon_{\mathrm{GL}}$. The material stiffness $\mathbf{D}$ is described by the Young's modulus E and the area of the cross-section A. By grouping the constant terms in $\mathbf{W}$ we obtain

$$\mathbf{d} = \underbrace{(a_{11} - A_{11})}_{d_1} \tag{5.58}$$

$$W_{11} = \frac{1}{4}\, T_{11}\, D_{11}\, T_{11}\, \alpha \qquad D_{11} = EA \qquad T_{11} = \frac{1}{A_{11}} \tag{5.59}$$

where $\mathbf{T}$ transforms the material stiffness $\mathbf{D}$ into parameter space. Correctness can be checked by evaluating the composition, which results in the familiar formula for the truss's internal energy.

$$\hat{\Pi} = P \circ \mathbf{d} = \frac{1}{2}\, \underbrace{\frac{1}{4}\, \frac{1}{A_{11}}\, EA\, \frac{1}{A_{11}}\, \alpha}_{W_{11}}\, \underbrace{(a_{11} - A_{11})}_{d_1}^2 = \frac{1}{2}\, EA (\underbrace{\frac{a_{11} - A_{11}}{2\, A_{11}}}_{\varepsilon_{\mathrm{GL}}})^2\, \alpha \tag{5.60}$$

The partial derivatives of $\mathbf{d}$ with respect to $\mathbf{g}$ are given by:

$$D\, d_1 = \begin{pmatrix} 2\, a_{1x} & 2\, a_{1y} & 2\, a_{1z} \end{pmatrix} \tag{5.61}$$

$$D^2\, d_1 = \begin{pmatrix} 2 & 0 & 0 \\ & 2 & 0 \\ \text{sym.} & & 2 \end{pmatrix} \tag{5.62}$$

This corresponds to the computational graph already shown in Figure 5.9b. The partial derivatives of $\hat{\Pi}$ and $\mathbf{d}$ are shown in the computational graph in Figure 5.13

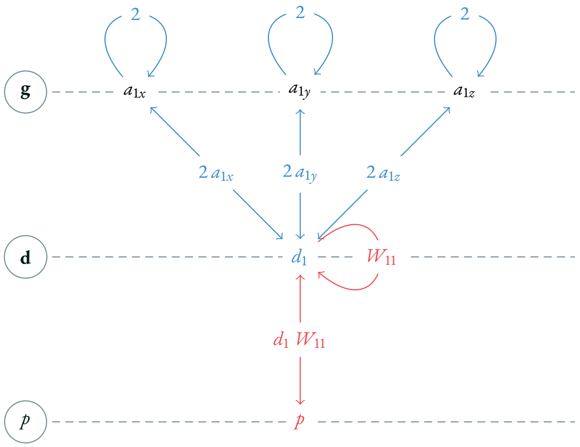


Figure 5.13: Computational graph of a truss element. Suboperations: p (red), $d_1 = a_{11} - A_{11}$ (blue)

which combines Figure 5.9b and Figure 5.12a. Using (5.23) and (5.24), we obtain $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$. This corresponds to the load vector and stiffness matrix of a generic truss of length α and base vector $\mathbf{a}_1$ (Figure 5.14a). In Section 5.7, we transform these results into the appropriate geometry description. This allows us to generate a classic nonlinear FE truss (Section 5.7), a simple isogeometric cable (Section 5.7), and an embedded cable (Section 5.7).

Membrane

The geometry of the membrane element is described by a surface. This gives us the two base vectors $\mathbf{a}_1$ and $\mathbf{a}_2$, which means that $\mathbf{g}$ consists of 6 components.

$$\mathbf{g} = (\underbrace{a_{1x}, a_{1y}, a_{1z}}_{\mathbf{a}_1}, \underbrace{a_{2x}, a_{2y}, a_{2z}}_{\mathbf{a}_2}) \tag{5.63}$$

For a membrane, we consider deformations in the direction of the base vectors (d_1 and d_2). However, shear deformation can also occur (d_3). This leads to the three

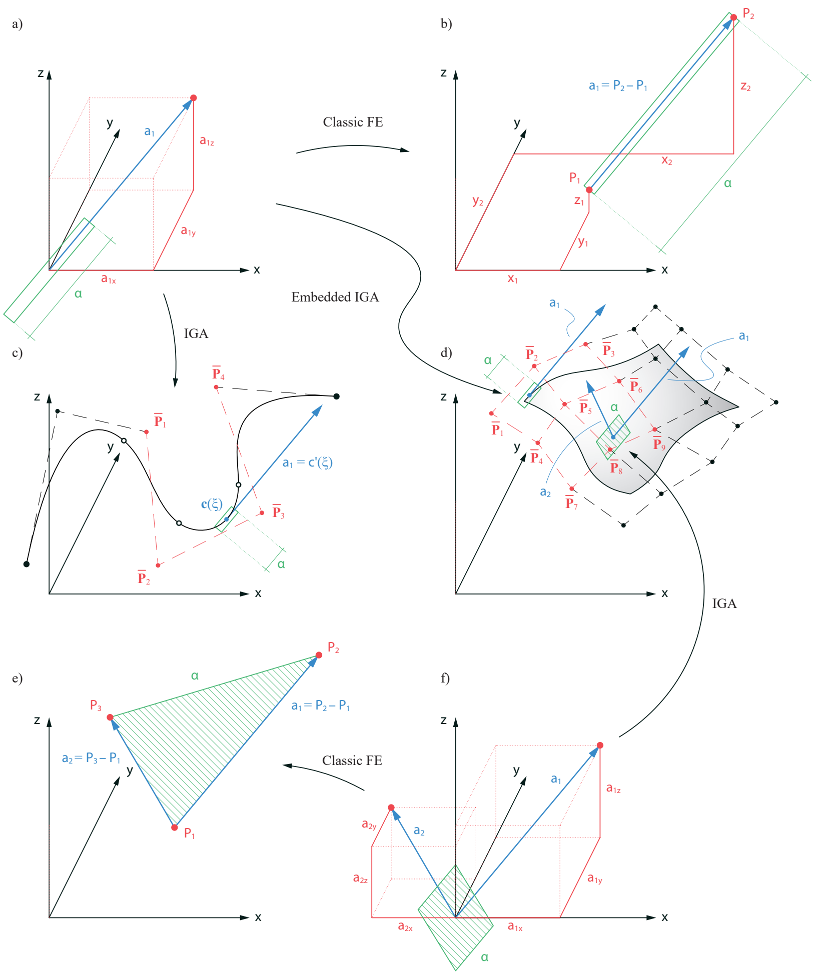


Figure 5.14: Force $\hat{\mathbf{F}}$ and Stiffness $\hat{\mathbf{K}}$ of (a) a generic truss of length α and base vector $\mathbf{a}_1$ is transformed to (b) a classic linear truss element, (c) an IGA cable element, (d) an embedded IGA cable element. In the same way (f) a generic membrane element of area α and base vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ is transformed to (e) a classic linear membrane element and (d) an IGA membrane element. The base vectors (blue) are controlled by different parameters (red).

components of **d**.

$$\mathbf{d} = (\underbrace{a_{11} - A_{11}}_{d_1}, \underbrace{a_{22} - A_{22}}_{d_2}, \underbrace{a_{12} - A_{12}}_{d_3}) \tag{5.64}$$

The weights are computed with Young's modulus E, Poisson's ratio ν, and thickness t. The transformation matrix $\mathbf{T}$ according to (29) in [27] transforms the material stiffness $\mathbf{D}$ from the local Cartesian space $\mathbf{e}_\gamma$ into the curvilinear space $\mathbf{A}_\alpha$. It takes into account that the shear deformation is symmetrical. Here $\mathbf{A}^\alpha$ denotes the contravariant base vectors of $\mathbf{A}_\alpha$.

$$t_{\gamma\delta}{}^{\alpha\beta} = (\mathbf{e}_\gamma \cdot \mathbf{A}^\alpha)(\mathbf{A}^\beta \cdot \mathbf{e}_\delta) \tag{5.65}$$

$$(T_{ab}) = \begin{pmatrix} t_{11}{}^{11} & t_{11}{}^{22} & t_{11}{}^{12} + t_{11}{}^{21} \\ t_{22}{}^{11} & t_{22}{}^{22} & t_{22}{}^{12} + t_{22}{}^{21} \\ 2\,t_{12}{}^{11} & 2\,t_{12}{}^{22} & 2\,(t_{12}{}^{12} + t_{12}{}^{21}) \end{pmatrix} \tag{5.66}$$

The partial derivatives of d_1 and d_2 are computed as in (5.62). For d_3 we obtain

$$D\,d_3 = \begin{pmatrix} a_{2x} & a_{2y} & a_{2z} & a_{1x} & a_{1y} & a_{1z} \end{pmatrix} \tag{5.67}$$

$$D^2\,d_3 = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ & 0 & 0 & 0 & 1 & 0 \\ & & 0 & 0 & 0 & 1 \\ & & & 0 & 0 & 0 \\ & \text{sym.} & & & 0 & 0 \\ & & & & & 0 \end{pmatrix} \tag{5.68}$$

which corresponds to Figure 5.9a. The partial derivatives are visualized by the computational graph in Figure 5.15, which combines Figure 5.9 and Figure 5.12b. It shows that the deformations d_1 and d_2 only have connections to $\mathbf{a}_1$ and $\mathbf{a}_2$, respectively. In contrast, d_3 represents the shear deformation and connects the two base vectors.

$$W_{ad} = \frac{1}{4}\, T_{ab}\, D_{bc}\, T_{cd}\, \alpha \qquad D_{ad} = \frac{E\,t}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ & 1 & 0 \\ \text{sym.} & & \frac{1-\nu}{2} \end{pmatrix}^{4} \tag{5.69}$$

By setting $W_{13} = W_{23} = W_{33} = 0$, we would remove all paths from $\hat{\Pi}$ over d_3. This would result in a surface element without shear stiffness.

We obtain $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ for a generic membrane element of area α and base vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ (Figure 5.14f). In Section 5.7, these results are transformed according to the respective geometry description of $\mathbf{a}_1$ and $\mathbf{a}_2$. Accordingly, we can generate a classic triangular FE membrane element (Section 5.7) or a simple isogeometric membrane (Section 5.7).

Beam

The isogeometric Bernoulli beam extends the truss element by adding terms for bending and torsion. The extensional strain energy can be taken from the truss element. To compute the energy arising from bending and torsion, we need the base vectors $\mathbf{a}_1$ and $\mathbf{a}_{1,1}$ as well as the torsion angle φ and its derivative $\varphi_{,1}$. Therefore, $\mathbf{g}$ consists of 8 components.

$$\mathbf{g} = (\underbrace{a_{1x}, a_{1y}, a_{1z}}_{\mathbf{a}_1}, \underbrace{a_{1,1x}, a_{1,1y}, a_{1,1z}}_{\mathbf{a}_{1,1}}, \varphi, \varphi_{,1}) \tag{5.70}$$

We now introduce an additional layer of intermediate results $\bar{\mathbf{g}}$, which consists of the base vectors $\mathbf{a}_1$, $\mathbf{a}_{1,1}$, $\mathbf{a}_{2,1}$ and $\mathbf{a}_{3,1}$, which extends the composition $\Pi = p \circ \mathbf{d} \circ \bar{\mathbf{g}} \circ \mathbf{g}$.

$$\bar{\mathbf{g}} = (\underbrace{a_{1x}, a_{1y}, a_{1z}}_{\mathbf{a}_1}, \underbrace{a_{1,1x}, a_{1,1y}, a_{1,1z}}_{\mathbf{a}_{1,1}}, \underbrace{a_{2,1x}, a_{2,1y}, a_{2,1z}}_{\mathbf{a}_{2,1}}, \underbrace{a_{3,1x}, a_{3,1y}, a_{3,1z}}_{\mathbf{a}_{3,1}}) \tag{5.71}$$

According to [28], the relation between $\mathbf{g}$ and $\bar{\mathbf{g}}$ contains a transformation of the reference base vectors using the Euler-Rodriguez formula. For a detailed explanation, please refer to the publication on the beam. This transformation renders the

[4] Incorrect indices from the original publication were corrected for the thesis.

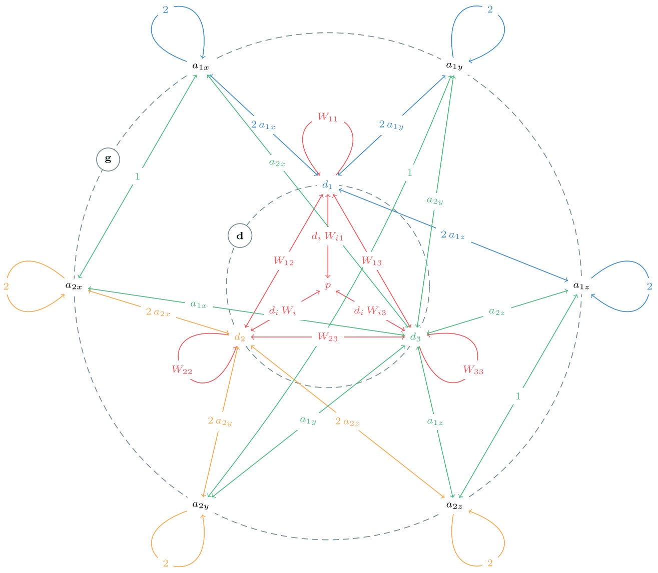


Figure 5.15: Computational graph of a membrane element. Suboperations: p (red), $d_1 = a_{11} - A_{11}$ (blue), $d_2 = a_{22} - A_{22}$ (orange), $d_3 = a_{12} - A_{12}$ (green).

computation of the derivatives very complex and leads to a dense computational graph. A detailed exploration of the graph can slightly increase the performance but leads to a more complex implementation. At this point, we have to make a trade-off between usability and performance. We therefore forego a manual analysis of the graph between these two layers and apply algorithmic differentiation with hyper-dual numbers to generate and evaluate these branches of the graph automatically.

For the remaining branches, we can use the same pattern as for the previous elements. In this case, **d** consists of three components arising from the bending deformation around the two axes of the cross-section (d_1 and d_2) and the deformation due to torsion (d_3).

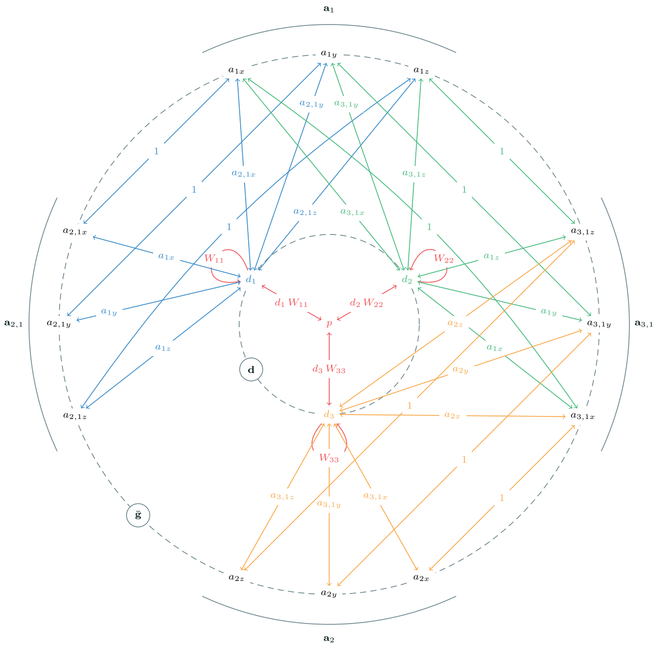


Figure 5.16: Computational graph of a beam element. Suboperations: p (red), $d_1 = b_2 - B_2$ (blue), $d_2 = b_3 - B_3$ (green), $d_3 = c_2 - C_2$ (orange).

$$\mathbf{d} = (\underbrace{b_2 - B_2}_{d_1}, \underbrace{b_3 - B_3}_{d_2}, \underbrace{c_2 - C_2}_{d_3}) \tag{5.72}$$

Here $b_\alpha = \mathbf{a}_{\alpha,1} \cdot \mathbf{a}_1$ and $c_2 = \mathbf{a}_{3,1} \cdot \mathbf{a}_2$. The material stiffness $\mathbf{D}$ depends on Young's modulus E, the shear modulus G, and the moments of inertia I, I_y and I_z.

The graph in Figure 5.16 represents the computation of $\bar{\mathbf{F}}$ and $\bar{\mathbf{K}}$. By applying the additional transformation from $\bar{\mathbf{g}}$ to $\mathbf{g}$, we obtain $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$. These results are then transformed according to Section 5.7 to compute $\mathbf{F}$ and $\mathbf{K}$ of the IGA beam element.

$$W_{ad} = T_{ab} D_{bc} T_{cd} \alpha \quad D_{ab} = \begin{pmatrix} EI_z & 0 & 0 \\ & EI_y & 0 \\ \text{sym.} & & GI \end{pmatrix} \quad T_{ab} = \begin{pmatrix} \frac{1}{A_{11}} & 0 & 0 \\ 0 & \frac{1}{A_{11}} & 0 \\ 0 & 0 & \frac{1}{\sqrt{A_{11}}} \end{pmatrix} \tag{5.73}$$ [5]

Shell

The nonlinear Kirchhoff-Love shell adds further bending terms to the membrane element. The extensional strain energy can be taken from the membrane element. To compute the bending and twisting energy, we need the base vectors $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_{1,1}$, $\mathbf{a}_{1,2}$ and $\mathbf{a}_{2,2}$. Thus $\mathbf{g}$ contains a total of 15 input parameters, while $\mathbf{d}$ comprises three components. We introduce an additional layer of parameters $\bar{\mathbf{g}}$ which contains $\mathbf{a}_3$, $\mathbf{a}_{1,1}$, $\mathbf{a}_{1,2}$ and $\mathbf{a}_{2,2}$.

$$\mathbf{g} = (\underbrace{a_{1x}, a_{1y}, a_{1z}}_{\mathbf{a}_1}, \underbrace{a_{2x}, a_{2y}, a_{2z}}_{\mathbf{a}_2}, \underbrace{a_{1,1x}, a_{1,1y}, a_{1,1z}}_{\mathbf{a}_{1,1}}, \underbrace{a_{1,2x}, a_{1,2y}, a_{1,2z}}_{\mathbf{a}_{1,2}}, \underbrace{a_{2,2x}, a_{2,2y}, a_{2,2z}}_{\mathbf{a}_{2,2}}) \tag{5.74}$$

$$\bar{\mathbf{g}} = (\underbrace{a_{3x}, a_{3y}, a_{3z}}_{\mathbf{a}_3}, \underbrace{a_{1,1x}, a_{1,1y}, a_{1,1z}}_{\mathbf{a}_{1,1}}, \underbrace{a_{1,2x}, a_{1,2y}, a_{1,2z}}_{\mathbf{a}_{1,2}}, \underbrace{a_{2,2x}, a_{2,2y}, a_{2,2z}}_{\mathbf{a}_{2,2}}) \tag{5.75}$$

$$\mathbf{d} = (\underbrace{b_{11} - B_{11}}_{d_1}, \underbrace{b_{22} - B_{22}}_{d_2}, \underbrace{b_{12} - B_{12}}_{d_3}) \tag{5.76}$$

Here $b_{\alpha\beta} = \mathbf{a}_{\alpha,\beta} \cdot \mathbf{a}_3$, where $\mathbf{a}_3$ is a unit vector perpendicular to $\mathbf{a}_1$ and $\mathbf{a}_2$. This relation connects the parameters of layer $\mathbf{g}$ and $\bar{\mathbf{g}}$.

$$\mathbf{a}_3 = \frac{\mathbf{a}_1 \times \mathbf{a}_2}{\|\mathbf{a}_1 \times \mathbf{a}_2\|}. \tag{5.77}$$

[5] Incorrect indices from the original publication were corrected for the thesis.

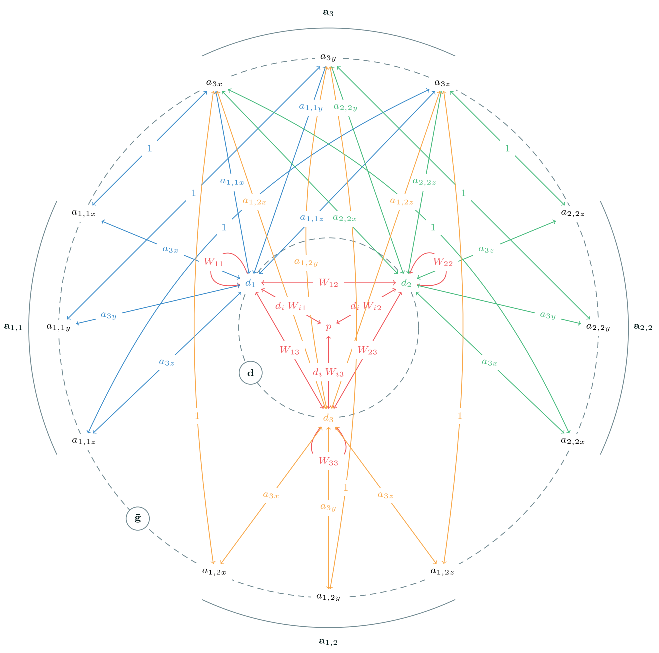


Figure 5.17: Computational graph of a shell element. Suboperations: p (red), $d_1 = a_{11} - A_{11}$ (blue), $d_2 = a_{22} - A_{22}$ (green), $d_3 = a_{12} - A_{12}$ (orange).

