Integrated structural analysis using isogeometric finite element methods

Integrierte Tragwerksanalysen mittels isogeometrischer Finite-Elemente-Methoden

DISSERTATION

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

eingereicht an der
Fakultät Bauingenieurwesen
der Bauhaus-Universität Weimar

vorgelegt von
M. Sc. Michael Schwedler
geboren am 21.09.1978 in Berlin

Weimar, April 2016

Mentor:
Prof. Dr.-Ing. habil. Carsten Könke, Bauhaus-Universität Weimar

Gutachter:
Prof. Dr. rer. nat. Ernst Rank, Technische Universität München
Prof. Dr.-Ing. Karl Beucke, Bauhaus-Universität Weimar

Tag der Disputation: 21. Oktober 2016

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Strukturmechanik der Bauhaus-Universität Weimar. Der darin dokumentierte Forschungsbeitrag steht in einem engen Zusammenhang mit der Bearbeitung des DFG-Einzelprojektes „Integrierte Tragwerksanalysen mittels Bauwerksinformationsmodellen und heterogen adaptiver isogeometrischer Finite-Elemente-Methoden", für dessen finanzielle Förderung ich der Deutschen Forschungsgemeinschaft zu großem Dank verpflichtet bin.

Bei der Umsetzung meiner wissenschaftlichen Vorhaben wurde mir mannigfaltige Unterstützung zuteil. Hierfür möchte ich mich an dieser Stelle ausdrücklich bei allen bedanken, die mir über die Jahre wissenschaftlich oder persönlich zur Seite gestanden sind.

Mein besonderer Dank gilt zuvorderst Professor Carsten Könke für die Anregung zu diesem Forschungsthema und die sehr gute wissenschaftliche Betreuung. Ohne die zahlreichen fachlichen Diskussionen wäre ein Gelingen der Arbeit so nicht möglich gewesen. Darüber hinaus danke ich ihm für die vielfältigen Möglichkeiten, die er mir an seinem Lehrstuhl eröffnet hat. Die mir dabei gewährten Freiheiten wusste ich als großes in mich gesetztes Vertrauen immer sehr zu schätzen.

Besonders bedanken möchte ich mich zudem bei Professor Ernst Rank und Professor Karl Beucke für die Bereitschaft zur Begutachtung meiner Dissertation. Ihr Interesse an meiner Forschung ist mir eine große Ehre, ihre wertvollen Anmerkungen haben einen signifikanten Anteil an der Qualität der Arbeit. Professor Karl Beucke danke ich darüber hinaus dafür, mich vor langer Zeit überhaupt erst zu einer wissenschaftlichen Laufbahn in Weimar ermutigt zu haben.

Mein herzlicher Dank gilt auch allen Mitarbeitern des Instituts für Strukturmechanik. Die freundschaftliche Arbeitsatmosphäre sowie die allgegenwärtige Unterstützung haben nicht nur ihren Anteil am Gelingen der Arbeit, sondern lassen mich auch die Jahre am Institut immer in bester Erinnerung behalten. Unzählige fachlich inspirierte Diskussionen mit meinen langjährigen Kollegen Albrecht, Andrea, Daniel, Heiko, Maik und Michael sowie meinen Bürogenossen Ingmar und Philipp haben meinen Alltag enorm bereichert. Nicht unerwähnt bleiben darf auch Frau Terber, die auf einzigartige Weise das Institut zusammenhält und deren Organisationstalent und Herzlichkeit ich sehr zu schätzen gelernt habe.

Von ganzem Herzen möchte ich mich bei meiner Familie und meinen Freunden für den Rückhalt, das Vertrauen und die nötige Ablenkung bedanken. Insbesondere meine Eltern haben mich über viele Jahre in allen Lebenslagen unterstützt, gefördert und an mich geglaubt. Ein ganz großer Dank gebührt meinem Partner Thorsten. Er hat mich nicht nur motiviert, dieses Promotionsvorhaben in Angriff zu nehmen, sondern bot mir über dessen Dauer auch das Verständnis, den Rückhalt, die Motivation und Unterstützung, die für die Vollendung der Arbeit unerlässlich waren.

Berlin, im Dezember 2016 *Michael Schwedler*

Abstract

The gradual digitization in the architecture, engineering, and construction industry over the past fifty years led to an extremely heterogeneous software environment, which today is embodied by the multitude of different digital tools and proprietary data formats used by the many specialists contributing to the design process in a construction project. Though these projects become increasingly complex, the demands on financial efficiency and the completion within a tight schedule grow at the same time. The digital collaboration of project partners has been identified as one key issue in successfully dealing with these challenges. Yet currently, the numerous software applications and their respective individual views on the design process severely impede that collaboration.

An approach to establish a unified basis for the digital collaboration, regardless of the existing software heterogeneity, is a comprehensive digital building model contributed to by all projects partners. This type of data management known as building information modeling (BIM) has many benefits, yet its adoption is associated with many difficulties and thus, proceeds only slowly. One aspect in the field of conflicting requirements on such a digital model is the cooperation of architects and structural engineers. Traditionally, these two disciplines use different abstractions of reality for their models that in consequence lead to incompatible digital representations thereof.

The onset of isogeometric analysis (IGA) promised to ease the discrepancy in design and analysis model representations. Yet, that initial focus quickly shifted towards using these methods as a more powerful basis for numerical simulations. Furthermore, the isogeometric representation alone is not capable of solving the model abstraction problem. It is thus the intention of this work to contribute to an improved digital collaboration of architects and engineers by exploring an integrated analysis approach on the basis of an unified digital model and solid geometry expressed by splines. In the course of this work, an analysis framework is developed that utilizes such models to automatically conduct numerical simulations commonly required in construction projects. In essence, this allows to retrieve structural analysis results from BIM models in a fast and simple manner, thereby facilitating rapid design iterations and profound design feedback.

The BIM implementation Industry Foundation Classes (IFC) is reviewed with regard to its capabilities of representing the unified model. The current IFC schema strongly supports the use of redundant model data, a major pitfall in digital collaboration. Additionally, it does not allow to describe the geometry by volumetric splines. As the pursued approach builds upon a unique model for both, architectural and structural design, and furthermore requires solid geometry, necessary schema modifications are suggested.

Structural entities are modeled by volumetric NURBS patches, each of which constitutes an individual subdomain that, with regard to the analysis, is incompatible with the remaining full model. The resulting consequences for numerical simulation are elaborated in this work. The

individual subdomains have to be weakly coupled, for which the mortar method is used. Different approaches to discretize the interface traction fields are implemented and their respective impact on the analysis results is evaluated. All necessary coupling conditions are automatically derived from the related geometry model.

The weak coupling procedure leads to a linear system of equations in saddle point form, which, owed to the volumetric modeling, is large in size and, the associated coefficient matrix has, due to the use of higher degree basis functions, a high bandwidth. The peculiarities of the system require adapted solution methods that generally cause higher numerical costs than the standard procedures for symmetric, positive-definite systems do. Different methods to solve the specific system are investigated and an efficient parallel algorithm is finally proposed.

When the structural analysis model is derived from the unified model in the BIM data, it does in general initially not meet the requirements on the discretization that are necessary to obtain sufficiently accurate analysis results. The consequently necessary patch refinements must be controlled automatically to allow for an entirely automatic analysis procedure. For that purpose, an empirical refinement scheme based on the geometrical and possibly mechanical properties of the specific entities is proposed. The level of refinement may be selectively manipulated by the structural engineer in charge. Furthermore, a Zienkiewicz-Zhu type error estimator is adapted for the use with isogeometric analysis results. It is shown that also this estimator can be used to steer an adaptive refinement procedure.

Kurzfassung

Die sich über die vergangenen 50 Jahre erstreckende, schrittweise erfolgte Digitalisierung in der Bauindustrie hat zu einem besonders uneinheitlichen Softwaremarkt geführt. Dieser wird von der Vielzahl verschiedener Programme und geschützter Datenformate verkörpert, welche die beteiligten Planer und Ausführenden eines Bauprojekts verwenden. Obwohl Bauprojekte zunehmend komplexer werden, steigen gleichzeitig die Anforderungen hinsichtlich der Kosteneffektivität und des Ablaufs innerhalb eines engen Zeitkorsetts. Einen wichtigen Beitrag diesen gestiegenen Erwartungen gerecht zu werden, soll die digitale Zusammenarbeit aller Projektbeteiligten leisten. Jedoch wird eine solche Zusammenarbeit derzeit von der Heterogenität des Softwareumfeldes massiv behindert.

Einen Ansatz, um unabhängig von der Softwareheterogenität eine einheitliche Basis für die digitale Zusammenarbeit zu schaffen, stellt ein umfassendes digitales Gebäudemodell dar, zu welchem alle Projektbeteiligten gleichermaßen beitragen. Diese als Bauwerks-Informations-Modellierung (BIM) bekannte Art des Datenmanagements hat zahlreiche Vorteile; die praktische Umsetzung ist jedoch mit vielen Schwierigkeiten verbunden und erfolgt daher nur langsam. Einen Aspekt im Spannungsfeld konkurrierender Anforderungen an ein solches digitales Modell stellt die Zusammenarbeit von Architekten und Tragwerksplanern dar. Traditionell verwenden diese beiden Planungsdisziplinen unterschiedliche Abstraktionen der Realität, was in der Folge jedoch zu Inkompatibilitäten bei den digitalen Modellen führt.