For the sake of simplicity, we use algorithmic differentiation with hyper-dual numbers to create the graph between $\bar{\mathbf{g}}$ and $\mathbf{g}$. For the remaining branches, we use the same pattern as for the other elements. To determine the required weights, we need the material stiffness for bending $\mathbf{D}$. The transformation matrix $\mathbf{T}$ is the same as for the membrane element.

Figure 5.17 shows the graph of the computation of $\bar{\mathbf{F}}$ and $\bar{\mathbf{K}}$. This corresponds to load and stiffness with respect to $\bar{\mathbf{g}}$. By transforming the results from $\bar{\mathbf{g}}$ to $\mathbf{g}$, we obtain $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$. Finally, we compute $\mathbf{F}$ and $\mathbf{K}$ for the IGA shell elements by applying the transformation coming from the geometric description (Section 5.7).

$$W_{ab} = \frac{1}{4}\, T_{ab}\, D_{ab}\, T_{ba}\, \alpha \qquad D_{ab} = \frac{E\,t^3}{12\,(1-\nu^2)} \begin{pmatrix} 1 & \nu & 0 \\ & 1 & 0 \\ \text{sym.} & & \frac{1-\nu}{2} \end{pmatrix}^{6} \tag{5.78}$$

Validation

In Section 5.5 it was shown analytically that AM leads to the same result as DM. To validate the implementation of the adjoint formulations, we compare the results with the research software Carat++, in which the isogeometric elements presented were previously implemented according to DM, using a classic manual approach. For validation, the load vector and stiffness matrix for the AM and DM were compared numerically for the two examples shown in Figure 5.22 and 5.23. Both produce the same results.

5.7 Evaluate Geometry

In this section, we consider different ways of calculating the basis vectors $\mathbf{a}_\alpha$ used in Section 5.6. We derive the partial derivatives for the individual cases. This allows us to transform $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ to different geometry descriptions according to (5.46) and (5.48).

A Simple Line

In this case, the base vector $\mathbf{a}_1$ is described as the connection of two points $\mathbf{P}_1 = (x_1, y_1, z_1)$ and $\mathbf{P}_2 = (x_2, y_2, z_2)$, as shown in Figure 5.14b. The control parameters are given by $\mathbf{x} = (x_1, y_1, z_1, x_2, y_2, z_2)$, while the base vector results from the difference between the coordinates.

$$\mathbf{a}_1 = (a_{1x}, a_{1y}, a_{1z}) = \mathbf{P}_2 - \mathbf{P}_1 = (x_2 - x_1, y_2 - y_1, z_2 - z_1) = \mathbf{g}(\mathbf{x}) \tag{5.79}$$

To perform the transformation into the geometry space according to Section 5.5 we need the partial derivatives of $\mathbf{g}$.

[6] Incorrect indices from the original publication were corrected for the thesis.

$$D\mathbf{g} = \left(\frac{\partial g_i}{\partial x_r}\right) = \begin{pmatrix} -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{pmatrix} \quad D^2\,\mathbf{g} = \left(\frac{\partial^2 g_i}{\partial x_r\,\partial x_s}\right) = \mathbf{0} \tag{5.80}$$

This enables us to transform $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ from Section 5.6 to $\mathbf{F}$ and $\mathbf{K}$ (Figure 5.14a to Figure 5.14b). For the length of influence α, we choose the real geometric distance between $\mathbf{P}_1$ and $\mathbf{P}_2$. This results in a classic truss element whose geometry is described by a simple line. As described in Section 5.4, we can use the sparse structure of the partial derivatives to eliminate paths. For example all second order paths vanish because all second order partial derivatives are zero. As a result, the number of operations is significantly reduced.

A Simple Triangle

As shown in Figure 5.14f, the base vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ define a triangle $(\mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3)$, such that $\mathbf{a}_1 = \mathbf{P}_2 - \mathbf{P}_1$ and $\mathbf{a}_2 = \mathbf{P}_3 - \mathbf{P}_1$. The coordinates of the points $\mathbf{P}$ are the control parameters $\mathbf{x} = (x_1, y_1, z_1, x_2, y_2, z_2, x_3, y_3, z_4)$.

$$D\mathbf{g} = \begin{pmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad D^2\,\mathbf{g} = \left(\frac{\partial^2 g_i}{\partial x_r\,\partial x_s}\right) = \mathbf{0} \tag{5.81}$$

The partial derivatives allow us to transform $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ from Section 5.6 to $\mathbf{F}$ and $\mathbf{K}$ of a classic triangular membrane element (Figure 5.14f to Figure 5.14e). The area of influence α is given by the area of the triangle.

Classic NURBS Geometries

In the simplest case, the shape of an isogeometric element is described by the NURBS function (5.18). The geometry of a cable is described by a NURBS curve

(Figure 5.14c). In this case, $\mathbf{x}$ consists of the coordinates of the control points $\bar{\mathbf{P}}_i$, where $|\bar{P}|$ depends on the polynomial degree.

$$\mathbf{x} = (x_1, y_1, z_1, x_2, y_2, z_2 \quad \dots \quad z_{|\bar{P}|}) \tag{5.82}$$

$$\mathbf{a}_1 = (a_{1x}, a_{1y}, a_{1z}) = R_{i,1}\,\mathbf{P}_i = \mathbf{g}(\mathbf{x}) \tag{5.83}$$

The partial derivatives

$$\begin{aligned} D\,\mathbf{a}_1 &= \begin{pmatrix} R_{1,1} & 0 & 0 & R_{2,1} & 0 & 0 & \dots & 0 \\ 0 & R_{1,1} & 0 & 0 & R_{2,1} & 0 & \dots & 0 \\ 0 & 0 & R_{1,1} & 0 & 0 & R_{2,1} & \dots & R_{|\bar{P}|,1} \end{pmatrix} \\ D^2\,\mathbf{a}_1 &= \mathbf{0} \end{aligned} \tag{5.84}$$

enable us to transform $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ from Section 5.6 to $\mathbf{F}$ and $\mathbf{K}$ of an IGA cable (Figure 5.14a to Figure 5.14c). For IGA elements, we perform numerical integration according to Section 5.2. We choose the length of influence α according to the weight of an integration point. Figure 5.18 shows the computational graph for the base vector $\mathbf{a}_1$. Computation of other basis vectors gives us the same graph but with different weights resulting from the corresponding basis functions. The extension to the second base vector $\mathbf{a}_2$ gives us the transformation for the IGA membrane element (Figure 5.14f to Figure 5.14d).

Embedded NURBS Geometries

As an example of embedded geometries, we consider a curve embedded into a surface. Embedded curves are used, for instance, for the description of edge cables [27]. The shape of the embedded curve $\mathbf{c}(\xi)$ results from the composition of a spatial NURBS surface $\mathbf{s}(\hat{\xi}, \hat{\eta})$ and a planar curve $\hat{\mathbf{c}}(\xi) = (\hat{\xi}, \hat{\eta})$ in the parameter space of $\mathbf{s}$ (Figure 5.19). According to (5.13) the shape functions $\hat{R}$ of the surface are evaluated for the parameters $\hat{\xi}, \hat{\eta}$. Therefore, $\mathbf{c}$ can be described as a linear combination of the basis functions $R_i = \hat{R}_i$ and the control points $\mathbf{P}_i$ of the surface.

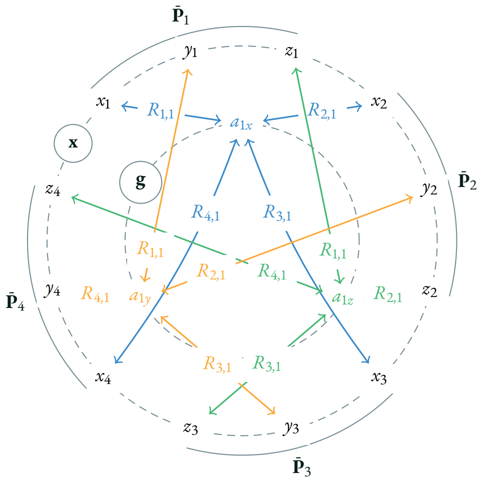


Figure 5.18: Computational graph of the base vector $\mathbf{a}_1$ of a NURBS geometry with four control points. The pattern of edges is repeated for each additional control point.

$$\begin{aligned} \mathbf{c} &= \mathbf{s} \circ (\hat{\xi}, \hat{\eta}) \\ &= \hat{R}_i(\hat{\xi}, \hat{\eta})\, \mathbf{P}_i \\ &= R_i(\xi)\, \mathbf{P}_i \end{aligned} \tag{5.85}$$

The base vector $\mathbf{a}_1$ which is needed for the formulation of an embedded cable is given by the first derivative of $\mathbf{c}$. It can be described by a linear combination of the first derivatives of the shape functions $R_{i,1} = \xi'\,\hat{R}_{i,1} + \eta'\,\hat{R}_{i,2}$ and the control points $\mathbf{P}_i$.

$$\begin{aligned} \mathbf{a}_1 = D\,\mathbf{c} &= \hat{\xi}'\,\hat{R}_{i,1}(\hat{\xi}, \hat{\eta})\,\mathbf{P}_i + \hat{\eta}'\,\hat{R}_{i,2}(\hat{\xi}, \hat{\eta})\,\mathbf{P}_i \\ &= (\hat{\xi}'\,\hat{R}_{i,1}(\hat{\xi}, \hat{\eta}) + \hat{\eta}'\,\hat{R}_{i,2}(\hat{\xi}, \hat{\eta}))\,\mathbf{P}_i \\ &= R_{i,1}(\xi)\,\mathbf{P}_i \end{aligned} \tag{5.86}$$

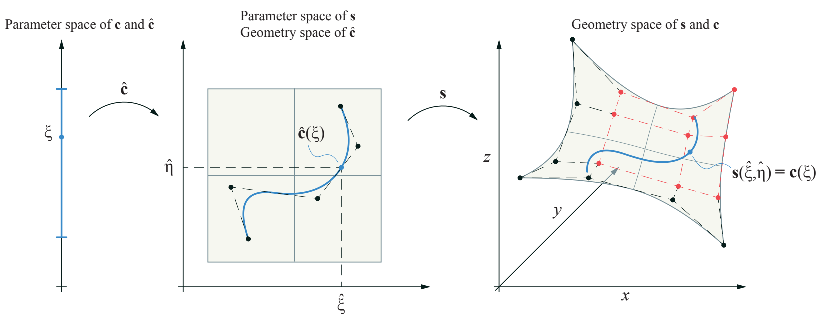


Figure 5.19: A curve (blue) embedded in a surface (gray). The curve parameter ξ is first transformed to the parameter space of the surface by $\hat{\mathbf{c}}$ and then to the geometry space by $\mathbf{s}$. The location of the point in geometry space is controlled by a subset of the the control points of the surface (red)

Computation of an embedded curve leads to the same computational graph as a classic NURBS curve (Figure 5.18). It is only necessary to change the weights according to the basis functions (5.86). Note that a more complex geometry description can result in a different computational graph. In such a case, it is not sufficient simply to replace the weights of the edges.

Now that we have the appropriate weights, we are able to transform $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ from Section 5.6 to $\mathbf{F}$ and $\mathbf{K}$ of an embedded IGA cable (Figure 5.14a to Figure 5.14d).

5.8 Coupling Conditions

We now return to the various coupling scenarios shown in Figure 5.1. The distance between two points $\mathbf{a}$ and $\mathbf{b}$ is given by the vector $\mathbf{d} = \mathbf{b} - \mathbf{a}$. To minimize the length of $\mathbf{d}$, we formulate the quadratic penalty functional according to Section 5.2.

$$\mathbf{g} = (\underbrace{x_a, y_a, z_a}_{\mathbf{a}}, \underbrace{x_b, y_b, z_b}_{\mathbf{b}}) \tag{5.87}$$

$$\mathbf{d} = (x_b - x_a, y_b - y_a, z_b - z_a) \tag{5.88}$$

$$P = \frac{1}{2}\, w\, (\mathbf{d} \cdot \mathbf{d}) \tag{5.89}$$

Note that (5.89) can be seen as a special case of (5.55), where $W_{ab} = w\,\delta_{ab}$ and δ_{ab} is the *Kronecker delta*. The partial derivatives are given by:

$$D\,P = \hat{\mathbf{F}} = w\,\begin{pmatrix} x_a - x_b & y_a - y_b & z_a - z_b & x_b - x_a & y_b - y_a & z_b - z_a \end{pmatrix} \tag{5.90}$$

$$D^2\,P = \hat{\mathbf{K}} = w \begin{pmatrix} 1 & 0 & 0 & -1 & 0 & 0 \\ & 1 & 0 & 0 & -1 & 0 \\ & & 1 & 0 & 0 & -1 \\ & & & 1 & 0 & 0 \\ & \text{sym.} & & & 1 & 0 \\ & & & & & 1 \end{pmatrix} \tag{5.91}$$

The corresponding computational graph is shown in Figure 5.20. This reveals an analogy to the *force density method* [69]. The force density method allows form finding of prestressed cable nets. The force density $q = S/L$ prescribes a constant ratio of internal force S and reference length L of the individual cables. If we choose $w = q$, then $\hat{\mathbf{F}}$ and $\hat{\mathbf{K}}$ correspond to the load vector and stiffness matrix of a force density element. This relation makes sense considering that the force density method searches for a minimum path network by minimizing the lengths of the individual cables.

Transfomation to $\mathbf{F}$ and $\mathbf{K}$ is performed by applying the chain rule according to the geometric description $\mathbf{g}(\mathbf{x})$. Various geometric descriptions of $\mathbf{a}$ and $\mathbf{b}$ are shown in Figure 5.1. Using different transformations in the geometry space, we can generate all the illustrated coupling scenarios from a simple force density element.

Minimizing the Distance Between Two Nodes

In this case, the coordinates of $\mathbf{a} = (x_a, y_a, z_a)$ and $\mathbf{b} = (x_b, y_b, z_b)$ are controlled explicitly (Figure 5.1a). The control parameters are given by $\mathbf{x} = (x_a, y_a, z_a, x_b, y_b, z_b)$. For the partial derivatives, we obtain $D g = \mathbf{I}$ and $D^2\,g = \mathbf{0}$, which eliminates the transformation from $\mathbf{g}$ to $\mathbf{x}$. This leads to $\mathbf{F} = \hat{\mathbf{F}}$ and $\mathbf{K} = \hat{\mathbf{K}}$, which corresponds to a classic computation using the force density method.

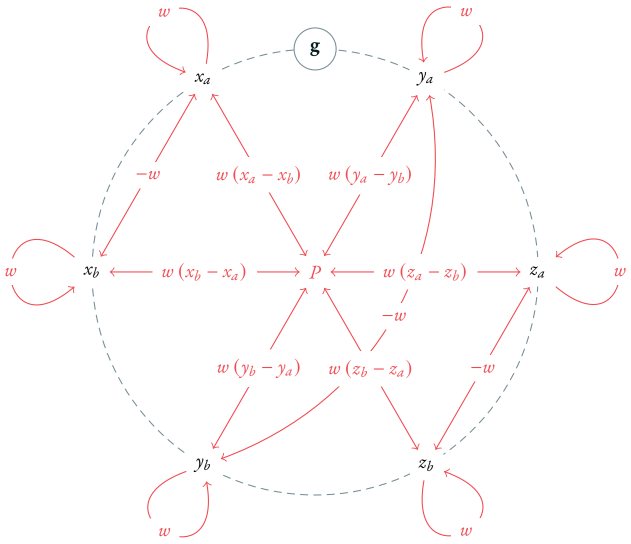


Figure 5.20: Computational graph of a force density element

Coupling of Two Points Embedded in NURBS

For the coupling, $\mathbf{a} = \mathbf{c}_a(t_a)$ and $\mathbf{b} = \mathbf{c}_b(t_b)$ are given implicitly by NURBS functions (Figure 5.1b). The location of **a** and **b** is controlled by the location of the control points $\mathbf{P}_i$. The partial derivatives of **g** are given by (5.84). A possible solution for **x** where $\mathbf{a} = \mathbf{b}$ is shown in Figure 5.21a. Such constraints can be used to couple trimmed NURBS geometries along the common edges.

Sliding Points

By choosing the curve parameters t_a and t_b as design parameters instead of the control points, we can implement sliding conditions [47] as shown in Figure 5.1c. For the geometric transformation, we compute the partial deivatives of **g** using FAD. A possible result is shown in Figure 5.21b. Note that in this case, the distance

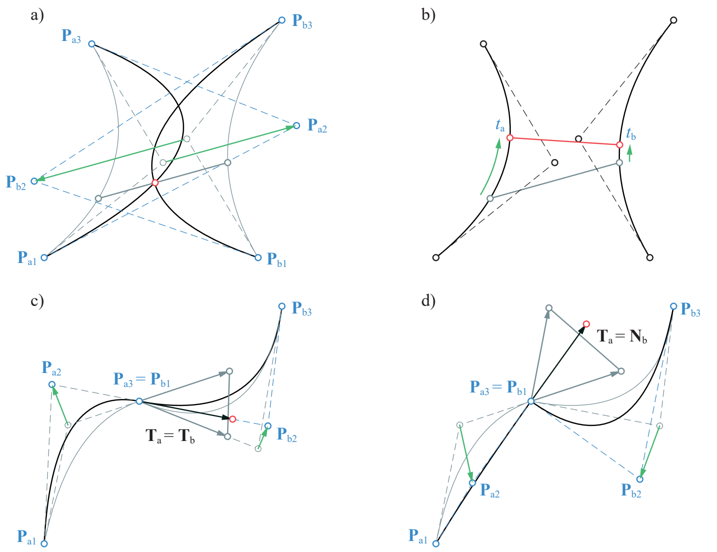


Figure 5.21: Different scenarios in which the distance between two points are minimized. Starting from an initial geometry (gray), the coupling constraints are not yet fulfilled, the control parameters (blue) are modified using the Newton-Raphson algorithm. The computed displacements (green) yield the final geometry (black), in which the distance between the two points is minimized (red).

between **a** and **b** is not zero but minimized as a least-square solution.

Coupling of Tangents and Normals

Instead of minimizing the distance between two points, it is also possible to minimize the difference between two vectors. This can also be done by transforming (5.90) and (5.91). In the case of Figure 5.1d, we write $\mathbf{a} = \mathbf{T}_a$ and $\mathbf{b} = \mathbf{T}_b$, where the tangents $\mathbf{T}_a$ and $\mathbf{T}_b$ are controlled by the locations of the control points $\mathbf{P}_i$. This can be used to achieve G1 continuity similar to [32, 34]. A possible solution is shown in Figure 5.21c. For the sake of simplicity, we use algorithmic differentiation with hyper-dual numbers to calculate the partial derivatives of **g**.

In the same way, it is possible to couple a tangent with a normal vector (Figure 5.1e).

In this case, we choose $\mathbf{a} = \mathbf{T}_a$ and $\mathbf{b} = \mathbf{N}_b$. A solution is shown in Figure 5.21d.

5.9 Comparison with a Classic Implementation

The adjoint method can be used in existing FE frameworks as a drop-in replacement. In this section we compare the performance of the adjoint method with that of an existing and established implementation within the research software Carat++. For this purposes, we will choose two examples using beam and shell elements.

Example 1 We perform a classic nonlinear structural analysis using an arc-shaped beam (Figure 5.22). The geometry of the arc corresponds to a semicircle with a radius $R = 5$. It is modeled with a NURBS curve of polynomial degree $p = 4$ and 34 control points. We choose a rectangular cross section with an area $A = 0.1 \times 0.1$. For the material parameters we choose $E = 21 \cdot 10^7$ and $G = E/2$. The fixed support (A) is rotated about the z-axis in each time step by $\pi/20 = 9°$ using displacement control. To model the clamping at each end and apply the rotation, we use vector couplings (see Section 5.8) with a weight of $w = 10^{12}$. By executing 40 time steps, we rotate (A) by a total of $360°$. The support on the right can now slide along the x-axis. We show the deformations for the rotation angles $0°$, $90°$, $180°$, $270°$, $315°$ and $360°$. The example is inspired from a blog post by Daniel Piker[7].

Example 2 We apply a discrete displacement field to transform a CAD model (Figure 5.23). The displacement field consists of of 8859 vectors and is the result of a shape optimization using *vertex morphing*. IGA displacement couplings are applied (weight $w_{\text{field}} = 1$) to transfer the displacements to the CAD geometry. The CAD geometry consists of 9 trimmed NURBS surfaces of polynomial degree $p = 4$ in each direction. To maintain geometric continuity at the edges, we apply coupling conditions to the displacements and normal vectors (weights $w_{\text{disp}} = 1$ and $w_{\text{norm}} = 10$). To keep the surfaces as smooth as possible, we apply elastic stiffness in the form of an IGA shell (thickness $t = 0.01$, $E = 1$, $\nu = 0$). To improve the conditioning of the problem, we apply a simple regularization term by adding a small constant value $\beta = 10^{-8}$ to the diagonal entries of the hessian [7]. In addition, we apply automatic weight scaling. Details of the methodology can be found in [70].

[7]https://www.grasshopper3d.com/group/kangaroo/forum/topics/fun-with-beams, accessed November 15, 2020

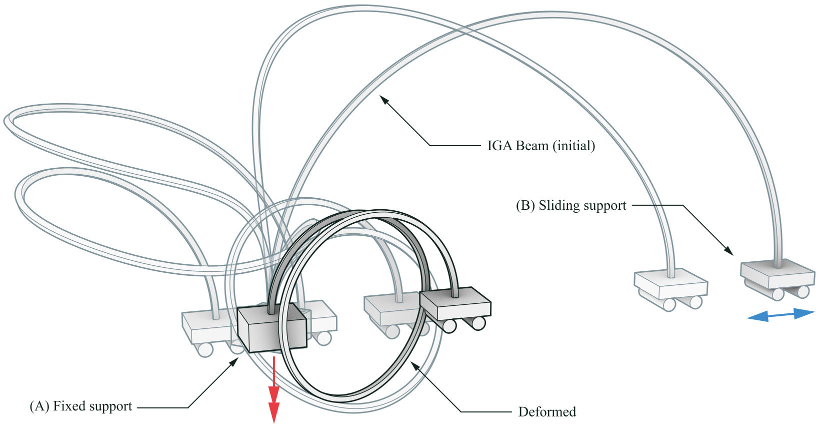


Figure 5.22: Benchmark using the IGA beam element. The rotation of (A) around the z-axis is increased continuously while (B) can slide along the x-axis.

Figure 5.23 shows the initial CAD geometry with the displacement field and the resulting geometry after three iterations.

To compare the performance, we vary the polynomial degree of the geometries and measure the duration for calculating **F** and **K** at a single integration point. Out of 10000 measurements we determine the arithmetic mean. As shown in Figure 5.24, the performance of the adjoint method is significally higher for the beam and for the shell. Note that no implementation was performed by professional software developers. However, the comparison shows that the adjoint method can lead to a significant increase in performance within a research code.

5.10 Conclusion and Outlook

For a consistent implementation of finite elements, the load vector and stiffness matrix are obtained from the first and second order derivatives of the energy functional. These derivatives can be computed using either the direct or adjoint method. In this paper, the two methods were compared for the implementation of various geometrical nonlinear isogeometric elements and coupling conditions using the penalty

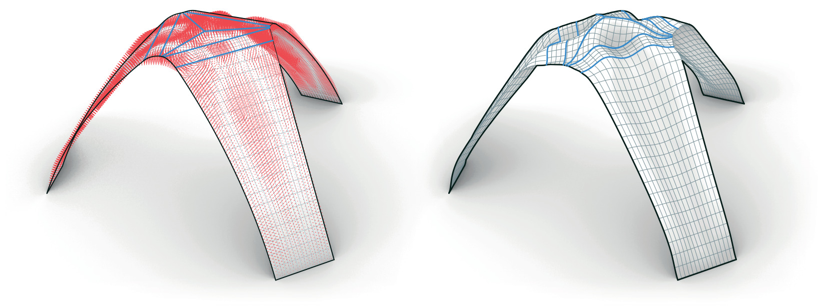


Figure 5.23: Benchmark using the IGA shell element. Coupling conditions are used to apply a displacement field to the geometry. At the same time, couplings ensure continuity at the edges.

method. The procedures of the direct and adjoint methods can be automated using algorithmic differentiation. This was further optimized by way of a detailed analysis of the computational graph for common mechanical and geometrical operations. The computational efforts were compared by analyzing the number of required operations. In addition, the performance of the adjoint method was compared with an existing, classic implementation of the direct method.

It turns out that the adjoint method has two advantages: (i) it leads to a core-congruential formulation that allows a clean separation into mechanical and geometrical components, and (ii) it reduces the influence of the polynomial degree of the geometry on the computational effort. This in turn improves the performance of the simulation. Both of these aspects are particularly important for CAD-integrated isogeometric analysis. The core-congruential formulation enables a modular implementation of IGA, in which the same mechanical properties are applied to different types of geometry. In this paper, we focus on linear and NURBS-based geometries. The technique can also be used to support additional geometry types, such as subdivision surfaces. The gain in performance may be used to achieve a higher level of interactivity for CAD-integrated analysis tools. This would be especially useful in early design stages, in which the influence of different parameters on the structure is studied. By extending the elements to account for prestress, the method may also show this advantage in the case of interactive form finding.

The method was applied to classic elastic curve and surface elements based on

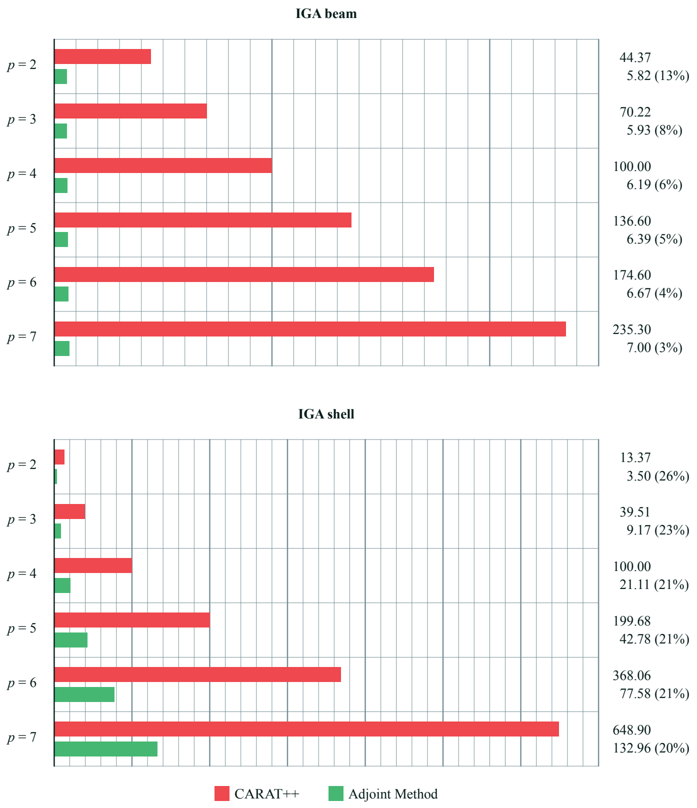


Figure 5.24: Comparison of the computation time for evaluating a single integration point of a beam and a shell element (less is better). We choose $p = 4$ as a reference value. Environment: MSVC 19.27, Windows 10.0.19041, Intel Xeon CPU E3-1280 v6, 3.90GHz, 1 CPU, 8 logical and 4 physical cores

Bernoulli and Kirchhoff-Love formulations. It should be found to be easily extendable to isogeometric solid elements or Reissner-Mindlin formulations based on energy functionals. In conjunction with a hierarchical approach [31], modularity can be further improved.

Acknowledgement

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the project “Integrierte Formfindung, Simulation und Optimierung mit der generalisierten Kraftdichtemethode auf der Grundlage gemischter Funktionale” (434336509) and by Bayerische Motoren Werke (BMW). The support is gratefully acknowledged.

6

Application in form-finding

The content of this chapter was previously published as: Oberbichler, T., Bletzinger, K.-U.: CAD-Integrated Form-Finding of Structural Membranes Using Extended Catmull–Clark Subdivision Surfaces. *Computer Methods in Applied Mechanics and Engineering*, 2022, https://doi.org/10.1016/j.cad.2022.103360

Acknowledgments

Extended Catmull-Clark subdivision surfaces are implemented within a CAD-integrated analysis workflow for form-finding by using the updated reference strategy. In this context, form-finding determines the shape of a structure that is in equilibrium for a predefined stress state. Following the principles of isogeometric analysis, the geometry for the form-finding is described by the control mesh of the subdivision surface. Using the subdivision scheme, the coarse control mesh is recursively refined to a smooth limit surface. The shape functions are obtained from the refinement process by applying algorithmic differentiation to the subdivision algorithm. This approach also supports sharp features like creases and corners. In this way, the modeling functionalities of common CAD environments are covered which allows seamless integration of the analysis into the CAD system. A core-congruential element formulation enables the efficient combination of mechanical properties with different geometric discretizations. This allows an existing framework to be extended easily with subdivision surfaces. Within this work, form-finding is applied to prestressed cable and membrane structures. Weak boundary conditions allow the modeling of oriented edge supports. The CAD-integrated implementation within the framework of a visual programming language is outlined. Using selected numerical examples, the method is demonstrated and verified.

6.1 Introduction

Membranes and cables allow the construction of impressive and at the same time lightweight structures. A fundamental characteristic of such designs is the special load transfer. The load is absorbed by tensile forces alone without any compression or bending forces, which allows a particularly slender design. These properties result in a strict relationship between form and forces. Inevitably, the design of membrane structures must take into account mechanical considerations. Form-finding becomes an integral part of the design process. The goal is to find an initially unknown geometry that is in equilibrium for a predefined internal stress state. Mechanical and geometrical conditions must be considered simultaneously. Due to the strong interaction, several iteration loops are usually necessary to find an appropriate design. This process can be enhanced by an interactive CAD-integrated implementation of form-finding.

The interactions of form and force are reflected in the digital tools: *computer-aided design* (CAD) is used to model the geometry while *computer-aided engineering* (CAE) is used to model the mechanical behavior usually based on the *finite element method* (FEM). For closer integration of these disciplines, *isogeometric analysis* (IGA) is a useful tool. The geometric description of the CAD is taken over for the analysis by FEM, which enables a fluent exchange of inputs and outputs. This approach is ideal for integrating mechanical analysis directly into CAD environments and accelerating the design process of mechanical structures. The *Analysis in CAD* (AiCAD) philosophy was introduced in [32, 27]. The focus was on the use of *non-uniform rational B-Splines* (NURBS) which are widely used in CAD for the modeling of free-form surfaces. More recently, *subdivision surfaces* (SubDs) have become more important in the field of architectural design and are supported by common CAD packages. CAD-integrated analysis tools must therefore be able to support this type of geometry.

This paper presents a concept to implement form-finding for combined membrane and cable structures based on Catmull-Clark surfaces within a CAD-integrated IGA environment. A comprehensive toolbox for CAD-integrated structural analysis is obtained from the combination of different mechanical elements (e.g., cables, membranes, beams, shells) and geometric constraints (e.g., support and coupling conditions) with various geometric discretizations (e.g., meshes, NURBS, or SubDs) and by using different analysis types (e.g., structural analysis, form-finding). To minimize the implementation effort, a modular approach is proposed which reuses existing components from NURBS-based IGA. The *updated reference strategy* is used for the form-finding. It is based on continuum mechanics and can be implemented within a classical finite element workflow. In [71], the authors describe a modular and computationally efficient framework for IGA by combining mechanical and geometric building blocks. Within this paper, this concept is extended by two aspects: (i) the extension of the mechanical building blocks (core elements) for prestressed structures which enables form-finding, and (ii) new geometric building blocks (geometric transformation) to discretize structures by extended Catmull-Clark SubDs.

In combination with the existing blocks, this results in a wide range of applications. Figure 6.1 shows the form-finding of a simple tent structure based on trimmed NURBS multipatches and SubDs. In both cases, the same core elements for membranes and cables are combined with the corresponding geometric descriptions for surfaces and edges.

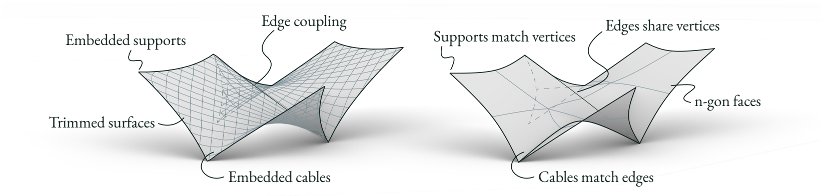


Figure 6.1: A membrane structure modelled with trimmed NURBS multipaches (left) and SubDs (right).

The implementation takes advantage of *algorithmic differentiation* (AD). AD is applied to the energy functional to compute the required forces and stiffnesses. The same technique is used to determine the shape functions for the SubDs and simplifies the implementation process for extended Catmull-Clark surfaces.

Section 6.2 provides an overview of the existing work related to isogeometric form-finding. The mechanically-driven process of form-finding is described in Section 6.3. This section introduces the prestressed core elements for cables and membranes and conditions for couplings and oriented supports. In Section 6.4, the geometric transformation is discussed which allows the transfer of the mechanical properties to SubDs. The extended subdivision rules are described and the computation of the required shape functions and the numerical integration process are demonstrated. Section 6.5 provides an overview of the proposed CAD-integrated analysis workflow. Within Section 6.6 the method is demonstrated and verified by several examples.

6.2 Related research

Klaus Linkwitz and Hans-Jörg Scheck provided the foundation for numerical form-finding of prestressed cable structures [72, 69] based on the force density method (FDM). The method was used to design the roof structure of the Olympic Stadium in Munich, in 1972. The FDM was generalized to the *updated reference strategy* (URS) by identifying the similarity between *force density* and the *second Piola-Kirchhoff stress* [73]. This allows the consistent extension to membrane structures. The geometry of the membranes and cables was described by line segments and triangles. The principles of FDM and URS are explained in Section 6.3.

URS was transferred to smooth NURBS-based geometries [27], based on the idea of *isogeometric analysis* (IGA). NURBS do not generally interpolate the control points, which is why boundary and coupling conditions usually cannot be modeled directly via the FE nodes. The *isogeometric B-Rep analysis* (IBRA), as introduced by [32], solves this problem by integrating trimming and weak coupling conditions. Mechanical properties are applied to points and curves which are embedded on the surface to model supports, coupling conditions, and edge cables. In [71], a modular and efficient framework for the implementation of CAD-integrated IGA was introduced. The usage of a core-congruential formulation [50, 49] allows the separation of the element formulation into a mechanical core element and a geometrical transformation. Mechanical properties can be applied to discrete meshes, NURBS, and other discretizations in a modular way.

The Catmull-Clark algorithm is a subdivision scheme introduced by Edwin Catmull and Jim Clark in 1978 [74]. Other than the Loop scheme [75], it is the most commonly used subdivision scheme and is available in a number of geometric modeling systems for architectural and engineering design. It represents the generalization of the subdivision process of uniform B-splines to control meshes with arbitrary topology. This refinement is an iterative process. In theory, an infinite number of refinement steps are necessary to obtain the smooth limit surface (Figure 6.2). To address this practical issue, [76] proposes an elegant approach by using an eigenvalue decomposition of the subdivision matrix. However, this approach requires a control mesh where each face is quadrilateral and contains only one extraordinary vertex. The classical Catmull-Clark subdivision scheme was extended to support boundary edges, sharp creases, and corners. In this work, a subset of the extensions from [77] is used which is currently supported by the CAD tool *Rhino*[1].

Loop subdivision surfaces were used to describe the geometry of finite shell elements in [78, 79, 80]. The support of extended Loop subdivision surfaces was shown in [81]. Isogeometric structural analysis based on Catmull-Clark subdivision was performed for solids [82] and shells [83]. A comprehensive compendium of isogeometric analysis based on SubDs is provided by [84, 85]. An advanced numerical integration scheme was presented in [86]. The generation of minimal subdivision surfaces was studied for the Loop [87, 88, 89] and the Catmull-Clark scheme [90, 91, 92]. In this context, minimum surfaces are generated from geometric considerations by iteratively minimizing the mean curvature flow.

[1]Rhinoceros 3D by Robert McNeel & Associates. Version 7 SR17 for Windows. https://www.rhino3d.com

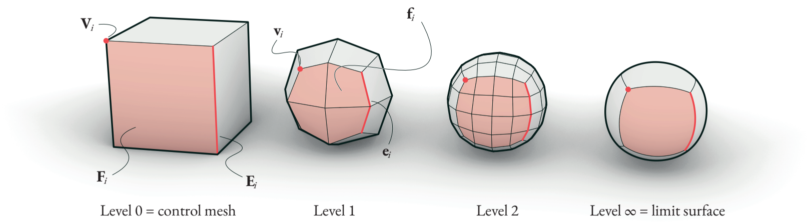


Figure 6.2: Refining a cube with the Catmull-Clark subdivision scheme.

For the CAD-integration of numerical analysis, there are two main approaches: a classic pre- and post-processing toolchain and a fully CAD-integrated approach. Pre- and post-processors that are embedded in the CAD are available for commercial analysis tools e.g., *Sofistik*[2] and *RFEM*[3]. This approach was also used for *TeDA/Kiwi!3d*[4] which acts as a pre-/post-processor for the NURBS-based IGA solver CARAT++[4]. This has the advantage, that an existing analysis package can be used. The data interface is usually realized by writing input files and reading output files. While it is easy to exchange data using such an interface, it is not possible to exchange functionalities. Therefore it is necessary to re-implement geometric algorithms and CAD data structures within the analysis tool and vice-versa. This approach allows the creation of parametric input files similar to text-based tools. However, there is no way to interact with the analysis process. Each parameter change results in a new input file and starts a new analysis. By contrast, fully CAD-integrated tools such as *Kangaroo*[5] allow direct interaction with the mechanical model. It is possible to observe and interact with the convergence process in real-time. The complete analysis process is integrated into the CAD environment. This simplifies data exchange and allows the use of the advanced geometric CAD algorithms.

[2]www.sofistik.com/products/finite-elements/rhinoceros-interface (access: 15.11.2021)

[3]www.dlubal.com/en/solutions/application-areas/building-information-modeling-bim/rfem-and-rhinoceros-grasshopper (access: 15.11.2021)

[4]www.cee.ed.tum.de/en/st/software (access: 15.11.2021)

[5]kangaroo3d.com (access: 15.11.2021)

6.3 Form-finding

Form-finding is an essential part of the design of lightweight structures. In classical structural analysis, the internal stress state results from the calculated deformations, while in form-finding, the final stress state is prescribed. The goal is to find a geometry that is balanced for this specific stress state. The following sections describe the basics of the force density method and its extension to the updated reference strategy, which is used for the design of cable and membrane structures.

Vectors, matrices, and tensors are denoted with bold letters. Their components are indicated in italics with the corresponding index. An n-dimensional vector $\mathbf{v}$ or an $n \times m$-dimensional matrix $\mathbf{M}$ is written as:

$$\mathbf{v} = \left(v_i\right) = \begin{pmatrix} v_1 & v_2 & \dots & v_n \end{pmatrix} \qquad \mathbf{M} = \left(M_{ij}\right) = \begin{pmatrix} M_{11} & \dots & M_{1m} \\ \vdots & \ddots & \vdots \\ M_{n1} & \dots & M_{nm} \end{pmatrix}$$

Einstein notation is used to express the summation of tensors, *Spivak's notation* for derivatives.

Force density method

The principle of the force density method is explained by a simple example shown in Figure 6.3a. The planar system consists of four nodes which are connected by three cables. Nodes 1-3 are fixed. Their position is denoted by $\mathbf{X}_i$. Node 0 is located at $\mathbf{X}_0$ and can move in both directions. The length of a cable i is denoted by L_i and the prescribed internal force by F_i. The sum of the forces acting on the free node results in the residual force $\mathbf{r} \neq \mathbf{0}$. Form and forces are not in equilibrium. This can be illustrated by a force diagram in which the forces are accumulated visually (Figure 6.3b). The system should be balanced by moving the free node as shown in Figure 6.3c to a position $\hat{\mathbf{x}}_0$ so that $\mathbf{r}(\mathbf{x}_0) = \mathbf{0}$. In the deflected configuration, the position of the node is denoted by $\mathbf{x}_0$ while the location of the supports is given by $\mathbf{x}_i = \mathbf{X}_i$. The lengths of the cables result accordingly:

$$l_i = |\mathbf{x}_0 - \mathbf{x}_i| \tag{6.1}$$

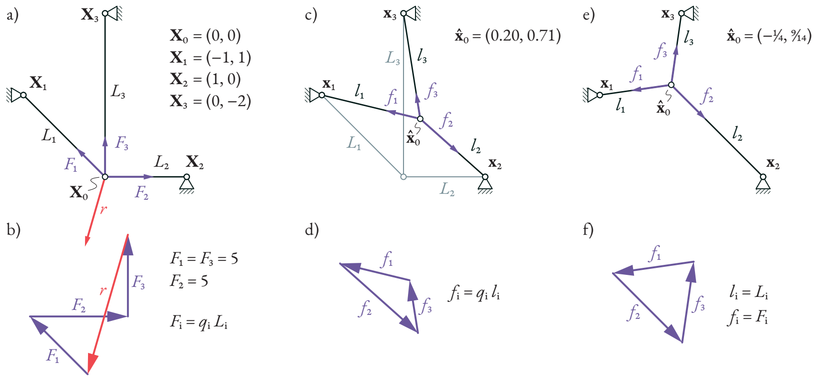


Figure 6.3: For FDM, the force densities q_i are specified for each member. A reference length L_i is required to determine q_i from prescribed internal forces F_i. The initial configuration (a) is not in equilibrium as the residual force does not vanish (b). The FDM provides a solution in equilibrium (c and d) where the prescribed force densities are maintained, but the internal forces f_i are different from F_i. URS provides a state of equilibrium in which the internal forces f_i correspond to the prescribed forces F_i (e and f).