Die Einführung isogeometrischer Verfahren (IGA) versprach, die Widersprüche in der Modellbeschreibung von Architektur- und Tragwerksentwurf abzuschwächen. Diese anfängliche Zielsetzung verschob sich jedoch schnell hin zu einer Verwendung der Verfahren als Grundlage leistungsfähigerer numerischer Berechnungen. Isogeometrische Geometriedarstellungen allein würden das Problem unterschiedlicher Modellabstraktionen jedoch ohnehin nicht lösen. Zielsetzung dieser Arbeit ist es daher, mit einem integrierten Tragwerksanalyseverfahren auf der Basis eines einheitlichen Datenmodells und volumetrischer, Spline-basierter Geometriebeschreibungen einen Beitrag zur Verbesserung der digitalen Zusammenarbeit von Architekten und Tragwerksplanern zu leisten. Im Rahmen der Arbeit wird eine Methodik entwickelt, welche auf Grundlage genannter Modelle eine automatische numerische Berechnung der Strukturen üblicher Bauprojekte zulässt. Im Wesentlichen ermöglicht das Verfahren, die Ergebnisse von strukturmechanischen Berechnungen auf schnelle und einfache Weise aus Bauwerksinformationsmodellen herzuleiten und somit rasche Entwurfsiterationen sowie fundierte Entwurfskritiken zu ermöglichen.

Der BIM-Standard Industry Foundation Classes (IFC) wird hinsichtlich seiner Eignung zur Abbildung eines vereinheitlichten Modells untersucht. Gegenwärtig fördert das IFC Schema die redundante Speicherung von Modelldaten, was mit Hinsicht auf die digitale Zusammenarbeit ein gravierender Nachteil ist. Zudem wird die Beschreibung von Geometrieobjekten mit volumetrischen Spline-Formulierungen nicht unterstützt. Da das verfolgte Analyseverfahren jedoch auf einem eindeutigen Modell für den Architektur- und Tragwerksentwurf beruht und

zudem entsprechende volumetrische Geometriebeschreibungen erfordert, sind Änderungen am Schema erforderlich, die im Rahmen der Arbeit vorgeschlagen werden.

Tragwerkselemente werden durch dreidimensionale NURBS-Patches modelliert. Jedes einzelne stellt einen Teilbereich des Gesamtmodells dar, welcher hinsichtlich der Analyse jedoch nicht mit dem restlichen Modell kompatibel ist. Dies hat Konsequenzen für die numerische Berechnung, welche im Verlauf der Arbeit erörtert werden. Die verschiedenen Teilbereiche sind mit einem schwachen Verfahren zu koppeln, wofür die Mortarmethode Verwendung findet. Im Bereich der Kopplungsstellen werden die Felder der Kontaktspannungen auf unterschiedliche Weise diskretisiert. In der Folge wird ausgewertet, wie sich die verwendeten Diskretisierungsansätze auf die Ergebnisse der Berechnung auswirken. Alle erforderlichen Kopplungsbeziehungen werden automatisiert aus der Geometriebeschreibung des jeweiligen Modells hergeleitet.

Die schwache Kopplung der Patches hat auch Einfluss auf das zu lösende Gleichungssystem, welches infolge des Verfahrens als Sattelpunktproblem vorliegt. Aufgrund der dreidimensionalen Geometrieobjekte und der Verwendung höherer Ansatzgrade bei den Basisfunktionen ist das Gleichungssystem verhältnismäßig groß; zudem hat die zugehörige Koeffizientenmatrix eine hohe Bandbreite. Die Eigenheiten des Systems erfordern angepasste Lösungsverfahren, welche grundsätzlich einen höheren Aufwand erfordern, als es die Standardverfahren für symmetrisch positiv-definite Systeme tun. Es werden einige Ansätze zur Lösung derartiger Systeme untersucht und auf dieser Grundlage ein effizientes paralleles Verfahren vorgestellt.

Die strukturmechanischen Modelle, welche aus den BIM-Datensätzen hergeleitet werden, erfüllen hinsichtlich ihrer Diskretisierung anfänglich nicht die Voraussetzungen, um die notwendige Genauigkeit bei den Berechnungsergebnissen zu erzielen. Für die Realisierung einer vollautomatischen Berechnung ist es somit erforderlich, auch die Verfeinerung der einzelnen Patches automatisiert zu steuern. Zu diesem Zweck wird ein empirisches Verfeinerungsschema vorgeschlagen, welches auf den geometrischen und bei Bedarf auch den mechanischen Eigenschaften der einzelnen Tragelemente beruht. Der Grad der Verfeinerung kann zudem gezielt von dem bearbeitenden Ingenieur beeinflusst werden. Darüber hinaus wird ein Fehlerschätzer auf der Basis von Zienkiewicz und Zhu für die Verwendung mit den isogeomtrischen Verfahren angepasst. Es wird gezeigt, dass auch dieser Fehlerschätzer zur Steuerung eines adaptiven Verfeinerungsverfahrens verwendet werden kann.

Contents

Nomenclature

Abbreviations

2D	two-dimensional
3D	three-dimensional
AABB	axis aligned bounding box
AECO	architecture, engineering, construction and operations
API	application programming interface
BiCGstab	bi-conjugate gradient stabilized method
BIM	building information modeling
BRep	boundary representation
BVP	boundary value problem
CAD	computer aided design
CAM	computer aided manufacturing
ccNUMA	cache-coherent non-uniform memory access
CG	computer graphics
CG	conjugate gradient method
COO	coordinate list storage format for sparse matrices
CPU	central processing unit
CSC	compressed sparse column storage format for sparse matrices
CSG	constructive solid geometry
DIAG	block diagonal precondition operator using the diagonal and the Schur complement of a 2×2 block matrix
DMP	distributed memory processing
DOF	degree of freedom
ECM	enterprise content management
FEA	finite element analysis
FEM	finite element method
FETI-DP	finite element tearing and interconnecting domain decomposition method, dual-primal variant
GMRES	generalized minimal residual method
GPU	graphics processing unit
GUI	graphical user interface
ID	identifier, a name or number uniquely identifying an object

IDDS	integrated design and delivery solutions
IETI	isogeometric tearing and interconnecting domain decomposition method
IFC	industry foundation classes
IGA	isogeometric analysis
ILU	incomplete LU factorization
ILU(0)	zero-fill incomplete LU factorization
ILUTP	dual threshold incomplete LU factorization with pivoting
ISO	International Organization for Standardization
LM	Lagrange multiplier
MINRES	minimum residual method
MPI	message-passing interface API
MVD	model view definition
NURBS	non-uniform rational B-splines
OBB	oriented bounding box
OpenMP	open multi-processing API
PDM	product data management
PLM	product lifecycle management
RAM	random access memory
RCM	reverse Cuthill-McKee reordering
SMP	symmetric multi-processor
SQMR	symmetric quasi-minimal residual method
STEP	standard for the exchange of product model data
SYMMLQ	symmetric LQ method
XML	extensible markup language

Symbols

Capital Greek Letters

Γ	boundary of a body
$\Gamma_c^{(i,j)}$	inter subdomain boundary of subdomains i and j
Γ_t	boundary with Neumann boundary conditions
Γ_u	boundary with Dirichlet boundary conditions
Ω	domain of a body in physical space
$\tilde{\Omega}_e$	domain of a finite element in natural coordinate space
$\hat{\Omega}$	domain of a body in parametric space

$\Omega^{(i)}$	subdomain of a body, i.e. a single NURBS patch
$\mathbf{H}$	knot vector of the parametric η direction
$\mathbf{\Xi}$	knot vector of the parametric ξ direction
$\mathbf{Z}$	knot vector of the parametric ζ direction

Small Greek Letters

α, β, ω	scaling factors
δ_{ij}	Kronecker delta
κ	condition number
λ	first Lamé constant
μ	second Lamé constant
ν	poisson ratio
ρ	material mass density
ξ, η, ζ	parametric coordinates
$\boldsymbol{\epsilon}$	linear or infinitesimal strain tensor
$\underline{\boldsymbol{\epsilon}}$	linear strain vector, Voigt notation of $\boldsymbol{\epsilon}$
$\boldsymbol{\lambda}$	Lagrange multiplier vector
$\tilde{\boldsymbol{\lambda}}^{c,(i,j)}$	vector of discrete Lagrange multiplier values for the coupling interface of subdomains i and j
$\boldsymbol{\sigma}$	linear stress tensor
$\underline{\boldsymbol{\sigma}}$	linear stress vector, Voigt notation of $\boldsymbol{\sigma}$
$\boldsymbol{\xi}$	position vector in parametric coordinates

Capital Latin Letters

C^i	class of functions with i continuous differentiations
E	Young's modulus
N	basis function, shape function
P	material point
$\hat{P}^{(a)}$	nodal point a of a finite element
$\hat{P}_m^{(a)}$	mortar node a
R	rational basis function, rational shape function
$\boldsymbol{B}$	strain displacement matrix