The internal force in each member is denoted by f_i. The forces acting on the free node are accumulated to the residual force $\mathbf{r}$. A linear system of equations is obtained by introducing the force density $q_i = f_i / l_i$ as a constant ratio.

$$\mathbf{r}(\mathbf{x}_0) = \sum_{i=1}^{3} f_i \, \frac{\mathbf{x}_0 - \mathbf{x}_i}{l_i} = \sum_{i=1}^{3} q_i \, (\mathbf{x}_0 - \mathbf{x}_i) \tag{6.2}$$

Equilibrium of forces is given when $\mathbf{r} = \mathbf{0}$. Starting at a initial guess for $\mathbf{x}_0$, the position $\hat{\mathbf{x}}_0$ of the free node for which this equilibrium condition is satisfied can be determined directly:

$$\hat{\mathbf{x}}_0 = \mathbf{x}_0 - \mathbf{K}^{-1}\, \mathbf{r}(\mathbf{x}_0) \tag{6.3}$$

$$\mathbf{K} = D\,\mathbf{r} = \sum_{i=1}^{3} \begin{pmatrix} q_i & 0 \\ 0 & q_i \end{pmatrix} \tag{6.4}$$

In this configuration, the forces are in equilibrium. The internal force acting in each member is given by $f_i = q_i\, l_i$ (Figure 6.3d). The matrix $\mathbf{K}$ is the Jacobian of $\mathbf{r}$ and contains the partial derivatives with respect to the degrees of freedom. Since $\mathbf{r}$ represents a linear system of equations, $\mathbf{K}$ is constant. As a consequence, the solution is independent of the initial guess $\mathbf{x}_0$. Only the location of the supports, the force densities, and the topology of the structure are required for the solution.

The solution of the FDM can be considered as a cable net with a minimum total length. This relationship becomes apparent when considering the corresponding optimization problem. The lengths l_i are accumulated into a single objective function Π using the penalty method. The contribution of the individual members is weighted according to the force density:

$$\Pi = \sum_{i=1}^{3} \frac{1}{2}\, q_i\, l_i^2 \tag{6.5}$$

Considering the physical units, Π can be interpreted mechanically as work or energy. The derivative of Π with respect to the degrees of freedom corresponds to (6.2). The procedure can be applied to an arbitrary number of elements and degrees of freedom. At $\hat{\mathbf{x}}_0$ the gradient $D\,\Pi = \mathbf{r}$ is zero while the Hessian $D^2\,\Pi = \mathbf{K}$ is positive definite. The solution is therefore a minimum of Π.

By introducing force densities, a linear system is obtained which can be solved in a reliable way. To determine the force density from a prescribed internal force f_i, the length l_i in the equilibrium state would already have to be known. Instead, some other reference length must be selected. By using, e.g., the initial configuration ($\mathbf{x}_0 = \mathbf{X}_0$), the force densities are $q_i = F_i / L_i$. Since the force density remains constant, a change in length must result in a change in forces which is defined by the ratio of l_i and L_i:

$$f_i = q_i\, l_i = F_i\, \frac{l_i}{L_i} \tag{6.6}$$

Therefore, the specified forces are only achieved if the length of the solution has already been used as a reference length. Since this length depends on the unknown solution, the force density must be adjusted iteratively. The updated reference strategy derives the distinction between reference and actual configurations using continuum mechanics and provides a generalized update scheme that can also applied to membranes.

Updated reference strategy

The system from Figure 6.3 is modeled with prestressed cable elements. A structure in equilibrium is characterized by a local minimum of the total energy. The internal energy Π_i of each member i is given by:

$$\Pi_i = \frac{1}{2}\,\sigma_{\mathrm{PK2},i}\,\varepsilon_{\mathrm{GL},i}\,A_i\,L_i + \bar{\sigma}_{\mathrm{PK2},i}\,\varepsilon_{\mathrm{GL},i}\,A_i\,L_i$$
$$\text{with}\quad \sigma_{\mathrm{PK2},i} = \varepsilon_{\mathrm{GL},i}\,E_i, \qquad \varepsilon_{\mathrm{GL,i}} = \frac{l_i^2 - L_i^2}{2\,L_i^2} \tag{6.7}$$

where E is Young's modulus, A the cross-sectional area, $\varepsilon_{\mathrm{GL}}$ the *Green-Lagrange strain*, σ_{PK2} the *second Piola-Kirchhoff stress*, and $\bar{\sigma}_{\mathrm{PK2}}$ the prestress in each member. Neglecting the elastic energy by setting $E_i = 0$ and accumulating the energy of each member results in the total energy of the prestressed system:

$$\Pi = \sum_{i=1}^{3} \bar{\sigma}_{\mathrm{PK2},i}\,\varepsilon_{\mathrm{GL},i}\,A_i\,L_i \tag{6.8}$$

The gradient of the energy with respect to the degrees of freedom provides the residual force $\mathbf{r}$. From the comparison with (6.2), the equivalence with the FDM and the analogy of force density and PK2 stress becomes apparent:

$$D\Pi = \mathbf{r} = \sum_{i=1}^{3} \underbrace{\frac{\bar{\sigma}_{\mathrm{PK2},i}\,A_i}{L_i}}_{q_i}\,(\mathbf{x}_0 - \mathbf{X}_i) \tag{6.9}$$

FDM is equivalent to an FEM model with the elastic stiffness $E = 0$ and the prestress $\bar{\sigma}_{\mathrm{PK2}} = q\,L/A$. Strains and stresses can be measured in different reference systems. Green-Lagrange strains and second Piola-Kirchhoff stress are defined in the undeformed (reference) configuration, while *Euler-Almansi strains* and *Cauchy stresses* σ_c are defined in the deformed (actual) configuration. The relation is given by:

$$q_i = \frac{\bar{\sigma}_{\mathrm{PK2},i}\, A_i}{L_i} = \frac{\bar{\sigma}_{\mathrm{c},i}\, a_i}{l_i} \tag{6.10}$$

where a is the cross-sectional area in the deformed configuration. Assuming a constant cross-section ($A_i = a_i$), this relation corresponds to (6.6). For form-finding, the Cauchy stresses should be prescribed. Therefore, (6.9) can be adapted by (6.10). The corresponding stiffness matrix is no longer constant and several Newton iterations are required for the solution. Since the discrete solution of a form-finding problem is usually not unique [93], the stiffness matrix might become singular. Additional conditions are required to regularize the problem, e.g., additional geometric constraints. URS proposes a superposition of the Cauchy problem with the PK2 problem using a homotopy factor λ which results in the modified problem:

$$\mathbf{r}_{\mathrm{mod}} = \lambda\, \mathbf{r}_{\mathrm{PK2}} + (1 - \lambda)\, \mathbf{r}_{\mathrm{c}} \tag{6.11}$$

where the PK2 problem $\mathbf{r}_{\mathrm{PK2}}$ acts as a regularization whose influence can be controlled by λ. In the context of this work the PK2 problem is used exclusively ($\lambda = 1$). This procedure has already been successfully applied in [93, 94] for the form-finding of membrane structures.

After each iteration, the reference lengths L_i of the PK2 problem are updated with the current lengths l_i. This way, the solution converges to the desired configuration at which $L_i = l_i$ and $\sigma_{\mathrm{PK2}} = \sigma_{\mathrm{c}}$. The solution obtained by URS after 20 iterations is shown in Figure 6.3e. The forces f_i are equal to the prescribed F_i (Figure 6.3f).

Due to the consistent mechanical derivation, URS allows a generalization of FDM for arbitrary element types, e.g., membranes. The corresponding formulation can be derived from the energy functional.

Energy functional

The functional Π which evaluates the internal energy of the structure can be obtained from the *Hu-Washizu functional* [95]. In this paper, the components originating from elastic deformation and prestress are considered. Extension to the additional terms such as external loads can be done analogously.

$$\Pi = \int_{\Omega} \underbrace{\frac{1}{2}\,(\mathbf{S} : \mathbf{E})}_{\text{elastic}} + \underbrace{(\bar{\mathbf{S}} : \mathbf{E})}_{\text{prestress}}\, \mathrm{d}\Omega \qquad \text{with} \quad \mathbf{S} = \mathbb{C} : \mathbf{E} \tag{6.12}$$

where **E**, **S**, $\bar{\mathbf{S}}$, and $\mathbb{C}$ are tensors representing the Green-Lagrange strain, second Piola-Kirchhoff stress, prestress, and the constitutive tensor. The shape of these tensors depends on the element type. A *St. Venant Kirchhoff material* is assumed. The *domain* of the structure is denoted by Ω.

The strains result from a comparison between the deformed and undeformed geometry and can be computed from the base vectors. The base vectors are the derivatives of the geometry at a specific point. Using FEM, the geometry is controlled by a set of *degrees of freedom* (DOFs). In the case of IGA, these are the location of the control points of the NURBS or SubDs.

The numerical evaluation of the integral in (6.12) is done by using an n-point quadrature scheme that approximates the integral by a sum. The domain Ω is decomposed according to the integration points into subdomains Ω_i with energy Π_i:

$$\Pi \approx \sum_{i=1}^{n} \underbrace{\left(\frac{1}{2}\,\mathbf{S}_i : \mathbf{E}_i + \bar{\mathbf{S}}_i : \mathbf{E}_i\right)\Omega_i}_{\Pi_i \text{ at integration point } i} \qquad \text{where} \quad \sum_{i=1}^{n} \Omega_i \approx \Omega \tag{6.13}$$

Quadrature points, which are influenced by the same DOFs, are combined within finite elements. This way the whole system is decomposed into a set of finite elements where each element consists of a variable list of quadrature points. To minimize the energy using the Newton-Raphson algorithm, gradient and Hessian of Π_i are required which correspond to the residual force **r** and stiffness **K** at the quadrature point.

Core-congruential formulation

This section focuses on the efficient computation of the derivatives of the energy functional Π_i at a specific evaluation point. The evaluation of Π_i can be decomposed into a concatenation of functions. For the sake of clarity, the index i is omitted in the following:

$$\Pi(\mathbf{x}) = (p \circ \mathbf{e} \circ \mathbf{g})(\mathbf{x}) \tag{6.14}$$

where $\mathbf{g}$ evaluates the base vectors of the deformed geometry based on the location of the control points $\mathbf{x}$, $\mathbf{e}$ evaluates the strains based on the base vectors and p evaluates the energy based on the strains. According to the chain rule, the first derivative results from the product of the individual Jacobian matrices. The product is associative, i.e., the order of operations does not matter:

$$D\Pi = Dp \cdot \underbrace{(D\mathbf{e} \cdot D\mathbf{g})}_{\text{B-Matrix}} = \underbrace{(Dp \cdot D\mathbf{e})}_{\text{Core-residual}} \cdot D\mathbf{g} \tag{6.15}$$

Multiplying from right to left gives the *B-matrix* as an intermediate result. It contains the derivatives of the strain with respect to the degrees of freedom. When multiplying from left to right, the intermediate results correspond to the derivatives of the energy with respect to the base vectors. Mechanically, this can be interpreted as the residual force of a core element. Considering the chain rule, the stiffness matrix can be evaluated in the same way to obtain the 'core stiffness'. According to [50, 49] this corresponds to a *core-congruential element formulation* (CCF).

For CAD-integrated IGA, there are three important benefits resulting from the CCF: (i) The core element is independend of the geometric discretization $\mathbf{g}$. By exchanging $D\mathbf{g}$, the same core element can be used for different geometry types without modifications. (ii) The size of the B-Matrix depends on the number of control-points while the sizes of core residual force and core stiffness are constant. Therefore CCF reduces dynamic memory allocation. This is especially useful when using programming languages with automatic memory management which are very common for the implementation of CAD plugins. (iii) The order of operations in (6.15) is not important for the result but has a big influence on the computational

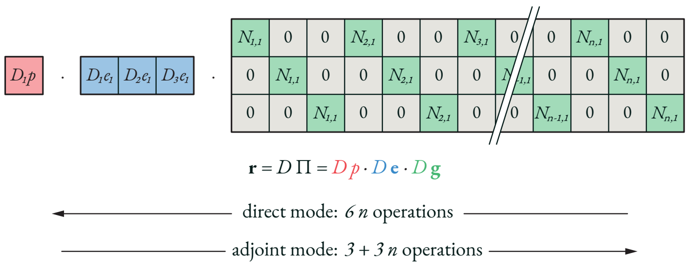

D_1p
D_1e_1 D_2e_1 D_3e_1
$N_{1,1}$ 0 0 $N_{2,1}$ 0 0 $N_{3,1}$ 0 0 $N_{n,1}$ 0 0
0 $N_{1,1}$ 0 0 $N_{2,1}$ 0 0 $N_{-1,1}$ 0 0 $N_{n,1}$ 0
0 0 $N_{1,1}$ 0 0 $N_{2,1}$ 0 0 $N_{n-1,1}$ 0 0 $N_{n,1}$
$\mathbf{r} = D\,\Pi = D\,p \cdot D\,\mathbf{e} \cdot D\,\mathbf{g}$
direct mode: $6\,n$ operations
adjoint mode: $3 + 3\,n$ operations

Figure 6.4: Direct and adjoint mode to compute the residual force of an IGA truss element with n nodes. The residuum is given by the product of the Jacobians of the individual functions. If $\mathbf{g}$ is a linear combination, then $D\,\mathbf{g}$ consists of the derivatives of the shape functions $N_{i,1}$.

effort. The CCF corresponds to the adjoint or backward mode of evaluating derivatives, while the approach via the B-matrix corresponds to the direct or forward mode. For functions with many inputs and few outputs, the adjoint mode is usually more efficient. This is the case for the energy functional which computes a scalar output depending on many control points. For simple FE elements which are described by a few nodes, this effect can be neglected. In the case of IGA, the number of control points is usually higher. Figure 6.4 shows the difference for the computation of the residual force for a simple IGA truss with n nodes. The adjoint mode only requires $(1 + n)/2n$ of the addition and multiplication operations that are required by the direct mode. The difference becomes even higher when calculating the stiffness matrix. A more detailed analysis is provided in [71].

The core element can be interpreted mechanically. For a cable, it corresponds to a simple linear truss element where the geometry is defined by the base vector $\mathbf{a}_1$ as shown in Figure 6.5. For IGA, the core elements are positioned at the integration points tangential to the smooth curve. The domain Ω_i results from the integration weight and the cross-sectional area. Residual force and stiffness are transformed to the control points of the smooth geometry according to the geometric parametrization $\mathbf{g}$. If $\mathbf{g}$ is a linear combination of the control points, then only the Jacobian $D\,\mathbf{g}$ is required to perform this transformation. In this case, the entries of $D\,\mathbf{g}$ correspond to the derivatives of the shape functions of the geometry.

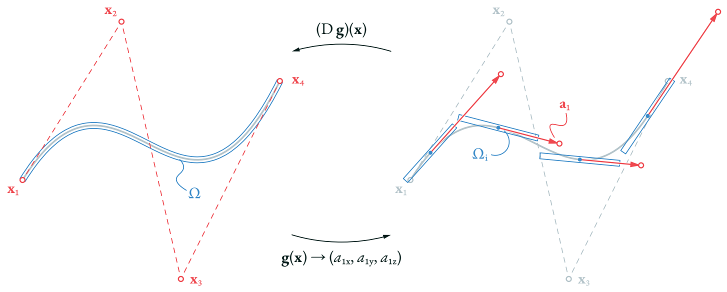


Figure 6.5: Transformation between control mesh (left) and core element (right).

Algorithmic differentiation

For the mechanical computation, the derivatives of the energy functional are required. *Algorithmic differentiation* (AD) is a set of techniques for transforming an algorithm that evaluates a numerical model into an algorithm that evaluates the numerical derivatives of the model with respect to specific design parameters [57]. In this work, a multivariate extension of *hyper-dual numbers* [14] is used for the algorithmic computation of the required derivatives. Similar to complex numbers, hyper-dual numbers consist not only of a real part but have additional dual components. They can be used to represent the coefficients of a 2nd-order Taylor series which are equivalent to the partial derivatives of a function. The multivariate extension allows the computation of derivatives of multivariate functions.

Operations with hyper-dual numbers take the chain rule into account. The result contains not only a scalar result but the Taylor series expansion about the evaluation point. By initializing the parameters of a function with hyper-dual numbers and performing a calculation (e.g., computing the energy), the result will also be a hyper-dual number that contains the derivatives w.r.t. the input parameters. With proper implementation, an algorithm can be used for the calculation of the derivatives just by exchanging the data type. This simplifies the development of IGA elements since only the energy functional has to be implemented.

The implementation of hyper-dual numbers is done by operator overloading. Classical mathematical operations are extended to apply the chain rule while computing the result. This corresponds to the direct mode of evaluating derivatives. Direct mode is easy to implement and intuitive to handle. However, it was shown in

Section 6.3 that it is less efficient than the adjoint mode when implementing IGA elements. As a trade-off between efficiency and usability, the direct mode is used to compute the core element and the geometry transformation separately. In the second step, the results are combined using the adjoint mode.

Besides the basic mathematical operations (e.g., addition, multiplication,...) additional operators are identified and implemented which are required frequently during the evaluation of the energy functional. Despite the scalar product and the squared vector norm, the function $p(\mathbf{d})$ is implemented and optimized:

$$p = V_a \, d_a + \frac{1}{2} W_{ab} \, d_a \, d_b \qquad \text{with} \quad W_{ab} = W_{ba} \tag{6.16}$$

where $\mathbf{V}$ is a constant vector, $\mathbf{W}$ is a constant symmetric matrix, and $\mathbf{d}$ is a vector that usually measures a deviation between an actual and a reference state. The implementation of the operator for hyper-dual numbers also requires the partial derivatives to apply the chain rule:

$$D_a \, p = V_a + W_{ab} \, d_a \qquad D_a D_b \, p = W_{ab} \tag{6.17}$$

The partial derivatives can be visualized in a computational graph as shown in Figures 6.6, 6.7, and 6.8. The layers p, $\mathbf{d}$, and $\mathbf{g}$ contain the parameters of the individual functions. Dependencies between parameters are described by edges. The weights correspond to the partial derivatives. Connections between two different layers are weighted with the first-order derivatives. Connections within the same layer have the second-order derivative as weight. In all cases, the connections on the inner layers (red) have the same weights which correspond to (6.17). The weights on the outer layers depend on the respective element type. For the implementation of the core elements, $\mathbf{V}$ and $\mathbf{W}$ are given by:

$$V_a = T_{ab} \, \bar{S}_b \, \Omega \tag{6.18}$$

$$W_{ab} = T_{ca} \, \mathbb{C}_{cd} \, T_{db} \, \Omega \tag{6.19}$$

where $\mathbf{T}$ represents a transformation matrix between the curvilinear system of $\mathbf{d}$ and the local cartesian system of $\bar{\mathbf{S}}$ and $\mathbb{C}$, $\bar{\mathbf{S}}$ is the PK2 prestress, and Ω is the domain

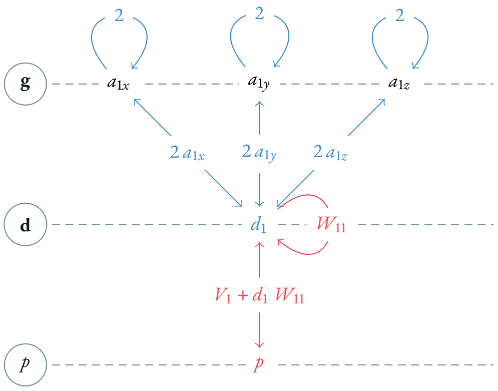


Figure 6.6: Computational graph of a core element for prestressed cable element.

of the corresponding quadrature point. It is sufficient to specify the formulas for $\mathbf{d}$, $\mathbb{C}$, and $\mathbf{T}$ for each element type. The computational steps for cables or trusses, membranes, and couplings are provided in the following sections. For a classical structural analysis, $\mathbf{V}$ and $\mathbf{W}$ remain constant while $\mathbf{d}$ is recalculated in each iteration. For URS $\mathbb{C} = \mathbf{0}$, which implies $\mathbf{W} = \mathbf{0}$ and neglects the elastic terms while $\mathbf{V}$ and $\mathbf{d}$ are updated after each iteration.

Cable

For a simple cable element, the base vector $\mathbf{a}_1$ is required. It is defined by the derivative of the curve w.r.t. the curve parameter. The corresponding base vector in the reference configuration is denoted by $\mathbf{A}_1$. The material stiffness is given by Young's modulus E. The tensors $\mathbf{d}$, $\mathbb{C}$, and $\mathbf{T}$ consist of single entries:

$$\mathbf{d} = \left(\mathbf{a}_1 \cdot \mathbf{a}_1 - \mathbf{A}_1 \cdot \mathbf{A}_1\right) \tag{6.20}$$

$$\mathbf{T} = \frac{1}{2}\left(\frac{1}{\mathbf{A}_1 \cdot \mathbf{A}_1}\right) \tag{6.21}$$

$$\mathbb{C} = \left(E\right) \tag{6.22}$$

$$\Omega = A\,\mathrm{d}L \tag{6.23}$$

where dL is obtained from the integration weight. The basis vectors are calculated according to the geometric discretization. The resulting geometric transformations are used to transform the residual force and stiffness of the core elements accordingly. Choosing $\mathbf{a}_1 = \mathbf{x}_0 - \mathbf{x}_i$ results in $\mathbf{a}_1 \cdot \mathbf{a}_1 = {l_i}^2$, $\mathbf{A}_1 \cdot \mathbf{A}_1 = {L_i}^2$ and $\mathrm{d}L = L$. This provides the energy functional of the linear cable or truss element given by (6.7). The relation to FDM is given by $\mathbf{V} = (q/2)$. Computing $\mathbf{a}_1$ based on NURBS results in the element formulation of the cable provided by [94]. In combination with the transformation for SubDs presented in Section 6.4, the core element can be transformed to represent the edges of a SubD.

Membrane

The geometry of the membrane element is described by a surface. This provides the two base vectors $\mathbf{a}_1$ and $\mathbf{a}_2$ in the deformed configuration, respectively $\mathbf{A}_1$ and $\mathbf{A}_2$ in the reference configuration. The vector $\mathbf{d}$ consists of three entries that measure the longitudinal deformations in the parameter directions and the shear deformation. Consequently, $\mathbf{T}$ and $\mathbb{C}$ are 3×3 matrices:

$$\mathbf{d} = \begin{pmatrix} \mathbf{a}_1 \cdot \mathbf{a}_1 - \mathbf{A}_1 \cdot \mathbf{A}_1 & \mathbf{a}_2 \cdot \mathbf{a}_2 - \mathbf{A}_2 \cdot \mathbf{A}_2 & \mathbf{a}_1 \cdot \mathbf{a}_2 - \mathbf{A}_1 \cdot \mathbf{A}_2 \end{pmatrix} \tag{6.24}$$

$$\mathbf{T} = \frac{1}{2} \begin{pmatrix} {T_{11}}^{11} & {T_{11}}^{22} & {T_{11}}^{12} + {T_{11}}^{21} \\ {T_{22}}^{11} & {T_{22}}^{22} & {T_{22}}^{12} + {T_{22}}^{21} \\ 2\,{T_{12}}^{11} & 2\,{T_{12}}^{22} & 2\,({T_{12}}^{12} + {T_{12}}^{21}) \end{pmatrix}$$

$$\text{with} \quad {T_{\gamma\delta}}^{\alpha\beta} = (\mathbf{e}_\gamma \cdot \mathbf{A}^\alpha)(\mathbf{A}^\beta \cdot \mathbf{e}_\delta) \tag{6.25}$$

$$\mathbb{C} = \frac{E}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix} \tag{6.26}$$

$$\Omega = t\,\mathrm{d}A \tag{6.27}$$

where ν is *Poisson's ratio*, t is the thickness of the membrane, and dA is obtained from the integration weight. The matrix $\mathbf{T}$ transforms between the curvilinear coordinate system of the surface ($\mathbf{A}_1$, $\mathbf{A}_2$) and the local coordinate system ($\mathbf{e}_1$, $\mathbf{e}_2$) where material properties and prestresses are defined. To adjust the orientation of $\mathbf{e}$, a projection strategy can be used as shown in [96]. This is important when dealing with anisotropic materials and prestresses which is not required in the context of

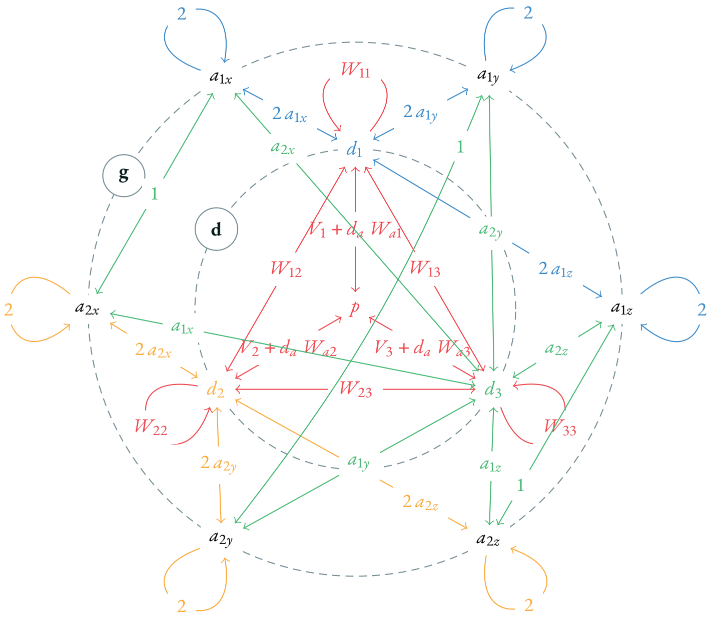


Figure 6.7: Computational graph of a prestressed membrane element.

this work. The local system is therefore chosen as a cartesian system where $\mathbf{e}_1$ is parallel to $\mathbf{A}_1$ and $\mathbf{e}_2$ lies in the tangential plane:

$$\mathbf{e}_1 = \frac{\mathbf{A}_1}{|\mathbf{A}_1|} \qquad \mathbf{e}_2 = \frac{\mathbf{A}_2 - (\mathbf{A}_2 \cdot \mathbf{e}_1)\,\mathbf{e}_1}{|\mathbf{A}_2 - (\mathbf{A}_2 \cdot \mathbf{e}_1)\,\mathbf{e}_1|} \tag{6.28}$$

Defining $\mathbf{a}_1$ and $\mathbf{a}_2$ by the edges of a triangle results in the classic mesh based membrane element. Computing the base vectors based on NURBS results in the membrane element provided by [94].

Weak coupling and support

Supports and couplings are implemented in classical FEM by fixing or coupling the corresponding DOFs. In general, this approach cannot be applied to NURBS, because the control mesh does not interpolate the limit surface. For trimmed NURBS surfaces, the boundaries are not defined directly by the control points, but by trimming curves in the parameter space. This requires an implementation using weak conditions (Figure 6.1). Control points of SubDs can be interpolated selectively by exploiting the sharp features. Boundaries are defined by the control mesh. Additional corner vertices can be used specifically for the modeling of supports. In addition, weak conditions are implemented for SubDs because in certain cases they simplify the modeling.

Weak couplings and supports can be interpreted as springs that minimize the distance between two free points (coupling) or between a free point and a fixed reference (support). By choosing a high stiffness, the distance can be minimized to match a certain tolerance. Such conditions can be positioned independently of the control mesh which allows intuitive modeling when dealing with CAD models. To minimize the difference between two spatial points $\mathbf{x}_a = (x_a, y_a, z_a)$ and $\mathbf{x}_b = (x_b, y_b, z_b)$ the following parameters are chosen:

$$\mathbf{d} = \begin{pmatrix} x_a - x_b & y_a - y_b & z_a - z_b \end{pmatrix} \tag{6.29}$$

$$\mathbf{T} = \begin{pmatrix} e_{1x} & e_{1y} & e_{1z} \\ e_{2x} & e_{2y} & e_{2z} \\ e_{3x} & e_{3y} & e_{3z} \end{pmatrix} \tag{6.30}$$

$$\mathbb{C} = \begin{pmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & k_3 \end{pmatrix} \tag{6.31}$$

$$\Omega = 1 \tag{6.32}$$

where the unit vectors $\mathbf{e}_1$, $\mathbf{e}_2$, and $\mathbf{e}_3$ define a local cartesian coordinate system in which the stiffnesses k_1, k_2, and k_3 are measured. For this element type, no prestress is applied as only the elastic stiffness is required. A weak support can be generated by assuming a constant $\mathbf{x}_b$ which represents the target location. Setting the stiffness in one direction to zero allows the implementation of a sliding support. By updating W_{ab} after each iteration, the coupling/spring can be dynamically oriented.

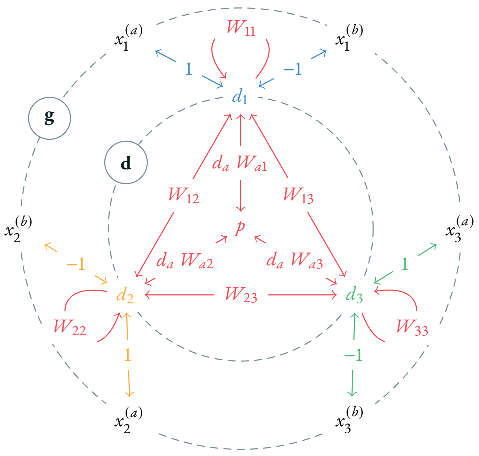


Figure 6.8: Computational graph of a core element for coupling.

6.4 Catmull-Clark subdivision surfaces

Subdivision surfaces are the result of a recursive refinement of an initial control mesh to a smooth limit surface. In this paper, the subdivision scheme of Catmull and Clark [74] is implemented, which is widely used in computational design. The scheme is modified by a subset of the extensions provided in [77] to support sharp creases and corners. The same subdivision rules are also used within the CAD environment Rhino which is used by the authors for the implementation of CAD-integrated analysis.

The Catmull-Clark algorithm allows the subdivision of an arbitrary n-gon mesh into a quadrilateral mesh. Each vertex has a *valence* that corresponds to the number of adjacent edges. The initial control mesh might contain *extraordinary vertices* (EV) not incident on four edges (valence $\neq 4$). Applying the subdivision algorithm to the coarse control mesh results in a finer mesh. This process can be repeated and converges to a smooth limit surface as shown in Figure 6.2. Vertices, edges, and faces of the control mesh have their counterparts on the limit surface. The limit surfaces are C^2-continuous except at points corresponding to an EV [97].

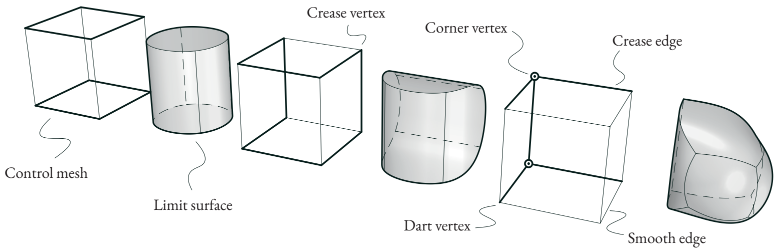


Figure 6.9: Control meshes with different tags and corresponding limit surface

Tagged control mesh

The algorithm refines an arbitrary control mesh. The geometry is defined by the location of the vertices **V** and the topology by the edges **E** and the faces **F**. Each edge connects exactly two vertices while each face connects at least three vertices. To support sharp features, different *tags* are assigned to edges and vertices. An edge might be tagged as a *crease* to model sharp interior and boundary edges. Similarly, a vertex can be tagged as a *crease*, *dart*, or *corner*. An edge or vertex without a tag is called *untagged* or *smooth*.

The influence of different tags on the limit surface is shown in Figure 6.9. Tagged edges are indicated by a thick line. A vertex connected to exactly one crease edge is assumed to be a dart. If the vertex is connected to two edges, it might be a crease or a corner vertex. To distinguish them, the corners are highlighted with the symbol $\odot$. Whenever a vertex is only connected to smooth edges, the vertex is assumed to be smooth.

Extended Catmull-Clark subdivision

This section contains an overview of the subdivision rules provided by [74, 77] which are used in the context of this work. In each subdivision step, a subdivision point is computed for each face $\mathbf{F}_i$, vertex $\mathbf{V}_i$ an edge $\mathbf{E}_i$. To determine the subdivision point $\mathbf{f}_i$ of the face $\mathbf{F}_i$, the average of all n adjacent vertices $\mathbf{V}_j$ is taken.

$$\mathbf{f}_i = \frac{1}{n}\sum_{j=1}^{n}\mathbf{V}_j \tag{6.33}$$

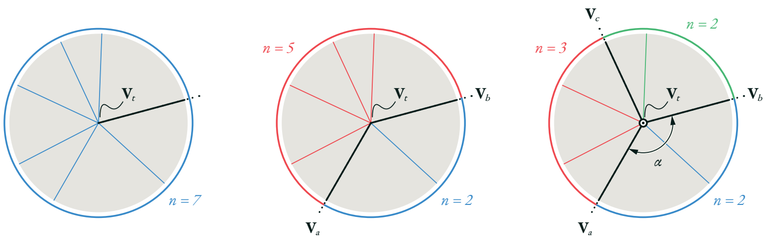


Figure 6.10: Crease edges spanning sectors around a tagged vertex $\mathbf{V}_t$. At a dart vertex (left) there is only one sector (red). A crease vertex (middle) results in two sectors (blue and red). A corner might be adjacent to an arbitrary number of sectors (e.g. blue, red, green).

Depending on the tag of $\mathbf{V}_i$, three scenarios need to be distinguished to compute the subdivision point $\mathbf{v}_i$ of the vertex $\mathbf{V}_i$. If $\mathbf{V}_i$ is a crease, there are exactly two crease edges connecting $\mathbf{V}_i$ with $\mathbf{V}_a$ and $\mathbf{V}_b$. For smooth and dart vertices, the valence of $\mathbf{V}_i$ is denoted with n. In this case, the valence is equal to the number of adjacent edges, faces, or vertices.

$$\mathbf{v}_i = \begin{cases} \mathbf{V}_i & \text{if } \mathbf{V}_i \text{ is a corner} \\ \frac{3}{4}\mathbf{V}_i + \frac{1}{8}\mathbf{V}_a + \frac{1}{8}\mathbf{V}_b & \text{if } \mathbf{V}_i \text{ is a crease} \\ \frac{n-2}{n}\mathbf{V}_i + \frac{1}{n^2}\sum_{j=1}^{n}\mathbf{V}_j + \frac{1}{n^2}\sum_{j=1}^{n}\mathbf{f}_j & \text{if } \mathbf{V}_i \text{ is smooth} \end{cases} \tag{6.34}$$

Next, an angle θ_i is computed for each smooth edge $\mathbf{E}_i$ that is connected to exactly one tagged vertex $\mathbf{V}_t$. If $\mathbf{V}_t$ is a dart, n corresponds to the number of faces connected to the vertex, otherwise, $\mathbf{E}_i$ is surrounded by two crease edges connecting $\mathbf{V}_t$ with $\mathbf{V}_a$ and $\mathbf{V}_b$. These two edges span a *sector* as shown in Figure 6.10. The number of faces within this sector is indicated by n while the angle α corresponds to the smaller angle between the two radii of the sector.

$$\theta_i = \begin{cases} \frac{2\pi}{n} & \text{if } \mathbf{V}_t \text{ is a dart} \\ \frac{\pi}{n} & \text{if } \mathbf{V}_t \text{ is a crease} \\ \frac{\alpha}{n} & \text{if } \mathbf{V}_t \text{ is a corner} \end{cases}$$

$$\text{where} \quad \alpha = \sphericalangle(\mathbf{V}_a, \mathbf{V}_t, \mathbf{V}_b) = \arccos\left(\frac{\mathbf{V}_a - \mathbf{V}_t}{|\mathbf{V}_a - \mathbf{V}_t|} \cdot \frac{\mathbf{V}_b - \mathbf{V}_t}{|\mathbf{V}_b - \mathbf{V}_t|}\right) \tag{6.35}$$

For the computation of the subdivision point $\mathbf{e}_i$ of an edge $\mathbf{E}_i$, three scenarios must be distinguished. Here we need to look at the tag of the edge as well as at the tags of the adjacent vertices. A smooth edge is connected to exactly two faces $\mathbf{f}_a$ and $\mathbf{f}_b$. If only one vertex of the edge is tagged, the tagged vertex is denoted $\mathbf{V}_t$, and the smooth vertex $\mathbf{V}_s$. Otherwise, the order of the vertices is not relevant and they are named $\mathbf{V}_a$ and $\mathbf{V}_b$.

$$\mathbf{e}_i = \begin{cases} \frac{1}{2}\mathbf{V}_a + \frac{1}{2}\mathbf{V}_b & \text{if } \mathbf{E}_i \text{ is a crease} \\ \frac{1}{4}\mathbf{V}_a + \frac{1}{4}\mathbf{V}_b + \frac{1}{4}\mathbf{f}_a + \frac{1}{4}\mathbf{f}_b & \text{if } \mathbf{V}_a \text{ and } \mathbf{V}_b \text{ are tagged} \\ & \text{or both are untagged} \\ \frac{1+\cos\theta}{4}\mathbf{V}_t + \frac{1-\cos\theta}{4}\mathbf{V}_s + \frac{1}{4}\mathbf{f}_a + \frac{1}{4}\mathbf{f}_b & \text{otherwise} \end{cases} \tag{6.36}$$

The subdivision points are the vertices of the refined mesh. From each $\mathbf{f}_i$ a new smooth vertex is obtained which has a valence equal to the number of vertices of $\mathbf{F}_i$. The tags of the new vertices obtained by $\mathbf{v}_i$ and $\mathbf{e}_i$ correspond to the tags of the original $\mathbf{V}_i$ or $\mathbf{E}_i$. For each edge $\mathbf{E}_i = \{\mathbf{V}_a, \mathbf{V}_b\}$ two new edges $\{\mathbf{v}_a, \mathbf{e}_i\}$ and $\{\mathbf{e}_i, \mathbf{v}_b\}$ are obtained which have the same tag and θ as the original $\mathbf{E}_i$. For each face $\mathbf{F}_i$ the subdivision point $\mathbf{f}_i$ is connected with the subdivision points $\mathbf{e}_j$ of the adjacent edges to new smooth edges $\mathbf{f}_i, \mathbf{e}_j$. Based on the new edges, each face is subdivided into n new faces. The same algorithm can be applied to the refined mesh. A vertex $\mathbf{V}_i^{(k)}$ of the control mesh at the refinement level k is associated with a vertex $\mathbf{V}_i^{(k+1)}$ on the refined mesh and converges to a point $\mathbf{V}_i^{(\infty)}$ on the limit surface.

Evaluating the limit surface

A convergence study of the refinement process is required to evaluate a point on the limit surface. This can be performed as an eigenvalue decomposition of the subdivision matrix $\mathbf{A}$. For a crease vertex $\mathbf{V}_i$ which is connected with crease edges $\mathbf{V}_a$ and $\mathbf{V}_b$, the coefficients of the subdivision matrix are provided by (6.34) and (6.36). Recursive refinement results in the k-th power of $\mathbf{A}$.

$$\begin{pmatrix} \mathbf{v}_i^{(k)} \\ \mathbf{e}_a^{(k)} \\ \mathbf{e}_b^{(k)} \end{pmatrix} = \underbrace{\begin{pmatrix} \frac{3}{4} & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \end{pmatrix}}_{\mathbf{A}}{}^{k} \begin{pmatrix} \mathbf{V}_t \\ \mathbf{V}_a \\ \mathbf{V}_b \end{pmatrix} \tag{6.37}$$

The subdivision matrix $\mathbf{A}$ can be decomposed into a linear combination of $\mathbf{\Lambda}$ and $\mathbf{Q}$. $\mathbf{\Lambda}$ is a diagonal matrix containing the eigenvalues of $\mathbf{A}$. The columns of $\mathbf{Q}$ are the corresponding eigenvectors.

$$\mathbf{A}^{(k)} = \mathbf{Q}\,\mathbf{\Lambda}^k\,\mathbf{Q}^{-1} = \begin{pmatrix} 1 & 0 & -\frac{1}{2} \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{4} \end{pmatrix}^{k} \begin{pmatrix} 1 & 0 & -\frac{1}{2} \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{pmatrix}^{-1} \tag{6.38}$$

For the evaluation of the limit state $k \to \infty$, only the diagonal matrix needs to be taken into account. It provides the limit of the matrix power.

$$\mathbf{A}^{(\infty)} = \begin{pmatrix} 1 & 0 & -\frac{1}{2} \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & -\frac{1}{2} \\ 1 & -1 & 1 \\ 1 & 1 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \frac{2}{3} & \frac{1}{6} & \frac{1}{6} \\ \frac{2}{3} & \frac{1}{6} & \frac{1}{6} \\ \frac{2}{3} & \frac{1}{6} & \frac{1}{6} \end{pmatrix} \tag{6.39}$$

With each subdivision step, the edges adjoined to the vertex become smaller. Consequently, the subdivision points of the vertex and the adjoined edges converge to a single point. Therefore, the rows of $\mathbf{A}^{(\infty)}$ are identical. The limit point of the vertex can be written as a linear combination of the control points.

$$\mathbf{v}_i^{(\infty)} = \frac{2}{3}\,\mathbf{V}_i + \frac{1}{6}\,\mathbf{V}_a + \frac{1}{6}\,\mathbf{V}_b \tag{6.40}$$

To evaluate a point on the limit curve of an edge for the parameter η, a finite number of subdivision steps needs to be performed. The subdivision algorithm must be applied k times until η lies within the span of a cubic B-spline. The spline describes the point as a linear combination of the basis functions B and 4 vertices $\mathbf{V}_i^{(k)}$ of the refined edge which defines the control polygon of the spline.

$$\mathbf{x}(\eta) = \sum_i^4 B_i(\eta)\,\mathbf{V}_i^{(k)} \tag{6.41}$$

Points within faces can be evaluated in a similar way. To define the location of the point, a local parametrization is required. After the first refinement step, the control mesh consists of quads (Figure 6.11). Each quad can be parameterized with the local coordinates η and ζ. Again, the mesh has to be refined k times until the point can be described with the help of a bicubic B-spline. The point is then defined by a linear combination of $4 \times 4 = 16$ vertices $\mathbf{V}_{ij}^{(k)}$ of the refined mesh.

$$\mathbf{x}(\eta, \xi) = \sum_i^4 \sum_j^4 B_i(\eta)\, B_j(\xi)\, \mathbf{x}_{ij} \tag{6.42}$$

The base vectors can be computed in a similar way using the derivatives of the B-spline basis functions with respect to the local parameters. Applying the subdivision algorithm k times can be simplified by an eigenvalue analysis as explained in [76].