$\boldsymbol{C}$	parametric curve
$\boldsymbol{D}$	linear elastic material tensor
$\underline{\boldsymbol{D}}$	linear elastic material matrix
$\boldsymbol{E}$	Green-Lagrange strain tensor
$\boldsymbol{H}$	displacement gradient
$\boldsymbol{J}$	Jacobian matrix
$\boldsymbol{K}^e$	element stiffness matrix
$\boldsymbol{K}$	global stiffness matrix
$\boldsymbol{N}$	matrix of shape functions of a finite element
$\boldsymbol{N}^{c,(i,j)}$	matrix of Lagrange muliplier field interpolation functions associated with the coupling interface of subdomains i and j
$\boldsymbol{P}$	control point, i.e. a position vector in physical coordinate space
$\boldsymbol{Q}$	control variable, control point equiavalent for storing the global solution values of a finite element analysis
$\boldsymbol{S}$	parametric surface
$\boldsymbol{T}$	Cauchy stress tensor, also control variable for the stress
$\boldsymbol{V}$	parametric solid (volume)
$\boldsymbol{X}$	position vector of the deformed configuration in physical coordinates
A	block diagonal matrix of subdomain stiffness matrices
B	block matrix of subdomain mortar matrices
LU	matrix decomposition into an upper U and lower L triangular matrix
S	Schur operator
$\boldsymbol{I}$	identity matrix
$\boldsymbol{K}$	coefficient matrix of the saddle point problem
$\boldsymbol{M}$	precondition operator for the saddle point problem
$\mathcal{B}$	deformable, continuous body
$\boldsymbol{\mathcal{L}}$	differential operator matrix
$\mathbb{R}^d$	d-dimensional real coordinate space

Small Latin Letters

e_i	eigenvalue i
$f(\cdot)$	function of $\cdot$
n_{en}	number of nodal points of a finite element

n_{ip}	number of element integration points
n_{mn}	number of mortar nodes
n_{sub}	number of subdomains in a multi-patch setting
p, q, r	polynomial degrees for three independent directions
r, s, t	natural coordinates
w	weight, in Gauss quadrature and as $(d+1)^{th}$ scalar component of a position vector in homogeneous space
x, y, z	physical coordinates
$\boldsymbol{b}$	body load (density)
$\boldsymbol{e}^h$	discretization error
$\check{\boldsymbol{e}}^h$	approximation of the discretization error
$\boldsymbol{e}_i$	orthonormal basis vector i of the Euclidean space
$\boldsymbol{f}$	force vector
$\tilde{\boldsymbol{f}}^e$	vector of all nodal point force vectors of a finite element
$\tilde{\boldsymbol{f}}$	global vector of all nodal point force vectors
$\tilde{\boldsymbol{g}}$	global vector of all control point force vectors
$\boldsymbol{m}^{(i)}$	mortar matrix for subdomain i
$\boldsymbol{n}$	unit outward normal vector
$\tilde{\boldsymbol{q}}^e$	vector of all control variables with support in a finite element
$\tilde{\boldsymbol{q}}$	global vector of all control variables
$\boldsymbol{r}$	position vector in natural coordinates
$\boldsymbol{t}$	traction vector
$\boldsymbol{u}$	displacement vector
$\hat{\boldsymbol{u}}^{(a)}$	displacement value at the nodal point a of a finite element
$\tilde{\boldsymbol{u}}^e$	vector of all nodal point displacement vectors of a finite element
$\tilde{\boldsymbol{u}}$	global vector of all nodal point displacement vectors
$\boldsymbol{x}$	initial position vector in physical coordinates
$\tilde{\boldsymbol{x}}^e$	vector of all nodal point position vectors of a finite element
$\mathbf{f}$	block vector of subdomain force vectors
$\mathbf{u}$	block vector of subdomain displacement vectors
$\boldsymbol{f}$	right hand side of the saddle point problem
$\boldsymbol{r}$	residual vector of the saddle point problem
$\boldsymbol{u}$	solution vector for the saddle point problem

Subscripts and Superscripts

e	finite element
h	finite dimensional approximation
ic	integration cell
seg	mortar segment
w	homogeneous coordinate space $\mathbb{R}^{d+1}$
(a)	finite element nodal point a

Mathematical operators

$\det(\cdot)$	determinant of $\cdot$
$\mathrm{nnz}(\cdot)$	number of nonzero elements in $\cdot$
$\mathrm{span}(\cdot)$	linear span, i.e. the set of all finite linear combinations of $\cdot$
$\|\cdot\|_E$	energy norm
$\|\cdot\|_2$	ℓ_2-norm
$(\cdot)_{,\alpha}$	partial differentiation with respect to α

Chapter 1

Introduction

1.1 Motivation

Beginning in the 1960s, the design process of construction projects has been gradually digitized. With the availability of personal computers in the 1980s, the use of software tools became a common practice in engineering consultancies. Since then, myriad applications for almost every aspect of the design, the construction, and lately also the operation phase were developed, sold and used. Each of them covers a very specific domain. There are tools for drawing architectural layouts, conducting structural analyses, drawing formwork and reinforcement plans, performing fire and evacuation simulations, planning building services, doing cost analyses and time scheduling, writing tenders, et cetera. The interaction of these applications on a data or software level is limited. In some cases, large software companies have built an integrated ecosystem around their own set of tools. Yet, the focus of such an ecosystem is limited and its integration is of no value, when different project partners use different software products (cf. Amor [7]).

The tools used by designers changed with time. Pen and paper were superseded by computers and the software running on them. Yet, the product remained the same – two-dimensional (2D) drawings containing "plans, sections, and elevations" [76, p. 23]. Reasons identified by Günthner and Borrmann [76] are the legal binding of printed 2D plots and the circumstance of requiring plotted plans for the execution of the construction on site, as no means of three-dimensional (3D) visualization are available for that purpose. Project partners exchange these 2D drawings, either in paper or in digital form, and integrate the vital information received from the others into their isolated digital model, where each individual model bases on the software tool that is used by the respective consultant. In most cases, the incorporation of the received information into the model must be performed manually or at least requires tedious manual rework after an automated data import. As design is an iterative process, modifications of the design and therefore changes in the drawings must be communicated, distributed and finally integrated in the various models a number of times. This is time consuming, costly, and error prone. Often, a qualified computational analysis is only done at the end of the design process with all final information at hand. Previous decisions were mostly made on the basis of experience only (cf. Sanguinetti et al. [144]).

Though yielding advanced possibilities especially for complex simulations, most software tools in architecture and engineering are still used to replicate traditional drafting processes. Other fields of industry, like the automotive or aerospace sectors, deployed the available hard and software technology to implement more fundamental changes in their operations. Not only do companies in these industries work extensively with digital three-dimensional models of their

products [76], but also they have digitized most processes. For this purpose, product lifecycle management (PLM) systems provide "all product or facility related information in required quality and at the right time and place" [1] on the basis of digital files and database records. Reaching such a sophisticated state of data management demands to agree on a standardized data representation that fulfills the needs of all involved partners but leaves the freedom to use the software applications of choice.

An approach of establishing product lifecycle management within the architecture, engineering, construction and operations (AECO) industry is building information modeling (BIM) or its more figuratively entitled enhancement integrated design and delivery solutions (IDDS) (cf. Owen et al. [124]). A long list compiled by Azhar [13] identifies high and numerous benefits from BIM. Yet despite these benefits, BIM adoption within the AECO industry has by far not reached a level that matches the expectations set into it. In Germany, the DIN institute just started the standardization process in 2015 with the establishment of the technical committee "Building Information Modeling". A recent Swedish study by Samuelson and Björk [143] also documents the lack of BIM implementation. Reasons for BIM finding its way into industry only slowly originate in the special characteristics of AECO. They include the following, many of them gathered by Eastman [58] and Romberg [136]:

> Buildings and other structures are usually unique – they are designed and built only once. In contrast to mass production, increased costs for the planning must redeem with a single copy of the product.
>
> In each project, a large number of specialists from different fields are involved, all of them having their unique view – and a corresponding model – on the project. In addition, they are usually employed at different consultancies, each working with its own set of tools at temporal and spatial separation from the others.
>
> The complexity and level of detail increase with the evolving project. As contracts with varying consultancies are made for the successive phases of a project, it is not the same specialists from one field that continuously work on a project. Digital models do not evolve but are recreated many times.
>
> Due to the uniqueness of a building, the formalization of the processes in the design and construction phases is extremely difficult. Moreover, modifications in the conditions under which a building is erected occur frequently. Either early assumptions prove to be wrong, the building owner changes its intentions, or regulatory constraints become relevant.
>
> The multitude of software tools used by the many specialists is extremely heterogeneous. Also do the tools evolve very fast when compared with the lifecycle of a building – sometimes even in comparison with the time required to finish the construction. Compatibility quickly becomes an issue. For older buildings, existing data is analog or often in a non-recoverable digital format.

A thorough study on BIM adoption, identifying causes of the current state and future requirements was performed by Gu and London [75]. Despite all existing difficulties, an improved digital cooperation of the partners involved in the planning and execution of a construction project is inevitable. Currently, the overall financial inefficiencies are unacceptably high (cf. Gallaher et al. [67]) and project specific time and budget restrictions are too demanding to handle the increasingly complex projects in the traditional manner. Digital cooperation has the

potential to ease the communication between the separated groups involved in a project, to allow concurrent work on shared project data, and to accelerate design iterations by supporting the feedback to other specialists' project contributions. A BIM framework within which such an improved cooperation may be achieved is sketched by Succar [155].