Quadrature

To integrate the energy functional over the structure numerically, the functional needs to be evaluated at certain quadrature points. Catmull-Clark surfaces are

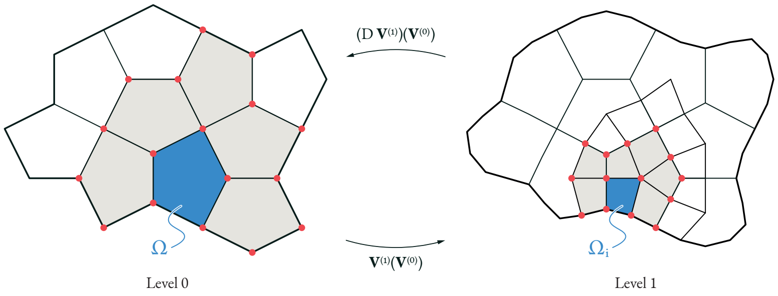


Figure 6.11: Catmull-Clark subdivision. Left: the limit surface of a face (blue) is influenced by a subset of the vertices (red). Right: after applying one step of the subdivision algorithm, an n-gon is subdivided into n quadrilateral faces. The limit surface of each face is influenced by a subset of the refined control mesh. The influence of the control vertices on the refined mesh is described by the partial derivatives.

infinite piecewise polynomials, which makes numerical integration difficult. Within this work, a 4-point *Gauss-Legendre* quadrature rule is used. This corresponds to the procedure for cubic splines in IGA. Each n-gon is subdivided into n quads. The $4 \times 4 = 16$ quadrature points are placed in the parameter space of each quad. This way $16n$ integration points per face are obtained.

The applied quadrature rule cannot perfectly integrate over SubDs. According to the examples in Section 6.6, the accuracy is sufficient in the context of the application. The technique has also already been used successfully in [98, 99]. Other integration strategies (e.g., [86]) were not investigated.

To evaluate the base vectors at the location of the quadrature points, the quad needs to be subdivided up to 4 further times as shown in Figure 6.12. After level 3, it is possible to compute 8 quadrature points without further assumptions. At level 4, the outermost quadrature points are evaluated which are at a smooth border edge. Finally, the remaining quadrature points at level 5 are evaluated. The process is optimized by subdividing only the required parts of the control mesh up to the finest level.

The transformation of the core elements from Section 6.3 according to the geometric description requires the partial derivatives of the base vectors at the quadrature points with respect to the initial control mesh. This includes the subdivision process of the tagged mesh as well as the evaluation of the B-splines. For this purpose, the

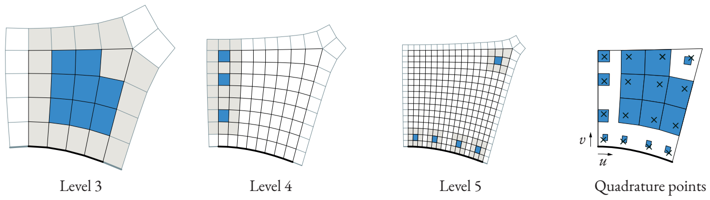


Figure 6.12: Position of the quadrature points. Different refinement levels are necessary for the calculation of the individual points.

partial derivatives of the individual steps must be combined using the chain rule. This concatenation is performed with the help of algorithmic differentiation.

Shape functions with algorithmic differentiation

To apply mechanical properties to SubDs we need to evaluate shape functions and their derivatives at certain points. They are defined by the derivatives of the base vectors w.r.t. the location of the control points. Since each refinement step depends on the previous one, the chain rule must be applied consistently. The procedure for the extended Loop subdivision scheme is described in [81].

In this work, algorithmic differentiation is used to compute the shape functions. This has the advantage that only the algorithm for the subdivision process has to be implemented and this simplifies the handling of different tags in the control mesh. By evaluating the algorithm with hyper-dual numbers, the required derivations are obtained automatically. The resulting Jacobian is equivalent to the evaluated shape functions denoted with $\mathbf{N}$. It accumulates the Jacobians of each refinement step and the B-spline shape functions from (6.41) and (6.42). Since only the first derivative is required, multivariate dual numbers instead of hyper-dual numbers can be used. They only store the 1st-order Taylor polynomial.

Based on the angle θ, the shape functions depend on the current geometry at smooth edges connected to corners. This results in nonlinear shape functions. As a simplification, θ is treated as a constant and updated after each iteration. Using this procedure, the first derivatives and therefore the equilibrium conditions are still valid. The influence of the simplified stiffness on the convergence behavior was not investigated. The geometry can be evaluated as a linear combination of the initial control points:

$$\mathbf{x} = \sum_{i=1}^{n} N_i(\eta, \zeta)\, \mathbf{V}_i \tag{6.43}$$

As shown in Figure 6.11, only a subset of the control vertices affect the geometry within a face. For the other vertices, the shape functions are zero. The definition of the finite element takes advantage of the sparse structure of **N**.

Finite elements

The subdivision surface is divided into finite elements dependent on the faces and edges of the control mesh. Within the same element, the points on the limit surface are influenced by the same control vertices and thus by the same DOFs. The number of DOFs might vary for each element depending on the topology and the tags of the control mesh. For the same reason, the evaluation process of the shape functions might be different for each element.

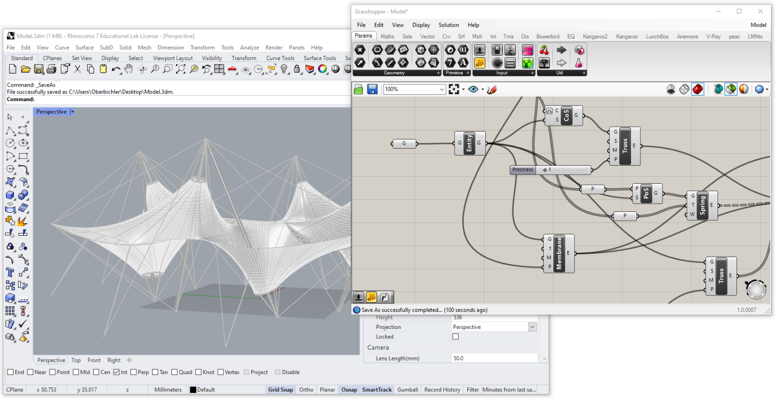
Therefore, the evaluated basis functions cannot be hard-coded as for classical finite elements. Instead, a similar approach as for trimmed NURBS elements is used. Each finite element stores a dynamic list that contains the required data to evaluate the core element and the geometric transformation at each quadrature point. In our scenario, this includes the weights **V**, **W**, and the shape function **N** for each quadrature point. With this information, the local residual force and stiffness can be computed for the element. The total energy, residual force, and stiffness for the whole system result from the sum of all elements.

6.5 CAD-integration

We outline the workflow of the fully CAD-integrated analysis in the research tool *EQlib*[6]. For the implementation, we extend the VPL Grasshopper[7] using custom components written in the programming language C# (Figure 6.13). The tool can be used to define and solve form-finding problems as well as any other types of numerical analysis and optimization problems. We start with a pure geometric model defined in the CAD system. Based on IGA, we also use the same geometric

[6]Access on request.

[7]www.grasshopper3d.com (access: 3.5.2022)


Model.3dm (1 MB) - Rhinoceros 7 Educational Lab License - [Perspective]
File Edit View Curve Surface SubD Solid Mesh Dimension Transform Tools Analyze Render Panels Help
Standard CPlanes Set View Display Select Viewport Layout Visibility Transform Curve Tools Surface Tools
Perspective
Perspective Top Front Right
Grasshopper - Model*
File Edit View Display Solution Help
Params Maths Sets Vector Crv Srf Msh Int Trns Dis Bowerbird EQ Kangaroo2 Kangaroo LunchBox Anemone V-Ray peac LMNts
Geometry Primitive Input Util
Entity
CoS
Truss
Prestress
PoS
Spring
Membrane
Truss
Projection Perspective
Locked
Camera
Lens Length(mm) 50.0

Command: _SaveAs
File successfully saved as C:\Users\Oberbichler\Desktop\Model.3dm.
Command:

Save As successfully completed... (100 seconds ago)

Figure 6.13: CAD-integrated analysis. Direct interaction with the analysis process simplifies the design process.

description for the analysis. In the first step, we apply additional properties to the geometric entities that are required for the specific analysis. Therefore, we provide a set of *design elements* for different tasks such as the definition of a membrane or a cable. The design element subdivides the problem into entities suitable for the design phase. By contrast, the finite elements subdivide the problem into smaller entities that are optimal for the solution process. The finite elements are created automatically in the background. Each design element is able to create a suitable control mesh and a bunch of finite elements for the analysis stage dependent on the definitions made by the designer. The concept is illustrated in Figure 6.14. Each finite element integrates a functional defined by the element formulation over a domain that is influenced by a subset of the control mesh. The integration domain is specified as a dynamic list. Each entry contains the required information to evaluate the core element at a specific location and to transform the result into the geometry. The required information might be stored either as a precomputed value (less computation) or as a function that is evaluated in each iteration (less memory). The same finite element type can be used for classic meshes, SubDs, and trimmed NURBS by simply changing the geometric transformation.

The VPL provides a flexible and extendable user interface for defining arbitrary com-

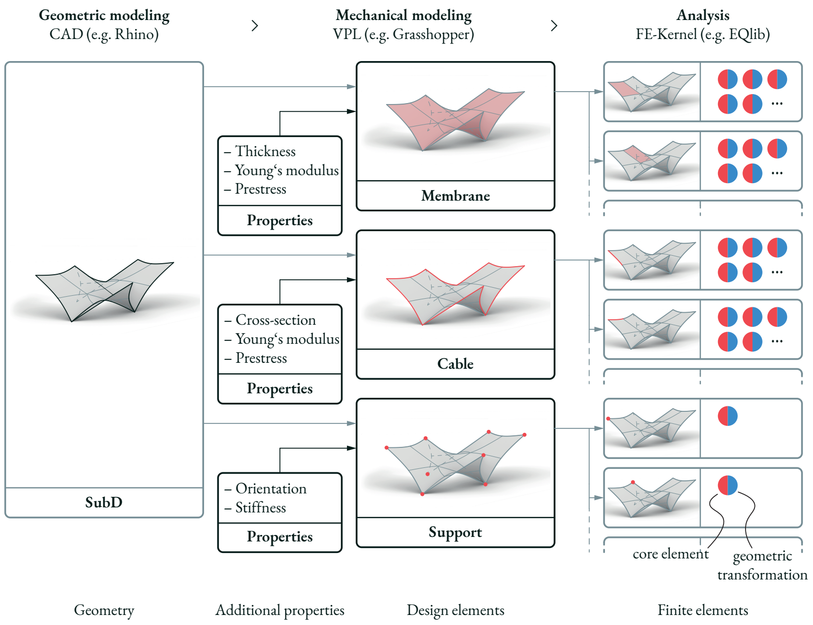


Figure 6.14: Diagram of the CAD-integrated workflow. Modeling of the geometry and application of the mechanical properties takes place at different levels of abstraction. The designer works with familiar entities such as faces and edges. The decomposition into finite elements for the analysis is done in the background.

putational models. Using a core-congruential formulation simplifies the memory management and allows an efficient implementation as a plugin that is integrated into the CAD environment. The direct integration allows access to the advanced geometry kernel of the CAD. For the solution process, we take advantage of the sparse linear algebra functionalities of the *Intel Math Kernel Library* (MKL). This allows even more complex examples to be solved in a CAD-integrated environment.

6.6 Examples

This section provides several numerical examples to verify the proposed method.

Modeling aspects

Figure 6.15 compares the required modeling techniques for a simple membrane structure when using NURBS and SubDs. NURBS surfaces are defined by a regular control mesh. For the design of more complex surface models, it is usually necessary to trim NURBS in the parameter space and to couple several trimmed NURBS to a multi-patch. The modeling of a pentagonal shape requires trimming edges that define the boundaries in the parameter space of the surfaces. The trimmed domain is used for the definition of a prestressed membrane. The trimming edges define prestressed edge cables. Two NURBS surfaces are coupled by a coupling edge which defines the domain for the coupling. The coupling acts as a spring which results in a small gap between the patches. Assigning a high stiffness to the coupling reduces the gap below the model tolerance. In addition, two coupling points at the end of the coupling edge are defined to avoid gaps due to the higher prestress of the edge cables. The corners of the trimmed domain do not correspond to the control points of the NURBS. Embedded points must be used to define the supports since the vertices of the trimmed surfaces do not coincide with the control mesh. To create C1 continuity along an edge, additional coupling conditions are needed to couple the surface normals of both sides [32]. Again the penalty factor determines how well continuity would be maintained.

Modeling the same structure with SubDs does not require trimming, embedding, or coupling. The control mesh can be adapted to the shape of the structure by combining triangles, quads, and pentagons. Corner vertices, boundaries, and inner edges of the limit surface have direct counterparts on the control mesh. Properties for membranes, cables, and supports are assigned directly to the entities of the mesh. This can significantly simplify the modeling of such structures. The continuity at the edges is inherently given by the definition of the SubDs and does not have to be minimized by a penalty factor. Support and coupling constraints can also increase the modeling capabilities of SubDs. This could be used, e.g., to connect SubDs with different refinement levels.

Form-finding of a catenoid

This example is based on [27] and compares an analytic solution of a minimal surface with the numerical results. The objective is to find a minimal surface that connects two circles of radius R and a vertical distance of H as shown in Figure 6.16. The most important points are repeated here. For a more detailed explanation, the

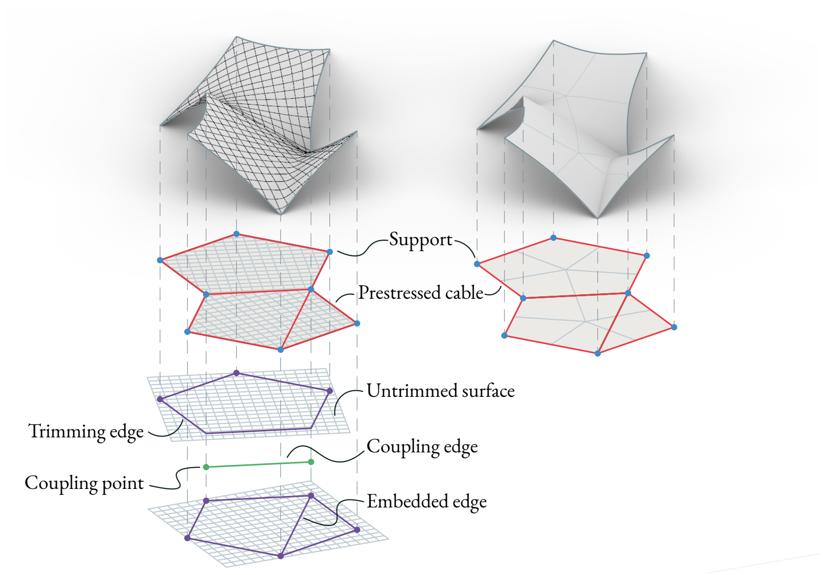


Figure 6.15: Modeling features required for NURBS (left) and SubDs (right).

interested reader should refer to the mentioned publication.

Analytic solution

According to the *Goldschmidt solution* [100], the analytic expression for the area of the minimal surface depends on the ratio between H and R:

$$A_{\text{analytic}} = \begin{cases} (r_0)^2\,\pi\,\sinh\left(\dfrac{H}{r_0}\right) + r_0\pi H & H \leq H_{\text{crit}} \approx 1.32548684\,R \\ 2R^2\pi & H > H_{\text{crit}} \end{cases} \tag{6.44}$$

For $H > H_{\text{crit}}$, the solution of the form-finding collapses into two circles connected by a straight line. Otherwise, the solution corresponds to a catenoid which is described in cylindrical coordinates (r, θ, z) by:

$$r(z) = r_0 \cosh\left(\frac{z}{r_0}\right) \qquad \theta \in [-\pi, \pi] \qquad z \in [-H/2, H/2] \tag{6.45}$$

The unknown constant r_0 is determined by the given radius at both edges:

$$R = r(z = \pm H/2) = r_0 \cosh\left(\frac{H}{2r_0}\right) \tag{6.46}$$

where r_0 denotes the minimal radius r_0 at $z = 0$. The expression cannot be resolved analytically for r_0. Instead, we solve it numerically for a given R and H with $r_0^{(0)} = R$ as an initial guess. With r_0 and H we are then able to compute the area of the catenoid in (6.44).

Numerical solution

We use a cylinder as the initial geometry for the form-finding. The cylinder has a radius R and a height H as shown in Figure 6.16. To test the functionality for arbitrary topologies, we use a control mesh that consists of triangles, quads, and hexagons. Using the same pattern, three control networks with different refinement levels were generated. Rational subdivision surfaces are required to represent conic sections exactly. These are not yet supported by the CAD program. Therefore the cylinder is approximated by non-rational SubDs. Depending on the resolution of the control mesh, there is already a deviation from the cylinder. To apply the boundary conditions we fix the displacement of the control points for the crease edges. We do this by fixing the DOFs of the corresponding FE nodes.

Figure 6.16 shows the convergence of the form-finding for different refinement levels. The numerical solution converges to the analytical reference. For $H > H_{\text{crit}}$, we also obtain the degenerate solution which consists of two discs.

Form-finding of a Schwarz-P minimal surface

The shape of a *Schwarz-P surface* is computed numerically via form-finding. The SubD shown in Figure 6.17a is used as the initial geometry. The geometry is defined by a single quadrilateral control mesh with EVs. At first, only a membrane prestress

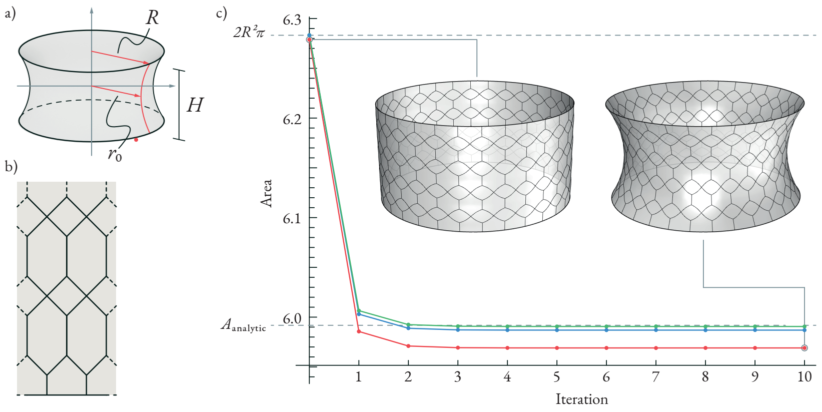


Figure 6.16: Form-finding of a catenoid. (a) Terminology for the catenoid (b) Pattern of the control mesh. (c) Convergence of the form-finding for different control meshes (red = 420 vertices/270 faces, blue = 1560 vertices/1020 faces, green = 6000 vertices/3960 faces).

of 1 is applied to the faces. The control points of the crease boundary edges are fixed in-plane by adding supports to the corresponding FE nodes. Figure 6.17 shows the convergence of the iterative form-finding process in red. After the first iteration, the area is reduced to 82% compared to the initial geometry (b). During the subsequent iterations, the geometry changes only slightly and straightaway matches the shape of the Schwarz-P surface. However, it becomes apparent that this solution is only an unstable local minimum. After 10 iterations, the geometry starts to collapse. The openings increase with each iteration (c). The geometry of the faces degenerates to lines while the area converges to zero (d and e). It is therefore a valid solution in the sense of a minimal surface. The instability of a problem is caused by minimal imperfections of the geometry that cannot be avoided in a numerical computation. The same behavior was explained in [93] for a *Costa surface*.

By applying prestressed cables to the crease edges, the solution process is directed in a particular direction. We choose a prestress of 1. The convergence is shown in blue. Due to the prestress in the cables, the length of the edges is reduced in each iteration (f). Again, the form-finding converges to a zero-area solution (g).

To stabilize the problem, the first 10 iterations are computed with membrane prestress only. Then an elastic stiffness is applied to the edges ($EA = 1$). This restricts the stretching and compression of the edges. A collapse of the surface is no longer possible (h).

Next, the boundary conditions are remodeled by using weak supports at the naked edges. The edges are fixed in the direction of the global coordinate system. Therefore, the transformation matrix can be ignored and the stiffness of the weak supports is defined in global directions. By increasing the penalty factor, the shape converges towards the previous solution where the FE nodes were fixed. Figure 6.18a compares the area for different penalty factors after 5 iterations.

Finally, the initial geometry is rotated in space through 30° around the global X- and Y-axis. Since the orientation of the edge planes is no longer along global directions, the transformation matrix is required. The form-finding is repeated for different penalty factors as shown in Figure 6.18b. By increasing the penalty factor, the form-finding converges towards the reference solution.

Form-finding of SkySong

Finally, we demonstrate the application of the method for a more complex example inspired by the *SkySong* multi-use campus in *Scottsdale, USA*. In the center of the campus, there is an impressive tensile structure that was designed by *FTL Design Engineering Studio* in 2009. The structure was remodeled by form-found SubDs in combination with trusses and cables. The dimensions were estimated on the basis of openly accessible image material. The ratio of the prestress of the membrane and edge cables was adjusted so that the appearance complies with the real building. The result is shown in Figure 6.19.

6.7 Conclusion

It was shown how Catmull-Clark subdivision surfaces can be used for the form-finding of lightweight structures in a CAD-integrated IGA environment by using URS. The approach is based on the structure of a classical FE program and allows combination with other elements and analysis methods. The modular design of CCF allows a mechanical core element to be combined with different geometric discretizations. It reduces the implementation effort and increases the computing speed at the same time. By supporting extended SubDs in addition to classical meshes and NURBS, all types of free-form geometries currently available in the state-of-the-art CAD environment Rhino can be covered. The seamless integration into

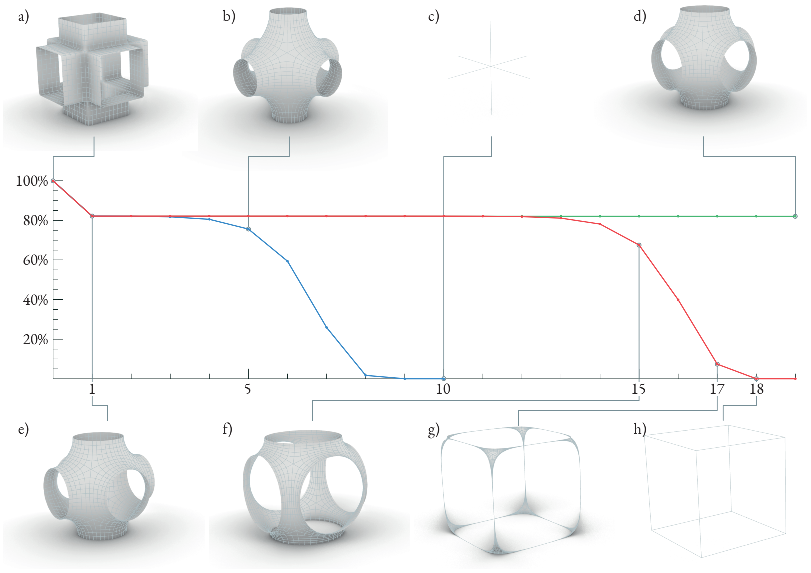


Figure 6.17: Form-finding for a Schwarz-P surface with different configurations: prestress only in the membrane (red); prestress in the membrane and the edge cables (blue); elastic stiffness in the edge cables after iteration 10 for stabilization (green).

a CAD system in combination with VPLs offers significant advantages, especially at the beginning of the design phase. Particularly in form-finding, several design iterations are necessary, which are accelerated by the direct integration.

Although the focus was on membrane and cable elements the methodology can be applied to other types of structural elements as well. According to CCF, the core elements from[71] can be used and combined with the geometry transformation for SubDs described in this paper. This yields a modular toolbox for numerical analysis.

There is particular potential for improvement in calculating and positioning the integration points. The current approach is very much based on the procedure for NURBS. The use of AD simplifies the computation of shape functions for extended SubDs. The procedure should be better adapted to SubDs to increase

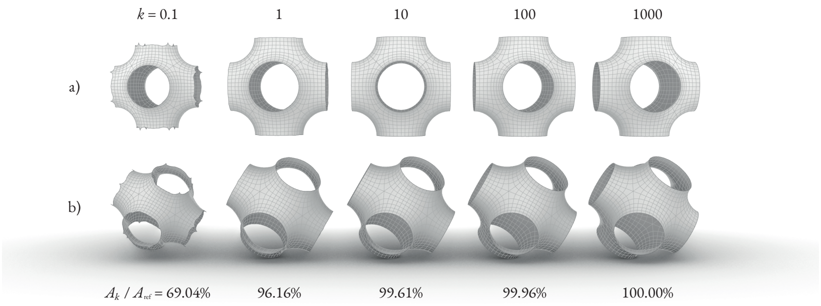


Figure 6.18: Edge support in global and local directions with weak boundary conditions. By increasing the penalty factor k, the area of the solution A_k converges towards the reference solution A_{ref}.

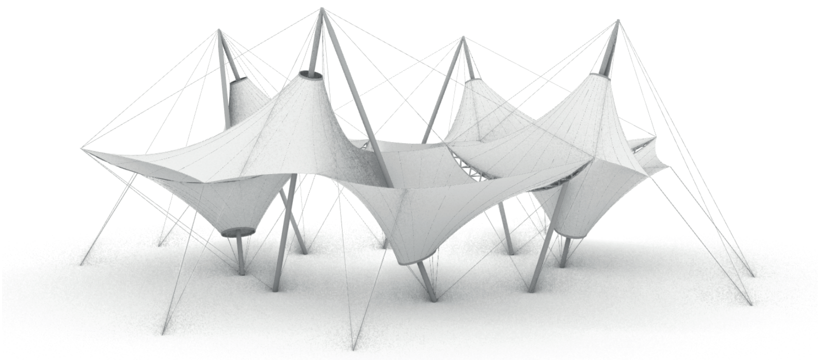


Figure 6.19: Form-finding of SkySong ASU.

performance. This could be done by use of optimized integration rules, or the pre-computation of shape functions for frequently occurring topological situations e.g., regular faces.

Acknowledgment

This work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the project “Integrierte Formfindung, Simulation und Optimierung mit der generalisierten Kraftdichtemethode auf der Grundlage gemischter Funktionale” (434336509). Its support is gratefully acknowledged.

7 Conclusion and outlook

Interactive form-finding and analysis tools integrated into CAD frameworks are improving the design process of lightweight structures significantly. In order to follow the concept of CAD-integrated IGA, a wide variety of geometry descriptions must be supported to let the boundaries between CAD and FEM disappear. In combination with different types of structural analysis and form-finding strategies, this results in a high implementation effort. Due to the complexity of the computation involved, an external finite element kernel is usually required. For the communication between FEA and CAD, additional interfaces must be developed, e.g. an exchange via writing and reading files. The exchange of functions, in particular, is restricted or even impossible. Functionalities from CAD have to be re-programmed in the FEA kernel and vice versa. This contradicts the idea of a 'fully CAD-integrated' design and analysis workflow . The computational speed of this classic approach remains rather poor and precludes an interactive design using smooth geometries. Certainly, the computational performance can be increased by using more powerful and expensive hardware. For the sake of efficiency, however, this work investigated how performance can be increased on the methodical level.

This thesis elaborated a methodology to address these issues and implement finite elements in an efficient and modular way. The mechanical model is interpreted as a parametric model which calculates the energy within the structure as a function of the degrees of freedom. A stable equilibrium will then correspond to a state of minimum energy. The energy functional is parameterized explicitly as a function of the location of discrete nodes. Finding a minimum of this function leads – for non-linear problems – to an inverse problem which is solved in an iterative process using the Newton-Raphson algorithm. This requires the residual force vector and the tangential stiffness matrix, which are obtained as the derivatives of the energy functional. The computation of these derivatives incurs most of the implementation effort. This effort is reduced by using AD. Instead of implementing the derivatives manually, only the energy functional needs to be implemented to compute the required derivatives. Since the energy functional can be reparameterized easily, a wide variety of geometry descriptions can be supported.

A short overview of different freeform parameterizations highlights the algorithmic character of subdivision geometries. The geometric functions for trimmed NURBS models required for IBRA were implemented in the package *ANurbs*[1] for Python and C++. In addition, an interface was created to exchange geometry models with the CAD software package Rhino. Therefore, ANurbs contains all the necessary functionalities to prepare geometries for IBRA. The implementation has also been adopted, for example, in the open-source FE solver *Kratos Multiphysics*[2].

The interpretation of the energy functional as a parametric model similar to a node-based VPL illustrates the decomposition of the functional into smaller components. The knowledge of this composition in the form of a computational graph enables the application of AD to determine the required derivatives automatically. The application of AD using hyperdual numbers was presented, which enables easy integration into existing FE frameworks. Here, the derivatives are calculated using the forward method. The computation of gradients and Hessians using hyperdual numbers was implemented within the package *HyperJet*[3] for the programming languages Python, C++ and C#.

To increase the computational performance, the adjoint method is used to evaluate the required derivatives. The energy functional is a scalar function of the degrees of freedom. For smooth free-form geometries, this results in a high number of input parameters. In this case, the adjoint method leads to a reduction of the computational effort without affecting the quality of the results. It has been shown

[1]https://github.com/oberbichler/ANurbs (access: 31.10.2022)
[2]https://github.com/KratosMultiphysics/Kratos (access: 31.10.2022)
[3]https://github.com/oberbichler/HyperJet (access: 31.10.2022)

that this leads to a reduction in computational operations and constant memory consumption, which is independent of the polynomial degree of the geometry. This was verified on the basis of theoretical considerations and also through practical benchmarks. It is important to take advantage of the well-known structure of an FEM problem in order to increase the performance of AD. The decomposition of the structure into finite elements, for example, can be used to obtain the sparse structure of the Hessian and allows the application of AD on the element level. For each element type, the corresponding energy functional needs to be formulated. In this way, mechanical elements, as well as geometric coupling conditions, can be realized. By implementing additional differentiation rules for mechanical operations, such as the computation of strains or the penalty functional, AD can be optimized for dealing with structural analysis.

In addition to the increased computational performance, the adjoint approach results in a modular structure of the element formulation. It allows a clean separation into mechanical and geometric components according to a core-congruential element formulation. The mechanical component is referred to as the "core element" and is independent of the chosen geometric description. Core elements for cables, membranes, beams and shells are formulated. The core elements include elastic deformation and prestress. This enables classic structural analysis as well as form-finding. In addition, hybrid structures can be modelled in a very elegant way, combining form-finding elements and classic elastic elements in one simulation. Various geometric descriptions can be used by combining the core element with different geometric transformations. The approach was presented using discrete meshes, NURBS and subdivision surfaces. These blocks permit numerous combinations , resulting in a versatile IGA toolbox. When implementing core element and geometry transformation, hyperdual numbers can be used.

The increase in efficiency allows direct integration of the analysis tools in CAD and simplifies the exchange between CAD and FEM without additional interfaces and code duplication. Functionalities and data structures can be shared . For example, geometric operations from CAD can be used by the FEM kernel to compute trimmed geometries. Moreover, it enables direct interaction with the analysis process. For smaller models, real-time calculations based on high-order geometries are possible. At the same time, the dimensions of the models that can be calculated in CAD-integrated analysis increase. The resulting methodology was implemented in the plugin *EQlib* for Rhino and Grasshopper. As a result, a modular and automatable framework was obtained, which also enables efficient implementation of IGA within interactive CAD environments.

This modular system can subsequently be extended to other applications. The modelling of design surfaces with the help of an implicit description of certain requirements, e.g. C2-continuous junctions of NURBS patches, was tested and seems to be a promising application. The solution of such optimization problems requires skilful adjustment of parameters to meet industry standards. Here, an efficient CAD-integrated implementation can help in choosing the appropriate settings.

By using advanced meta-programming capabilities, the integration of AD into CAD tools can be further simplified. The user could then formulate their own elements or goals on-the-fly and adapt them perfectly to the problem. The algorithm, generated in the background, for calculating the derivatives can make use of high-end techniques such as parallelization and GPU acceleration without the designer having to deal with them. Initial experiments with SIMD instructions have already significantly accelerated the calculation.

Since VPLs already present a computational graph, AD could be used to integrate gradient-based optimization into such frameworks.

In the long term, this work should contribute to simplifying the implementation of IGA and making it more accessible. The engineer and designer can focus on the abstract formulation of the problem, while the machine can optimally perform the systematic but time-consuming implementation of the derivatives. This allows collaboration between man and machine, where the strengths of both sides are combined in a productive way.

List of Figures

Bibliography

[1] I. E. Sutherland, Sketchpad a Man-Machine Graphical Communication System, SIMULATION 2 (5) (1964) R–3–R–20. doi:10.1177/003754976400200514.

[2] A. Key, Doing with Images Makes Symbols: Communicating with Computers (1987).

[3] F. Dieringer, B. Philipp, R. Wüchner, K.-U. Bletzinger, Numerical Methods for the Design and Analysis of Hybrid Structures, International Journal of Space Structures 28 (3-4) (2013) 149–160. doi:10.1260/0266-3511.28.3-4.149.

[4] L. Fuhrimann, V. Moosavi, P. O. Ohlbrock, P. Dacunto, Data-Driven Design: Exploring new Structural Forms using Machine Learning and Graphic Statics, in: Proceedings of the IASS Annual Symposium 2018, 2018, pp. 1–8.

[5] T. Oberbichler, A. Bauer, A.-K. Goldbach, R. Wüchner, K.-U. Bletzinger, CAD integrated analysis for the design process, Bautechnik 96 (5) (2019). doi:10.1002/bate.201800105.

[6] P. O. Ohlbrock, Combinatorial Equilibrium Modelling, Ph.D. thesis, ETH Zürich (4 2021). doi:10.3929/ETHZ-B-000478732.

[7] J. Nocedal, S. Wright, Numerical Optimization, Springer Series in Operations Research and Financial Engineering, Springer New York, 2006. doi:10.1007/978-0-387-40065-5.

[8] M. J. Kochenderfer, T. A. Wheeler, Algorithms for Optimization, The MIT Press, 2019.

[9] R. Norton, Design of Machinery, 6th Edition, McGraw-Hill Education, New York, 2020.

[10] P. O. Ohlbrock, P. D'Acunto, A Computer-Aided Approach to Equilibrium Design Based on Graphic Statics and Combinatorial Variations, Computer-Aided Design 121 (2020) 102802. doi:10.1016/j.cad.2019.102802.

[11] R. Pastrana, P. O. Ohlbrock, T. Oberbichler, P. D'Acunto, S. Parascho, Constrained Form-Finding of Tension-Compression Structures using Automatic Differentiation, Computer-Aided Design (2022) 103435doi:10.1016/j.cad.2022.103435.

[12] J. McCarthy, An Algebraic Language for the Manipulation of Symbolic Expressions (1958).

[13] F. V. McBride, Computer Aided Manipulation of Symbols, Ph.D. thesis, Queen's University of Belfast, Belfast (1970).

[14] J. A. Fike, J. J. Alonso, The Development of Hyper-Dual Numbers for Exact Second-Derivative Calculations, AIAA paper 2011-886, 49th AIAA Aerospace Sciences Meeting (2011). doi:10.2514/6.2011-886.

[15] F. Biscani, Parallel sparse polynomial multiplication on modern hardware architectures, in: Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation - ISSAC '12, ACM Press, New York, New York, USA, 2012, pp. 83–90. doi:10.1145/2442829.2442845.

[16] T. A. Grandine, The Extensive Use of Splines at Boeing, SIAM News (2005).

[17] H. Pottmann, A. Asperl, M. Hofer, A. Kilian, D. Bentley, Architectural Geometry, Bentley Institute Press, 2007.

[18] G. De Rham, Un peu de mathématiques à propos d'une courbe plane, Elemente der Mathematik 2 (1947) 89–97. doi:10.5169/seals-12825.

[19] G. M. Chaikin, An algorithm for high-speed curve generation, Computer Graphics and Image Processing 3 (4) (1974) 346–349. doi:10.1016/0146-664X(74)90028-8.

[20] J. M. Lane, R. F. Riesenfeld, A theoretical development for the computer generation and display of piecewise polynomial surfaces, IEEE transactions on pattern analysis and machine intelligence 2 (1) (1980) 35–46. doi:10.1109/TPAMI.1980.4766968.

[21] C. de Boor, On calculating with B-splines, Journal of Approximation Theory 6 (1) (1972) 50–62. doi:10.1016/0021-9045(72)90080-9.

[22] L. Piegl, W. Tiller, The NURBS Book, Monographs in Visual Communications, Springer Berlin Heidelberg, Berlin, Heidelberg, 1995. doi:10.1007/978-3-642-97385-7.

[23] D. Rogers, An Introduction to NURBS, 1st Edition, Morgan Kaufmann, 2000. doi:10.1016/B978-1-55860-669-2.X5000-3.

[24] T. Oberbichler, E. Schling, K.-U. Bletzinger, Tracing curvature paths on trimmed multipatch surfaces (under review), Applied Mathematical Modelling (2023).

[25] T. J. Hughes, J. A. Cottrell, Y. Bazilevs, Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement, Computer Methods in Applied Mechanics and Engineering 194 (39-41) (2005) 4135–4195. doi:10.1016/j.cma.2004.10.008.

[26] J. A. Cottrell, T. J. R. Hughes, Y. Bazilevs, Isogeometric Analysis, John Wiley & Sons, Ltd, Chichester, UK, 2009. doi:10.1002/9780470749081.

[27] B. Philipp, M. Breitenberger, I. D'Auria, R. Wüchner, K. U. Bletzinger, Integrated design and analysis of structural membranes using the Isogeometric B-Rep Analysis, Computer Methods in Applied Mechanics and Engineering 303 (2016) 312–340. doi:10.1016/j.cma.2016.02.003.

[28] A. M. Bauer, M. Breitenberger, B. Philipp, R. Wüchner, K. U. Bletzinger, Nonlinear isogeometric spatial Bernoulli beam, Computer Methods in Applied Mechanics and Engineering 303 (2016) 101–127. doi:10.1016/j.cma.2015.12.027.

[29] J. Kiendl, K. U. Bletzinger, J. Linhard, R. Wüchner, Isogeometric shell analysis with Kirchhoff-Love elements, Computer Methods

in Applied Mechanics and Engineering 198 (49-52) (2009) 3902–3914. doi:10.1016/j.cma.2009.08.013.

[30] B. Oesterle, R. Sachse, E. Ramm, M. Bischoff, Hierarchic isogeometric large rotation shell elements including linearized transverse shear parametrization, Computer Methods in Applied Mechanics and Engineering 321 (2017) 383–405. doi:10.1016/j.cma.2017.03.031.

[31] R. Echter, B. Oesterle, M. Bischoff, A hierarchic family of isogeometric shell finite elements, Computer Methods in Applied Mechanics and Engineering 254 (2013) 170–180. doi:10.1016/j.cma.2012.10.018.

[32] M. Breitenberger, A. Apostolatos, B. Philipp, R. Wüchner, K. U. Bletzinger, Analysis in computer aided design: Nonlinear isogeometric B-Rep analysis of shell structures, Computer Methods in Applied Mechanics and Engineering 284 (2015) 401–457. doi:10.1016/j.cma.2014.09.033.

[33] A. J. Herrema, E. L. Johnson, D. Proserpio, M. C. Wu, J. Kiendl, M. C. Hsu, Penalty coupling of non-matching isogeometric Kirchhoff–Love shell patches with application to composite wind turbine blades, Computer Methods in Applied Mechanics and Engineering 346 (2019) 810–840. doi:10.1016/j.cma.2018.08.038.