One aspect in the debate about the conflicting fields of a traditional design workflow on one side and fully digitized, software managed data and processes on the other is the relation between architects and structural engineers. The exchange of data between the design and analysis domain and the mutual provision of design feedback dominate this relation. Highly simplified, it reduces to the architect presenting their ideas regarding the shape of a building and the structural engineer assessing the feasibility and providing suggestions for modifications which in turn lead to new ideas on the side of the architect.

In the process of assessing the feasibility, a structural engineer has to develop the structural design of the building. Traditionally, the engineer extracts the possibly load bearing parts from the architect's building model, decomposes them into individual elements, and assigns them their role in the structural design. For each of these elements, the received and transmitted load is determined and its capacity is assessed by performing a calculation on a dimensionally reduced abstraction of that structural element. Accounting for the interaction between the individual elements thereby depends completely on the engineer. Formerly, these calculations were conducted by hand. Today, this approach is exactly reproduced with individual software modules for any kind of building element abstraction.

The sequential processes of extraction, decomposition, role assignment and abstraction are extremely difficult to formalize for general structures and therefore impede the adoption of BIM. Moreover, they are time consuming and therefore costly, especially as they have to be iteratively repeated for modified architectural designs. Beside the costs, there is another downside to the time-intensive manual work. As, for obvious reasons, the architect continues their own design work during structural design, the data worked on by the engineer is outdated long before results are available. However, hand calculations, the historic reason that originally legitimated the traditional workflow have almost vanished.

Accordingly, modeling and analyzing the structure in its entirety becomes increasingly relevant in the engineering practice (cf. Fastabend et al. [63]). The required computational power is readily available and numerical procedures like the finite element method (FEM) have widely replaced analytical and empirical relations used in hand calculations and software tools replicating these calculations. Assessing the impact of local modifications on the entire structure benefits considerably from analyzing the complete structure within a single model. However, the model remains an assembly of dimensionally reduced structural elements. An in any manner automated transfer of an architects building model into such a structural model, not to mention the reverse for a feedback operation, is not possible, completely independent of the type of building model created by the architect (2D, 3D, or BIM).

Abandoning the dimensional reduction and performing the structural analysis on a 3D solid model is not yet considered an option in engineering practice. However, provided that architects base their work upon corresponding models, it would greatly simplify matters. This idea is confirmed with the doctoral thesis of Niggl [121]. He recognizes significant advantages in the use of volumetric models for structural analysis, especially regarding cooperative work. According to him, the consistency of different models with individual perspectives on a building would be easier to achieve and redundancies could be avoided. A general system architecture

for BIM that focuses on a stronger integration of design and analysis (not restricted to structural analysis) is presented by Sanguinetti et al. [144]. They conclude that there must be only one user defined model within the BIM data – the architects building model. This model must be sufficiently generic to be automatically post-processed to then form the basis of any analysis that is to be conducted. Therefore, of course, the building entities within the BIM model must be supplemented with the specific attributes of the respective analysis domain. This type of mapping operation was already a subject of Rombergs's [136] work which also deals with the structural analysis of solid models. He focuses on deriving geometric models from BIM representations that are suitable for automatic mesh generation and consecutive numerical analysis.

With the proposal of the isogeometric analysis (IGA) concept by Hughes et al. [83], the idea of creating the analysis model from BIM data received another impetus, at least with regard to structural analysis. With this method, geometric representations used in computer aided design (CAD) software may directly be employed for the numerical analysis of a structure. Previously, geometric design with CAD applications relied for mainly historic reasons on geometry formulations that are completely different from the shape representations used within the context of finite elements. Isogeometric analysis was expected to eventually render the transformation of geometric entities between building and structural analysis models superfluous.

1.2 Aims and scope of the work

It is the principal goal of this work to contribute to an improved interoperability between the fields of architecture and structural analysis and thereby supporting a successful implementation of building information modeling. The introduction of the isogeometric analysis concept has caused an extensive research activity in that field. However, the initial idea of "bridging the gap between design and analysis" [43, p. 6] fell somewhat out of focus after finding that in many cases IGA is also capable of delivering better numerical results in a finite element analysis (FEA) than the traditional formulations do. Therefore, it is the intention of this work to revitalize the initial idea by making isogeometric analysis a driving factor of an improved digital cooperation between architects and structural engineers. Following the proposition of using a unified model as the basis of all digital collaboration, it is studied how BIM data with shape representations expressed by spline formulations can be employed in an automated translation of the BIM data into a model eligible for structural analysis. Obtaining the desired results from the initially pure geometric building model in a preferably fast and simple, yet reliable manner is a main concern of this work.

At present, the transformation from a geometric to a structural model constitutes one of the striking shortcomings in the relation between architectural and structural design. The generation of a comprehensive structural model for different conceptual designs is time consuming. As can be seen from fig. 1.1, about 75% of overall engineering time is required to generate the final simulation model from the design input. The necessary steps cannot be automated. Evaluating design alternatives or even exploring the space of possible structural designs in early project phases is thus expensive and accordingly seldom done in a satisfactory manner. Consequently, feedback to an architect's ideas is given by experience, which is rarely optimal. In this work, the situation is to be improved by the automation of many steps in the process chain. A sequence of interconnected methods is developed that renders the automatic model transformation and its subsequent structural analysis not only a possible option but a practice that

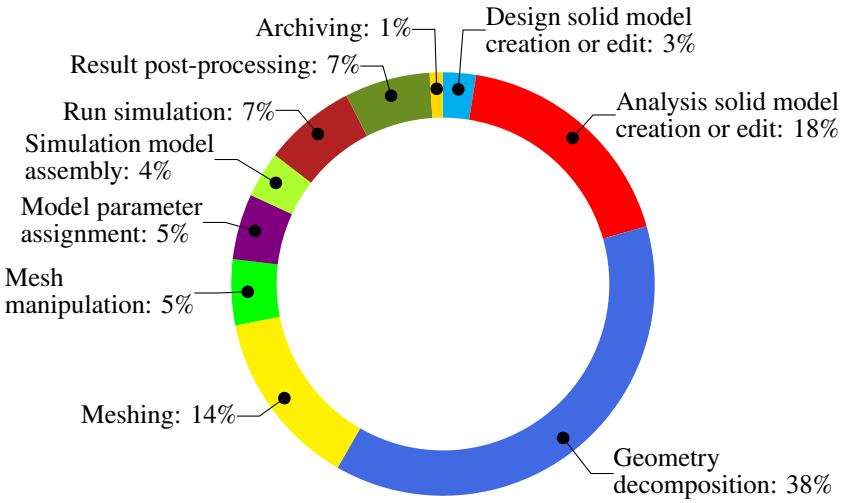


Figure 1.1: Distribution of normalized engineering time required for the process steps in linear and nonlinear structural analysis, equally weighted, according to Hardwick et al. [77]

provides distinct benefits over existing approaches. In sum, these methods constitute a framework for an integrated structural analysis approach. As a proof of concept, the entire procedure is implemented in a stand-alone software application.

As elaborated in the preceding section and the references given therein, the dimensional reduction generally dealt with in structural analysis is an obstacle for the interoperability of design and analysis models. Therefore, all geometric models are required to conform to the physical reality of volumetric bodies. The implications of this requirement are discussed in several places in this thesis.

Though many different spline formulations have existed for a long time or were recently developed in the context of IGA, non-uniform rational B-splines (NURBS) are widely acknowledged as the current standard within CAD software. Hence, they form the natural basis for geometry representations within this work.

For the creation of a structural analysis model, the necessary geometric information must be retrieved from the provided BIM data set. The shapes of the building elements contained in the data are assumed to be defined by one or multiple NURBS patches, a condition that allows for the application of the isogeometric concept. Thus, the mesh already inherent to the patches can be utilized for the finite element formulation and the explicit meshing of the geometry becomes superfluous. As, however, NURBS patches originally serve the purpose of geometry representation, their inherent meshes are in general non-conforming on a multi-patch level. This is addressed by using the mortar method to weakly couple these patches. A necessary condition for the evaluation of the mortar integrals during the coupling is, of course, the knowledge about the areas of NURBS patch contact. Since this information is implicitly contained in the shape representation, a redundant explicit storage in the BIM data is not an acceptable option. Accordingly, the spatial relation of the patches and their areas of contact are established algorithmically. The resulting requirement for the application of the mortar method is geometric compatibility among the patches. Since no other limitations regarding to the degree, the control points or the knot vectors of the NURBS patches apply, there exists great freedom for geometric modeling.

The original purpose of NURBS patches within BIM is to represent geometric entities but not

to provide a finite element discretization thereof. Thus, the initial discretization derived from the patches is unlikely to suffice for the computation of numerical results with desired accuracy. In order to obtain reliable results nonetheless, adaptive mesh refinement utilizing two, possibly supplementary strategies is implemented. In contrast to a traditional finite element analysis, the knowledge of the building elements' geometry is preserved throughout the analysis process. This circumstance is used to formulate empirical refinement rules for each building element. A classic adaptive strategy can be pursued alternatively or to further improve the result quality after applying the empirical rules. Driver for the adaptive refinement is a Zienkiewicz-Zhu type a posteriori error estimator. Based on recovered fields from superconvergent point results, the developed error estimator indicates those elements that require refinement.