[34] A. M. Bauer, R. Wüchner, K. U. Bletzinger, Weak coupling of nonlinear isogeometric spatial Bernoulli beams, Computer Methods in Applied Mechanics and Engineering 361 (4 2020). doi:10.1016/j.cma.2019.112747.

[35] W. Dornisch, G. Vitucci, S. Klinkel, The weak substitution method - an application of the mortar method for patch coupling in NURBS-based isogeometric analysis, International Journal for Numerical Methods in Engineering 103 (3) (2015) 205–234. doi:10.1002/nme.4918.

[36] A. Apostolatos, Isogeometric Analysis of Thin-Walled Structures on Multipatch Surfaces in Fluid-Structure Interaction, Ph.D. thesis, Technische Universität München (2019).

[37] M. Ruess, D. Schillinger, A. I. Özcan, E. Rank, Weak coupling for isogeometric analysis of non-matching and trimmed multi-patch geometries, Computer Methods in Applied Mechanics and Engineering 269 (2014) 46–71. doi:10.1016/j.cma.2013.10.009.

[38] V. P. Nguyen, P. Kerfriden, M. Brino, S. P. Bordas, E. Bonisoli, Nitsche's method for two and three dimensional NURBS patch coupling, Computational Mechanics 53 (6) (2014) 1163–1182. doi:10.1007/s00466-013-0955-3.

[39] X. Du, G. Zhao, W. Wang, Nitsche method for isogeometric analysis of Reissner-Mindlin plate with non-conforming multi-patches, Computer Aided Geometric Design 35-36 (2015) 121–136. doi:10.1016/j.cagd.2015.03.005.

[40] Y. Guo, M. Ruess, Nitsche's method for a coupling of isogeometric thin shells and blended shell structures, Computer Methods in Applied Mechanics and Engineering 284 (2015) 881–905. doi:10.1016/j.cma.2014.11.014.

[41] W. Jiang, C. Annavarapu, J. E. Dolbow, I. Harari, A robust Nitsche's formulation for interface problems with spline-based finite elements, International Journal for Numerical Methods in Engineering 104 (7) (2015) 676–696. doi:10.1002/nme.4766.

[42] Y. Guo, M. Ruess, D. Schillinger, A parameter-free variational coupling approach for trimmed isogeometric thin shells, Computational Mechanics 59 (4) (2017) 693–715. doi:10.1007/s00466-016-1368-x.

[43] Y. Guo, J. Heller, T. J. Hughes, M. Ruess, D. Schillinger, Variationally consistent isogeometric analysis of trimmed thin shells at finite deformations, based on the STEP exchange format, Computer Methods in Applied Mechanics and Engineering 336 (2018) 39–79. doi:10.1016/j.cma.2018.02.027.

[44] E. Brivadis, A. Buffa, B. Wohlmuth, L. Wunderlich, Isogeometric mortar methods, Computer Methods in Applied Mechanics and Engineering 284 (2015) 292–319. doi:10.1016/j.cma.2014.09.012.

[45] S. Schuß, M. Dittmann, B. Wohlmuth, S. Klinkel, C. Hesch, Multi-patch isogeometric analysis for Kirchhoff–Love shell elements, Computer Methods in Applied Mechanics and Engineering 349 (2019) 91–116. doi:10.1016/j.cma.2019.02.015.

[46] A. M. Bauer, M. Breitenberger, B. Philipp, R. Wüchner, K. U. Bletzinger, Embedded structural entities in NURBS-based isogeometric analysis, Computer Methods in Applied Mechanics and Engineering 325 (2017) 198–218. doi:10.1016/j.cma.2017.07.010.

[47] A. M. Bauer, R. Wüchner, K. U. Bletzinger, Innovative CAD-integrated isogeometric simulation of sliding edge cables in lightweight structures, Journal of the International Association for Shell and Spatial Structures 59 (4) (2018) 251–258. doi:10.20898/j.iass.2018.198.039.

[48] E. Haug, K. Choi, V. Komkov, Design Sensitivity Analysis of Structural Systems, Academic Press, 1986.

[49] C. A. Felippa, L. A. Crivelli, B. Haugen, A survey of the core-congruential formulation for geometrically nonlinear TL finite elements, Archives of Computational Methods in Engineering 1 (1994) 1–48. doi:10.1007/BF02736179.

[50] L. A. Crivelli, C. A. Felippa, A three-dimensional non-linear Timoshenko beam based on the core-congruential formulation, International Journal for Numerical Methods in Engineering 36 (1993) 993. doi:10.1002/nme.1620362106.

[51] A. Vigliotti, F. Auricchio, Automatic Differentiation for Solid Mechanics, Archives of Computational Methods in Engineering (2020). doi:10.1007/s11831-019-09396-y.

[52] Technical University of Munich, Carat++ (2020).
URL `www.cee.ed.tum.de/st/software/forschung/carat/`

[53] J. Rombouts, G. Lombaert, L. De Laet, M. Schevenels, On the equivalence of dynamic relaxation and the Newton-Raphson method, International Journal for Numerical Methods in Engineering 113 (9) (2018) 1531–1539. doi:10.1002/nme.5707.

[54] C. Ridders, Accurate computation of $F'(x)$ and $F'(x)\,F''(x)$, Advances in Engineering Software (1978) 4 (2) (1982) 75–76. doi:10.1016/S0141-1195(82)80057-0.

[55] C. Elliott, The simple essence of automatic differentiation, Proceedings of the ACM on Programming Languages 2 (ICFP) (2018) 1–29. doi:10.1145/3236765.

[56] G. Corliss, C. Faure, A. Griewank, L. Hascoet, U. Naumann, Automatic Differentiation of Algorithms, Springer New York, New York, NY, 2002. doi:10.1007/978-1-4613-0075-5.

[57] A. Griewank, A. Walther, Evaluating Derivatives, Society for Industrial and Applied Mathematics, 2008. doi:10.1137/1.9780898717761.

[58] M. Henrard, Algorithmic Differentiation in Finance Explained, Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-53979-9.

[59] C. C. Margossian, A Review of Automatic Differentiation and its Efficient Implementation, WIREs Data Mining and Knowledge Discovery 9 (4) (2019) e1305–undefined. doi:10.1002/widm.1305.

[60] A. H. Gebremedhin, A. Walther, An introduction to algorithmic differentiation, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 10 (1) (1 2020). doi:10.1002/widm.1334.

[61] M. Sagebaum, T. Albring, N. R. Gauger, High-performance derivative computations using CoDiPack, ACM Transactions on Mathematical Software 45 (4) (12 2019). doi:10.1145/3356900.

[62] A. Walther, A. Griewank, Getting Started with ADOL-C, in: Combinatorial Scientific Computing, Chapman-Hall CRC Computational Science, 2012, pp. 181–202. doi:10.1201/b11644-8.

[63] U. Naumann, K. Leppkes, J. Lotz, Derivative Code by Overloading in C++ (dco/c++): Introduction and Summary of Features., Tech. rep., RWTH Aachen University (2016).

[64] J. López, C. Anitescu, T. Rabczuk, Isogeometric structural shape optimization using automatic sensitivity analysis, Applied Mathematical Modelling 89 (2021) 1004–1024. doi:10.1016/j.apm.2020.07.027.

[65] S. Roman, The Formula of Faà di Bruno, The American Mathematical Monthly 87 (10) (1980) 805–809. doi:10.1080/00029890.1980.11995156.

[66] M. Hardy, Combinatorics of Partial Derivatives, The Electronic Journal of Combinatorics 13 (1) (1 2006). doi:10.37236/1027.

[67] M. Spivak, Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus, 5th Edition, Westview Press, 1971. doi:10.1201/9780429501906.

[68] T. Oberbichler, HyperJet, http://github.com/oberbichler/HyperJet (2019). doi:10.5281/zenodo.4599178.
URL `https://github.com/oberbichler/HyperJet`

[69] K. Linkwitz, H. J. Schek, L. Gründig, Die Gleichgewichtsberechnung von Seilnetzen unter Zusatzbedingungen, Ingenieur-Archiv 43 (4) (1974) 183–192. doi:10.1007/BF00534000.

[70] D. Baumgärtner, On the grid-based shape optimization of structures with internal flow and the feedback of shape changes into a CAD model, Accepted, Ph.D. thesis, Technische Universität München (2020).

[71] T. Oberbichler, R. Wüchner, K.-U. Bletzinger, Efficient computation of nonlinear isogeometric elements using the adjoint method and algorithmic differentiation, Computer Methods in Applied Mechanics and Engineering 381 (2021). doi:10.1016/j.cma.2021.113817.

[72] K. Linkwitz, H. J. Schek, Einige Bemerkungen zur Berechnung von vorgespannten Seilnetzkonstruktionen, Ingenieur-archiv 40 (3) (1971). doi:10.1007/BF00532146.

[73] K.-U. Bletzinger, E. Ramm, A General Finite Element Approach to the form Finding of Tensile Structures by the Updated Reference Strategy, International Journal of Space Structures 14 (2) (1999) 131–145. doi:10.1260/0266351991494759.

[74] E. Catmull, J. Clark, Recursively generated B-spline surfaces on arbitrary topological meshes, Computer-Aided Design 10 (6) (1978) 350–355. doi:10.1016/0010-4485(78)90110-0.

[75] C. Loop, Smooth Subdivision Surfaces Based on Triangles (8 1987).

[76] J. Stam, Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values, in: Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98, ACM Press, New York, New York, USA, 1998, pp. 395–404. doi:10.1145/280814.280945.

[77] H. Biermann, A. Levin, D. Zorin, Piecewise Smooth Subdivision Surfaces with Normal Control, in: Proceedings of the 27th Annual Conference

on Computer Graphics and Interactive Techniques, ACM Press/Addison-Wesley Publishing Co., 2000, pp. 113–120.

[78] F. Cirak, M. Ortiz, P. Schröder, Subdivision surfaces: a new paradigm for thin-shell finite-element analysis, International Journal for Numerical Methods in Engineering 47 (2000) 2039–2072.

[79] F. Cirak, M. Ortiz, Fully C1-conforming subdivision elements for finite deformation thin-shell analysis, International Journal for Numerical Methods in Engineering 51 (7) (2001) 813–833. doi:10.1002/NME.182.

[80] F. Cirak, M. J. Scott, E. K. Antonsson, M. Ortiz, P. Schröder, Integrated Modeling, Finite-Element Analysis, and Engineering Design for Thin-Shell Structures using Subdivision, Computer-Aided Design 34 (2002) 2002. doi:10.1016/S0010-4485(01)00061-6.

[81] F. Cirak, Q. Long, Subdivision shells with exact boundary control and non-manifold geometry, International Journal for Numerical Methods in Engineering 88 (9) (2011) 897–923. doi:10.1002/nme.3206.

[82] D. Burkhart, B. Hamann, G. Umlauf, Iso-geometric Finite Element Analysis Based on Catmull-Clark Subdivision Solids, International Journal for Numerical Methods in Engineering 29 (5) (2010). doi:10.1111/j.1467-8659.2010.01766.x.

[83] A. Wawrzinek, K. Hildebrandt, K. Polthier, Koiter's thin shells on catmull-clark limit surfaces, VMV 2011 - Vision, Modeling and Visualization (2011) 113–120doi:10.2312/PE/VMV/VMV11/113-120.

[84] P. J. Barendrecht, Isogeometric Analysis for Subdivision Surfaces (2013).

[85] P. Barendrecht, Splines for engineers: with selected applications in numerical methods and computer graphics, Ph.D. thesis, University of Groningen (12 2019). doi:10.33612/diss.102688532.

[86] P. J. Barendrecht, M. Bartoň, J. Kosinka, Efficient quadrature rules for subdivision surfaces in isogeometric analysis, Computer Methods in Applied Mechanics and Engineering 340 (2018) 1–23. doi:10.1016/j.cma.2018.05.017.

[87] Q. Pan, C. Chen, G. Xu, Isogeometric finite element approximation of minimal surfaces based on extended loop subdivision, Journal of Computational Physics 343 (8 2017). doi:10.1016/j.jcp.2017.04.030.

[88] Q. Pan, T. Rabczuk, G. Xu, C. Chen, Isogeometric analysis for surface PDEs with extended Loop subdivision, Journal of Computational Physics 398 (2019) 108892. doi:10.1016/j.jcp.2019.108892.

[89] Q. Pan, T. Rabczuk, C. Chen, Subdivision based isogeometric analysis for geometric flows, International Journal for Numerical Methods in Engineering 123 (2) (2022) 610–633. doi:10.1002/nme.6870.

[90] Q. Pan, G. Xu, Construction of Minimal Catmull-Clark's Subdivision Surfaces with Given Boundaries, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 6130 LNCS (2010) 206–218. doi:10.1007/978-3-642-13411-1_14.

[91] A. Wawrzinek, On Isoparametric Catmull-Clark Finite Elements for Mean Curvature Flow, Ph.D. thesis, Freie Universität Berlin, Berlin (11 2016).

[92] Q. Pan, T. Rabczuk, C. Chen, G. Xu, K. Pan, Isogeometric analysis of minimal surfaces on the basis of extended Catmull–Clark subdivision, Computer Methods in Applied Mechanics and Engineering 337 (8 2018). doi:10.1016/j.cma.2018.03.040.

[93] J. M. F. Linhard, Numerisch-mechanische Betrachtung des Entwurfsprozesses von Membrantragwerken, Ph.D. thesis, Technische Universität München (2009).

[94] B. Philipp, Methodological Treatment of Non-linear Structural Behavior in the Design, Analysis and Verification of Lightweight Structures, Ph.D. thesis, Technische Universität München (2017).

[95] K. Washizu, Variational Methods in Elasticity and Plasticity, 3rd Edition, Pergamon Press, New York, 1984.

[96] R. Wüchner, K.-U. Bletzinger, Stress-adapted numerical form finding of pre-stressed surfaces by the updated reference strategy, International Journal for Numerical Methods in Engineering 64 (2) (2005) 143–166. doi:10.1002/nme.1344.

[97] C. Loop, Second order smoothness over extraordinary vertices, in: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing - SGP '04, ACM Press, New York, New York, USA, 2004, pp. 165–174. doi:10.1145/1057432.1057454.

[98] Q. Pan, G. Xu, G. Xu, Y. Zhang, Isogeometric analysis based on extended Catmull–Clark subdivision, Computers & Mathematics with Applications 71 (1) (2016) 105–119. doi:10.1016/j.camwa.2015.11.012.

[99] Q. Pan, T. Rabczuk, X. Yang, Subdivision-based isogeometric analysis for second order partial differential equations on surfaces, Computational Mechanics 68 (5) (2021) 1205–1221. doi:10.1007/s00466-021-02065-7.

[100] C. W. B. Goldschmidt, Determinatio superficiei minimae rotatione curvae data duo puncta jungentis circa datum axem ortae, typis Dieterichianis, 1831.

Bisherige Titel der Schriftenreihe

Band	Titel
1	Frank Koschnick, *Geometrische Lockingeffekte bei Finiten Elementen und ein allgemeines Konzept zu ihrer Vermeidung*, 2004.
2	Natalia Camprubi, *Design and Analysis in Shape Optimization of Shells*, 2004.
3	Bernhard Thomee, *Physikalisch nichtlineare Berechnung von Stahlfaserbetonkonstruktionen*, 2005.
4	Fernaß Daoud, *Formoptimierung von Freiformschalen - Mathematische Algorithmen und Filtertechniken*, 2005.
5	Manfred Bischoff, *Models and Finite Elements for Thin-walled Structures*, 2005.
6	Alexander Hörmann, *Ermittlung optimierter Stabwerkmodelle auf Basis des Kraftflusses als Anwendung plattformunabhängiger Prozesskopplung*, 2006.
7	Roland Wüchner, *Mechanik und Numerik der Formfindung und Fluid-Struktur-Interaktion von Membrantragwerken*, 2006.
8	Florian Jurecka, *Robust Design Optimization Based on Metamodeling Techniques*, 2007.
9	Johannes Linhard, *Numerisch-mechanische Betrachtung des Entwurfsprozesses von Membrantragwerken*, 2009.
10	Alexander Kupzok, *Modeling the Interaction of Wind and Membrane Structures by Numerical Simulation*, 2009.
11	Bin Yang, *Modified Particle Swarm Optimizers and their Application to Robust Design and Structural Optimization*, 2009.

Band	**Titel**
12	Michael Fleischer, *Absicherung der virtuellen Prozesskette für Folgeoperationen in der Umformtechnik*, 2009.
13	Amphon Jrusjrungkiat, *Nonlinear Analysis of Pneumatic Membranes - From Subgrid to Interface*, 2009.
14	Alexander Michalski, *Simulation leichter Flächentragwerke in einer numerisch generierten atmosphärischen Grenzschicht*, 2010.
15	Matthias Firl, *Optimal Shape Design of Shell Structures*, 2010.
16	Thomas Gallinger, *Effiziente Algorithmen zur partitionierten Lösung stark gekoppelter Probleme der Fluid-Struktur-Wechselwirkung*, 2011.
17	Josef Kiendl, *Isogeometric Analysis and Shape Optimal Design of Shell Structures*, 2011.
18	Joseph Jordan, *Effiziente Simulation großer Mauerwerksstrukturen mit diskreten Rissmodellen*, 2011.
19	Albrecht von Boetticher, *Flexible Hangmurenbarrieren: Eine numerische Modellierung des Tragwerks, der Hangmure und der Fluid-Struktur-Interaktion*, 2012.
20	Robert Schmidt, *Trimming, Mapping, and Optimization in Isogeometric Analysis of Shell Structures*, 2013.
21	Michael Fischer, *Finite Element Based Simulation, Design and Control of Piezoelectric and Lightweight Smart Structures*, 2013.
22	Falko Hartmut Dieringer, *Numerical Methods for the Design and Analysis for Tensile Structures*, 2014.
23	Rupert Fisch, *Code Verification of Partitioned FSI Environments for Lightweight Structures*, 2014.

Band	**Titel**
24	Stefan Sicklinger, *Stabilized Co-Simulation of Coupled Problems Including Fields and Signals*, 2014.
25	Madjid Hojjat, *Node-based parametrization for shape optimal design*, 2015.
26	Ute Israel, *Optimierung in der Fluid-Struktur-Interaktion - Sensitivitätsanalyse für die Formoptimierung auf Grundlage des partitionierten Verfahrens*, 2015.
27	Electra Stavropoulou, *Sensitivity analysis and regularization for shape optimization of coupled problems*, 2015.
28	Daniel Markus, *Numerical and Experimental Modeling for Shape Optimization of Offshore Structures*, 2015.
29	Pablo Suárez, *Design Process for the Shape Optimization of Pressurized Bulkheads as Components of Aircraft Structures*, 2015.
30	Armin Widhammer, *Variation of Reference Strategy - Generation of Optimized Cutting Patterns for Textile Fabrics*, 2015.
31	Helmut Masching, *Parameter Free Optimization of Shape Adaptive Shell Structures*, 2016.
32	Hao Zhang, *A General Approach for Solving Inverse Problems in Geophysical Systems by Applying Finite Element Method and Metamodel Techniques*, 2016.
33	Tianyang Wang, *Development of Co-Simulation Environment and Mapping Algorithms*, 2016.
34	Michael Breitenberger, *CAD-integrated Design and Analysis of Shell Structures*, 2016.

Band	**Titel**
46	Daniel Baumgärtner, *On the Grid-based Shape Optimization of Structures with Internal Flow and the Feedback of Shape Changes into a CAD Model*, 2020.
47	Mohamed Khalil, *Combining Physics-based models and machine learning for an Enhanced Structural Health Monitoring*, 2021.
48	Long Chen, *Gradient Descent Akin Method*, 2021.
49	Aditya Ghantasala, *Coupling Procedures for Fluid-Fluid and Fluid-Structure Interaction Problems Based on Domain Decomposition Methods*, 2021.
50	Ann-Kathrin Goldbach, *The CAD-Integrated Design Cycle for Structural Membranes*, 2021.
51	Iñigo Pablo López Canalejo, *A Finite-Element Transonic Potential Flow Solver with an Embedded Wake Approach for Aircraft Conceptual Design*, 2022.
52	Mayu Sakuma, *An Application of Multi-Fidelity Uncertainty Quantification for Computational Wind Engineering*, 2022.
53	Suneth Warnakulasuriya, *Development of Methods for Finite Element-Based Sensitivity Analysis and Goal-Directed Mesh Refinement Using the Adjoint Approach for Steady and Transient Flows*, 2022.
54	Klaus Bernd Sautter, *Modeling and Simulation of Flexible Protective Structures by Coupling Particle and Finite Element Methods*, 2022.
55	Efthymios Papoutsis, *On the incorporation of industrial constraints in node-based optimization for car body design*, 2023.
56	Thomas Josef Oberbichler, *A modular and efficient implementation of isogeometric analysis for the interactive CAD-integrated design of lightweight structures*, 2023.