For two reasons, some attention has to be paid to the process of solving the linear system of equations. Employing solid formulations for the analysis of an entire structure leads inevitably to a large number of equations and therewith to high computational costs. The use of the mortar method with Lagrange multipliers, moreover, renders the final system of equations a saddle point problem. Solving this system is more complex and in particular different from solving a standard finite element problem in the field of structural mechanics, which commonly yields a symmetric, positive-definite coefficient matrix. Then again, the saddle point problem provides also an opportunity for an efficient parallel solution approach that is to be discussed in this work.

The primary results obtained from the analysis of solid models are displacements with strains and stresses being available after post-processing the primary results. Common structural design regulations, however, frequently demand internal forces as input for the dimensioning. This discrepancy is an issue when using solid models, yet it is beyond the scope of this work and therefore shall be addressed only briefly. Internal forces result from the analysis of dimensionally reduced structural parts. Obviously, there are historic reasons for their use in the regulations. Niggl [121] demonstrates how they can be integrated from stress fields when dealing with solid models. In principle, however, it should be noted, that internal forces are an engineering construct, which could be abandoned in the regulations, when dimensional reduction is superseded by solid model analysis.

1.3 Outline of the work

Chapter 2 gives an insight into product data management in the AECO industry. Different concepts are introduced and their suitability for the integration of the architectural and structural domains is discussed. It is explained, why the desired level of interoperability is difficult to achieve with the current and intended future practice. Propositions for modifications dealing with these difficulties are made. They form the basis of the integrated analysis approach outlined in subsequent chapters.

Chapter 3 provides basic knowledge about the isogeometric concept. The boundary value problem (BVP) of linear elasticity is introduced, followed by a short presentation on the displacement based finite element method as a means of solving the BVP. The chapter proceeds with an overview of spline geometry, in particular of B-splines and NURBS. Emphasis is put on the properties of the basis functions and on operations for their enrichment. Showing isogeometric analysis to be an extension of standard finite element method concludes this chapter.

Chapter 4 introduces the mortar method as a technique that allows the integration of multiple non-conforming meshes in a single analysis. The theoretical background of the method is presented first. The focus of this chapter is yet on the full procedure that is pursued to automatically obtain a valid numerical model from an initially pure geometry description. Several numerical examples are provided at the end of the chapter. The results computed with different mortar approaches are evaluated and discussed.

Different issues regarding the linear system of equations that arises in the result of combining the isogeometric method with the mortar coupling technique are addressed in chapter 5. A large part of that chapter is dedicated to a discussion on parallel solution strategies for the resulting saddle point problem. Other topics are the efficient evaluation of the global stiffness matrix and the postprocessing of analysis results.

Chapter 6 finally discusses appropriate strategies for the enrichment of the numerical model obtained from the geometric representation. The discretization of the initial model suits the requirements of the geometry, but generally it is too coarse to compute results of desired accuracy. Therefore, the model must be refined. Two methodologies to steer the refinement are presented. One is based on the classic recovery error estimator by Zienkiewicz and Zhu that is adapted for IGA, the other bases on the empirical model knowledge of the structural engineer.

Chapter 7 concludes this work with an overview of achieved results and suggestions for further research.

Chapter 2

Integrated structural analysis approach

2.1 Introduction

The motivation for an improved collaboration of structural analysis and architectural design was already given in sect. 1.1 of the previous chapter. Ideally, the analysis and design processes would be completely digitally integrated thus allowing easy communication on any level. A coherent and consistent data basis is a key factor for this. Communication and data exchange can only be reliable, when there is an agreement on the underlying data concept. The involved parties must share the same working basis. Accordingly, a problem is imposed on the collaboration by independent redundant information within a model. The redundant data can easily become unsynchronized which would render the model inconsistent. A common understanding of what is the current model would then be impossible to achieve. For the case of automated communication and data exchange processes, the current model state and modifications thereon must also be identifiable by the software applications used within the project. Defining a software interoperable model and managing the model data throughout the project are therefore two indispensable tasks when integration is desired.

Providing access to up-to-date model data and simultaneously preventing modifications of already outdated model revisions can be handled with data management systems. Within the AECO industry, these systems are established under the name *virtual project room*. They are especially suited for the cross-enterprise use. In addition to managing digital documents over the internet, they often provide collaborative features like shared calendars, messaging, or facilities for ordering plotted drawings. Within these systems, model data is managed on the basis of individual files, CAD drawings for instance, typically stored in the proprietary format of the software application used to create it or in a view-only, fixed-layout derivation thereof. The sum of all files constitutes the building model. Commonly, a database supports the file management by storing for instance file descriptions, relations between files, access rights, and so forth. Also, the data management system may possibly be able to display the file content. However, since apart from that it cannot interpret the files, the enforcement of model consistency remains a user responsibility. Refer to Mersch [114, 115] for details on this topic.

A different approach is pursued with product models. A product, i.e. the building, is no further defined by the content of as many files, but by a unified semantic model of the building. With this approach, the building model is not primarily the building's geometry expressed as an assembly of geometric primitives within CAD drawings, but the sum of objects representing individual informational components of the building project. These objects are hierarchically

structured and related to each other. For instance, a *slab* object could be related to a *building story* object linking the slab to the story where it is built. Furthermore, the slab is likely to be associated with a geometrical representation, a placement and a material property allowing the slab to be drawn at the correct position and with proper texture. A detailed description of a generic product model for buildings can be found in [59]. When stored in a database like fashion at a central location, this approach is named product model server. It then permits distributed access and version control and therewith provides up-to-date model data for all project partners. Instead of a set of files, model updates concern individual objects within the hierarchical data structure. The semantics of the model provide a possibility to perform consistency checks.

The product model server approach is the basis for the integration of structural analysis with the design process pursued in this thesis. Therefore, it will be discussed in more detail in the section to follow. The conception for the integration is explained afterwards.

2.2 Product data management in civil engineering

2.2.1 General concept

The traditional notion of a building model is the geometric representation of a building, either on paper or within a CAD environment. The overall model can be regarded as a sum of documents, the drawings stored as digital files. Data management is concerned with managing these files and their revisions, a task not only relevant to the AECO industry.

Enterprise content management (ECM) systems are the counterpart to the already named virtual project rooms. They similarly base on managing individual documents but are especially suited for the use within a specific company. An enhancement to the ECM systems are product data management (PDM) systems. In contrast to the file focus of ECM systems, PDM systems manage the information on the basis of individual product components. This approach requires a strong hierarchical structuring of the total of information into preferably small parts. Pursuing the structuring in a rigorous manner lends some semantics to the model, which can then be utilized by PDM systems for their managing tasks. At the bottom line, however, the discrete informational entities associated with the product components are stored as individual files. Though related to each other by a database or by other files, the file content itself remains a black box to the PDM system. And since a major part of the model is black-boxed data to the system, the system itself can hardly be used to ensure interoperability between applications using this data. An overview on these information management systems with application to civil engineering is provided by Günthner and Borrmann [76, sec. 3.4].

Setting up a PDM system for a specific application, i.e. a product and its development processes involves extensive configuration work. The high effort associated with the customization is an important reason for the PDM systems being primarily employed in large companies with an engineering context, e.g. in the automotive, aerospace, or other high-tech industries [1]. Also, interoperability within large companies can be ascertained by other means, for instance by prescribing the software applications to be used. Within the AECO industry with its individual building products and complex contractor structures, neither the effort of setting-up the system nor the dictation of software applications is feasible for a temporary “virtual enterprise”

[102] comprising all contactors. Nonetheless, Günthner and Borrmann [76] demonstrated the fundamental suitability of PDM systems also for construction projects.

A weakness of document based systems like virtual project rooms, ECMs and in the consequence also PDMs, with regard to document management is the concurrent work on the same document or part. Prior to any modification, the respective file must be locked and can only be unlocked after the modified file was updated within the system. Without locking, file consistency cannot be ensured as parallel edits could occur. Depending on the "size" of the informational entity and the time required for its editing, the collaboration of project partners can be severely impaired. Especially in the context of AECO projects, where the substructuring of a building down onto the level of individual building elements is not expedient for most engineering domains and where design modifications can require rather long time periods, the pessimistic concurrency control is likely to impose a problem for the overall workflow.

The sum of the geometric entities contained in a traditional building model have a meaning for human beings, as humans interpret them with regard to the building that is to be constructed. However, such a model is not interpretable by software. As in the case of PDM systems, it remains a black box. To overcome this limitation of the purely geometric models – not only within the AECO industry – product data models that include semantic information were created.

According to Fenves et al. [64], reliable product data models must filter, structure, integrate, control, and channel all the information arising during a products lifecycle in such a way, that any actor receives and manipulates only the information pertinent for their task. In the domain of AECO, such a data model is referred to as building information model. The Contractor's Guide to BIM [10] describes the instantiation of this model as "a data-rich, object-oriented, intelligent and parametric digital representation of the facility, from which views and data appropriate to various users' needs can be extracted and analyzed to generate information that can be used to make decisions and improve the process of delivering the facility". Building information modeling is the process of developing and using this model instantiation. Note that the data model and the instantiation for a specific building are equally referred to as building information model. Therefore, the domain-neutral term product data management continues to be used for the data model in this work.[1]

In the general understanding, a building information model is associated with a central repository storing the data – the product model server. And obviously, the data must contain a lot more than the geometric representations in order to meet the definition of BIM. Materials, costs, time, and suppliers are, among many others, also informational entities to be stored, all of which must be interconnected by relationships. Entities and relationships are considered objects that are instantiated from classes in the product data model. A class defines the typical attributes of the specific object type and the instantiation of predefined classes adds semantic meaning to the information. Conversely, this means, that storing information requires an appropriate class to exist in the data model. With regard to the multitude of possible object types required for the various engineering and non-engineering domains involved in the planning, construction and operation of a building, this imposes high demands on the data model.

[1] Note that in general, the terms *building information model* and *building information modeling* are very inconsistently used by different authors. Often, the abbreviation *BIM* is associated with both terms and no distinction is made for its usage.

Whereas the interoperability of the various domains benefits from the semantics of the model, data management is facilitated by the storage of information as structured objects. Users retrieve the relevant model subset from the central repository and work on their local copy. After finishing their design work, modifications can be identified in terms of new, deleted and modified objects. An approach of reintegrating only the changes in the central storage and thereby creating a new version of the entire model is presented by Weise and Katranuschkov [167]. Also Nour and Beucke [122] support object versioning as an approach of data management in a collaborative BIM environment.

2.2.2 Industry Foundation Classes

A product data model for the use within the AECO industry are the Industry Foundation Classes (IFC). They are developed by buildingSMART[2], an international organization that emerged from a former alliance of industry partners. With the current version IFC4 being standardized in ISO 16739 [86], IFC is presently the only relevant open data model for building information modeling.[3] Various commercial software companies have developed proprietary BIM solutions[4] yet the idea of interoperability is counteracted by using vendor specific data models.

Industry foundation classes are defined in EXPRESS, a standardized[5] language for the description of product data models. Like regular programming languages, EXPRESS knows some basic data types, strings, reals and integers for instance, and allows the construction of *entity* named data containers that aggregate a number of attributes. The attributes consist of basic data types or other entities. Also, inheritance, a basic concept of object-oriented programming languages, exists within EXPRESS. An entity can be declared a subtype of another entity or it can be of abstract type and thus requires subentities for instantiation. The purpose of EXPRESS is the definition of product data models as high-level schemas. For the implementation in computer applications, the schema must be translated into software. The implementation details are left to the software engineer.

Following the entity-relationship model by Chen [36], the IFC provide a generic basis for a hierarchically structured, highly attributed model of the components and processes of a building's life cycle. "Any semantically treated thing or process"[6] within the model is derived from the *IfcObjectDefinition* entity and all of these thing or process objects are interconnected by relation objects, which are a subtype of the *IfcRelationship* entity. Figure 2.1 illustrates the hierarchical structure of an IFC based building information model in a simplified manner. In this figure, relations are colored yellow and semantical objects blue or violet. Of special meaning to the architectural and structural analysis domains is the violet colored *IfcBuildingElement* entity, which is an abstract representation of all "structural and space separating systems"[7] that

[2] buildingSMART International Ltd., website: www.buildingsmart.org. The committees responsible for IFC development, the *Model Support Group* and the *Implementation Support Group* run their own website at: www.buildingsmart-tech.org. Both websites last accessed on 2015-06-15.

[3] The *CIMsteel Integration Standard* version 2 (CIS/2) is also a recognized, non-proprietary standard for the digital exchange of lifecycle data. Yet, its scope is limited to structural steelwork (cf. Reed [134]).

[4] Some examples of proprietary BIM applications are Autodesk Revit, Bentley AECOsim Building Designer, Graphisoft ARCHICAD, RIB iTWO, Nemetschek Vectorworks

[5] ISO 10303-11:2004 – Industrial automation systems and integration – Product data representation and exchange – Part 11: Description methods: The EXPRESS language reference manual

[6] Industry Foundation Classes Release 4 Specification, 5.1.3.7 IfcObjectDefinition

[7] Part of the entity definition in the IFC by buildingSMART.

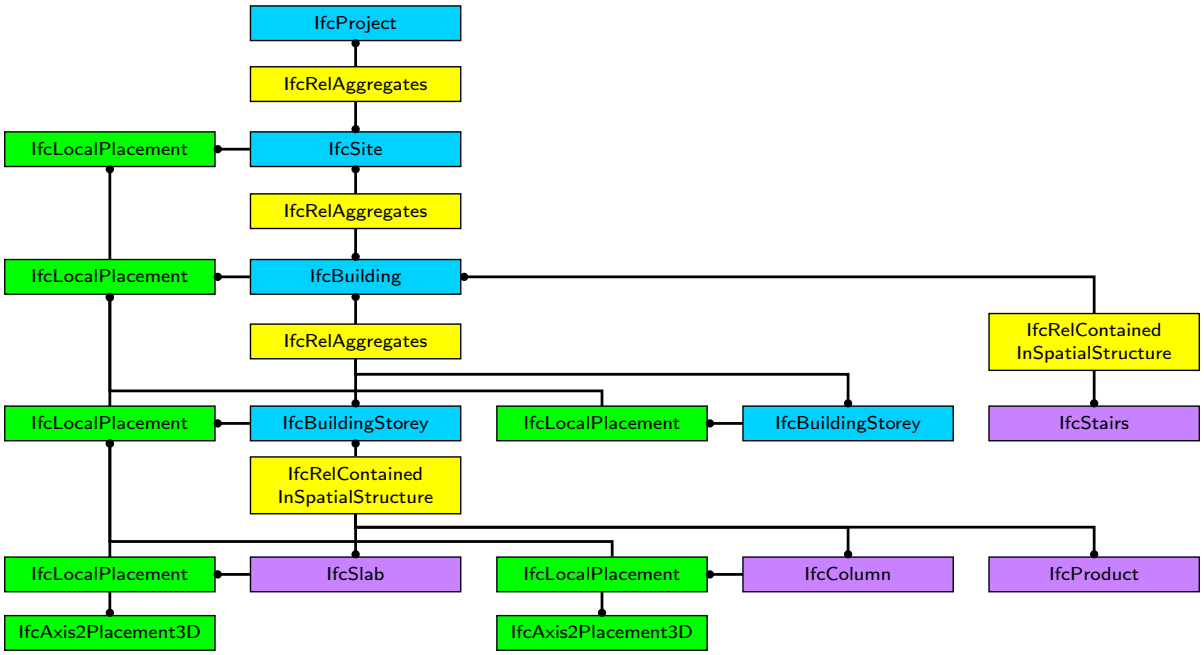


Figure 2.1: Hierarchical structure of a simplified, IFC based building information model. Root entry is the *IfcProject* entity.

make up a building. *IfcProduct* is the supertype of these building elements. Product entities are commonly represented by a shape and have a placement attribute. Resources, in the context of the IFC layer definition, enhance the available information of any *IfcProduct* subtype. Materials, placements and geometric representations are, for example, such resources (green). They cannot exist on their own but must always be linked to a semantic object. Figure 2.1 depicts the hierarchical structure of the model and the relation of the individual objects. *IfcProject* is the root of any such model and may contain several sites (*IfcSite*) which in turn may contain multiple buildings. Each building is decomposed into a defined number of building storys. The *IfcBuildingStorey* entity then aggregates the building elements of this story – a slab and a column for the given example. Building elements can also be directly related with the *IfcBuilding* entity. Navigation along the depicted relationships and the non-depicted inverse relations puts all object based information snippets into the context of the building or project respectively.

The IFC data model comprises four conceptual layers, the named resource layer, a core layer, an interoperability layer, and a domain layer. Data models for products used across different domains are included on the interoperability layer, whereas the domain layer contains specialized data models that are generally used by a specific discipline only. Currently, the domain layer comprises eight distinguished domains, the architectural and the structural analysis domains among them. Since a global building model can easily grow to be very complex and accordingly, the implementation requirements for a software utilizing the entire model are very demanding, only a subset of the entire IFC data schema must be supported by a specific software application. Such a subset is referred to as model view definition (MVD).

The exchange of IFC based model data can be accomplished with the help of a model server. At www.bimserver.org, there exists an open source implementation by Beetz et al. [23] and also

commercial services are available, the IFChub[8] and the EDMmodelServer[9] are two examples thereof. However, to date IFC based model servers are primarily subject of academic research in the context of collaborative work. Software vendors prefer their proprietary formats and regard IFC as a standardized format for the file based exchange of complete models (MVDs) only. Accordingly, a range of commercial AECO applications allows the export of proprietary models to IFC, stored in human-readable text files, but they currently do not provide an interface to IFC based model servers. The text files containing the IFC based models are either stored in STEP[10] or XML[11] format. Since files in XML format are significantly larger, the latter format is rarely used. Despite the existing standard, the practical exchange of file based BIM models is yet accompanied by many problems. The transfer of model data from one software application to another as well as the roundtrip within the same application can lead to geometric misinterpretations and model modifications, cf. [87, 128, 153] for details.

A full description of the IFC standard cannot be given within this context. A section containing an overview of IFC is included in the BIM handbook by Eastman et al. [57]. For a review of IFC development, the reader is referred to Laakso and Kiviniemi [100], and a documentation of the full standard is available online[12].

2.3 Structural analysis and product model data

The building elements *IfcSlab*, *IfcColumn* and *IfcStairs* in fig. 2.1 are part of the interoperability layer and therefore should be accessible by and meaningful for the structural analysis domain. As an example, the column and its detailed representation within the IFC data model will be subsequently examined for the interoperable use with architectural and structural analysis tasks.

A simple IFC representation of a column from an architectural point of view is depicted in fig. 2.2. There is the *IfcColumn* entity providing the semantic meaning, its placement with respect to the building story containing the column, and an associated shape representation. The actual IFC objects defining the shape are not shown. Instead, the figure contains a box with three dots that symbolize the numerous possibilities of shape representation. Basically, they comprise the wide variety of methods offered by CAD applications. Among them are constructive solid geometry (CSG), extruded solids, boundary representations (BReps), tessellated surface models, and so forth. Of course, an architect could associate further information with the column entity, but there is no requirement to do so.

[8] cf. www.ifchub.com, last accessed: 2015-06-24

[9] cf. www.epmtech.jotne.com/solutions/bim, last accessed: 2015-06-24

[10] The STEP file format (STandard for the Exchange of Product model data) is standardized in ISO 10303-21:2002 – *Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure*. The standard provides a file layout for any product model encoded with the EXPRESS language. IFC files in STEP format usually have the ".ifc" extension.

[11] Since XML is supported by a wide range of software applications, IFC based building models can also be encoded in an ".ifcXML" file. The format is standardized in ISO 10303-28:2007 – *Industrial automation systems and integration – Product data representation and exchange – Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas*.

[12] cf. www.buildingsmart-tech.org/ifc/IFC4/final/html, last accessed: 2015-06-24

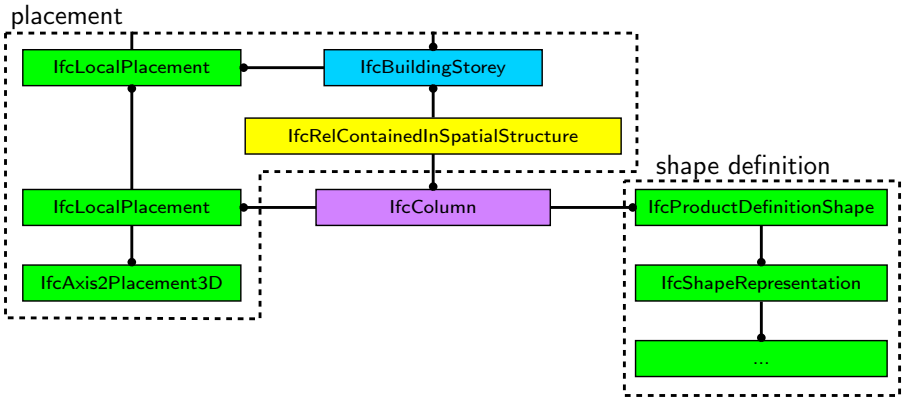


Figure 2.2: Representation of a column by the IFC product data model from an architectural point of view.

The same column expressed from a structural analysis point of view is depicted in fig. 2.3. Though the *IfcColumn* entity is part of IFC's interoperability layer, it neither appears in the figure nor is it in any way required for the column representation. Instead, there is the *IfcStructuralCurveMember*, an entity from the structural analysis domain on the domain layer. Its placement is typically given in relation to a structural analysis model object that may be related with a building object. Like the architectural column, the structural curve member requires a shape definition, which is expressed in a rather simple manner by the columns topology, a straight line or edge connecting two vertices. In addition to the topology, a cross section defined by its geometrical properties and a material law with associated parameters are required

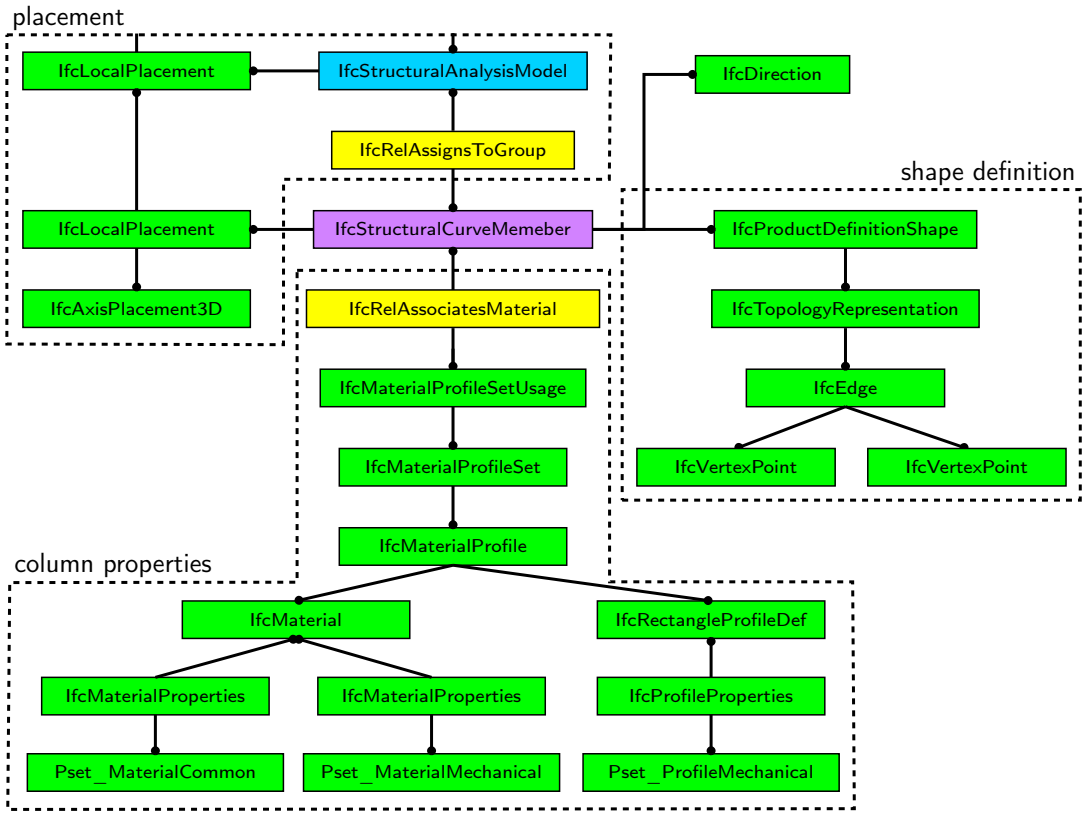


Figure 2.3: Column representation by the IFC product data model from a structural analysis point of view.

to perform a structural analysis. The respective objects are linked to the column by a relationship object. Furthermore, node objects are required to connect the column to other structural entities. However, for reasons of space and clarity, they are omitted here.

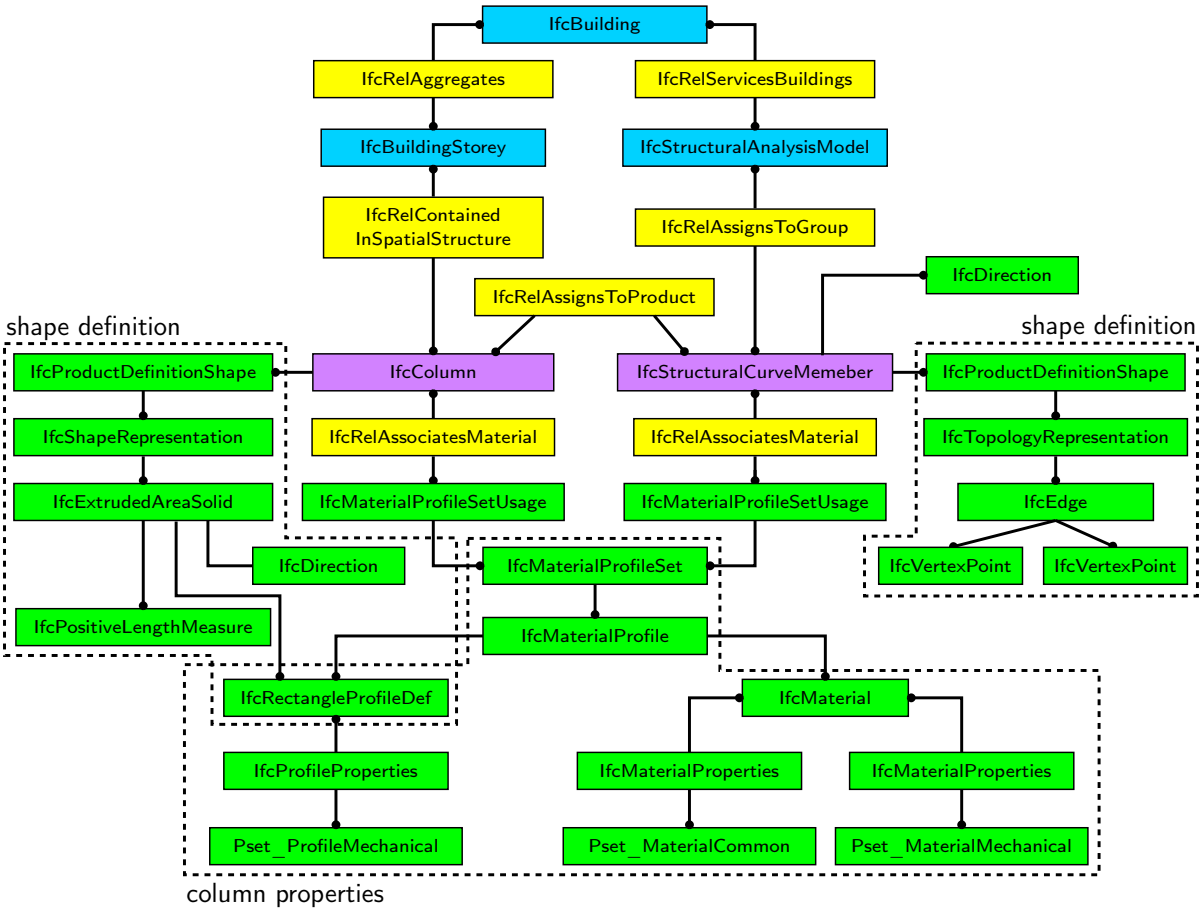


Figure 2.4: Joined column representation within the IFC product data model. For the reason of space and the purpose of clarity, the placement objects are neglected within this figure.

The architectural and the structural column representations in figs. 2.2 and 2.3 can coexist within the same building information model without any relation or association between them. Owing to the workflow of their creation, the case of a non-existent link is the most likely one. The column is created in a CAD tool and then exported to the BIM model. The structural curve member is the result of an export from a structural analysis application, certainly operated by a different person. Since there is no interdependency between the two entities, each of them can be modified irrespective of the other and thus, consistency is not guaranteed.

The IFC data model provides means to connect the two views, the result being depicted in fig. 2.4. The entities *IfcColumn* and *IfcStructuralCurveMember* can be directly linked via the objectified relationship entity *IfcRelAssignsToProduct*. Furthermore, they can share a single resource set for the definition of the material and the cross section data. And in case the architectural shape definition can be restricted to a certain type of shape representation, i.e. to an area extruded along a given path, the cross section object can be reused for shape representation. Establishing this joined representation within the BIM model requires manual work, as the various links cannot be created automatically, especially when the structural information is

added only after the architect's work is finished for the time being. Impeding the joined representation is also the current utilization of MVDs: the architect's CAD application can safely ignore the *IfcStructuralCurveMember* entity and everything else associated with it, as it stems from the "unknown" structural analysis domain. However, providing the software obstacle is overcome, some interdependency between the two model views would be achievable with the extra effort made during the creation of the combined model. For instance, a subsequent modification of the cross section would have to be done only once to be present in both views. Yet, the fundamental problem remains. There exist two shape definitions, one architectural shape and one structural topology representation that are not guaranteed to be consistent. Restraining the topology model in such a way, that it is completely dependent on the architectural shape model might be an option in the simple case of a column, but it is prohibitive for a complete structural model, the required effort is simply too high.

The example of the column is used to illustrate the problems of a comprehensive building information model. They exist in a similar manner for other building elements and are likely to increase with model size. Also, it must be considered, that there often is no one to one correspondence between building and structural elements making it even more difficult to establish relations between the two views of the model. It must be concluded, that a digital, platform-independent collaboration on the given basis of the current BIM implementation is difficult and the limited benefits might not balance the effort required for model creation.

It is to be noted, that the discussion in this section only concerns the interoperability aspect of the IFC data model with regard to architectural design and structural analysis but not the general capability of IFC to adequately represent structural analysis models. For the latter, the reader is referred to Wan et al. [164]. Here, required information on analysis data like loads, load cases, mechanical properties, analysis specifications and alike is presumed to be provided in sufficient quality by the BIM model. Since this data is unique to the domain of structural analysis, there are no direct interoperability issues.

2.4 Integrating design and analysis

Abandoning the standard approach outlined in foregoing sect. 2.3, this work pursues the combination of the architectural and structural analysis model views by thoroughly integrating the product data models of the two fields and thereby preventing any model redundancies. For the domain of structural analysis, the use of *IfcStructuralItem* entities or any subtype thereof, e.g. the *IfcStructuralCurveMember*, is relinquished in favor of the subtypes of *IfcBuildingElement* like the *IfcColumn*. Thus, the structural analysis model uses the same *IfcProduct* subtype entities as the architectural model does.

For the example of the column, the integrated approach is depicted in fig. 2.5. Though still containing all required information, the model's simplicity stands out on the figure, especially when compared with previous fig. 2.4 of a joined design and analysis representation. The advantages are obvious. Multiple property entities of identical type but with possibly contradicting content cannot be linked to a single building element entity. In theory, it is possible to add a structural topology representation to an object that is already associated with shape objects, but this must be excluded by definition. Otherwise, it would again allow redundancies in the model data. Only the original shape representation of the architectural model is used for structural analysis.

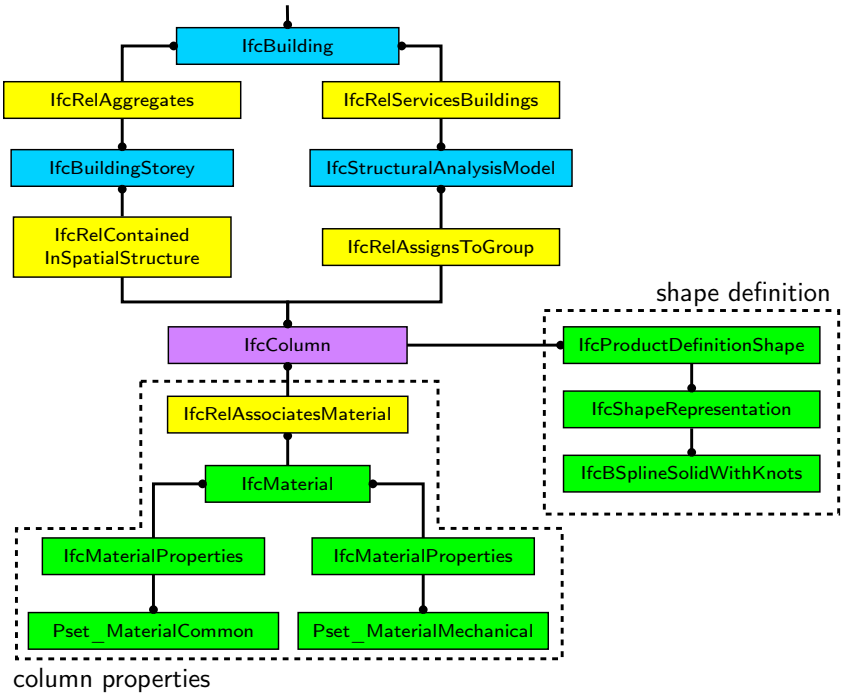


Figure 2.5: Product model of a column following an integrated approach of design and analysis. The *IfcBuildingElement* subtype *IfcColumn* is used for both, the architectural and the structural analysis model views, redundancies do not exist.

The implications for the definition of the shape representations and for the analysis process are discussed below.

The abundance of currently available geometric formulations for shape representation in the industry foundation classes must be limited to spline-based formulations, i.e. to NURBS within this work. NURBS provide a unified mathematical basis for a wide range of geometric representations in any number of spatial dimensions. Furthermore, they are suitable for analysis (cf. chapter 3) and typically, they are also supported by CAD applications. However, it is also acknowledged, that at present, they are not necessarily the standard formulation for many of the available geometric representations within these applications. Especially volumetric bodies are often represented by either CSG models[13] or BReps[14] but not by NURBS solids. Yet, software tools can be modified. And building information modeling requires a fundamental adaption of the entire modeling process in any case (cf. Eastman et al. [57]), including modifications in currently used CAD tools. With regard to the geometric formulations, these modifications can be nearly invisible for the user: Since NURBS are not only capable of representing freeform shapes but also geometric primitives, they can constitute the underlying formulation of representations that would not need such a sophisticated formulation just for the purpose of visualization. Nevertheless, the complete integration is not going to be possible without the awareness of designers that their geometric model is to be used for purposes other than visual shape representation. Plume and Mitchell [131] discuss this topic with regard to BIM

[13] CSG, or constructive solid geometry, denotes the geometric representation of volumetric bodies with solid primitives like spheres, cylinders, or cuboids. More complex shapes are built from these primitives with the help of transformations and successive Boolean operations, the CSG tree.

[14] BReps, or boundary representations, denotes the description of volumetric bodies by their boundary surfaces. Commonly, polyhedrons are used for this purpose but other surface formulations are possible as well.

and Cohen et al. [39] in the context of IGA.

A promising approach of modeling NURBS solids is outlined in the thesis of Stein [154]. And the study by Gallaher et al. [67], which quantifies the costs owed to interoperability problems to $15.8 billion alone for the U.S. capital facilities industry in the year 2002, should provide sufficient reason to implement the necessary modifications. However, only time will show whether CAD tools are eventually capable of modeling NURBS solids or any other prospective spline-based formulation as they are shortly discussed in sect. 3.5.6. For this work, it is assumed they do.

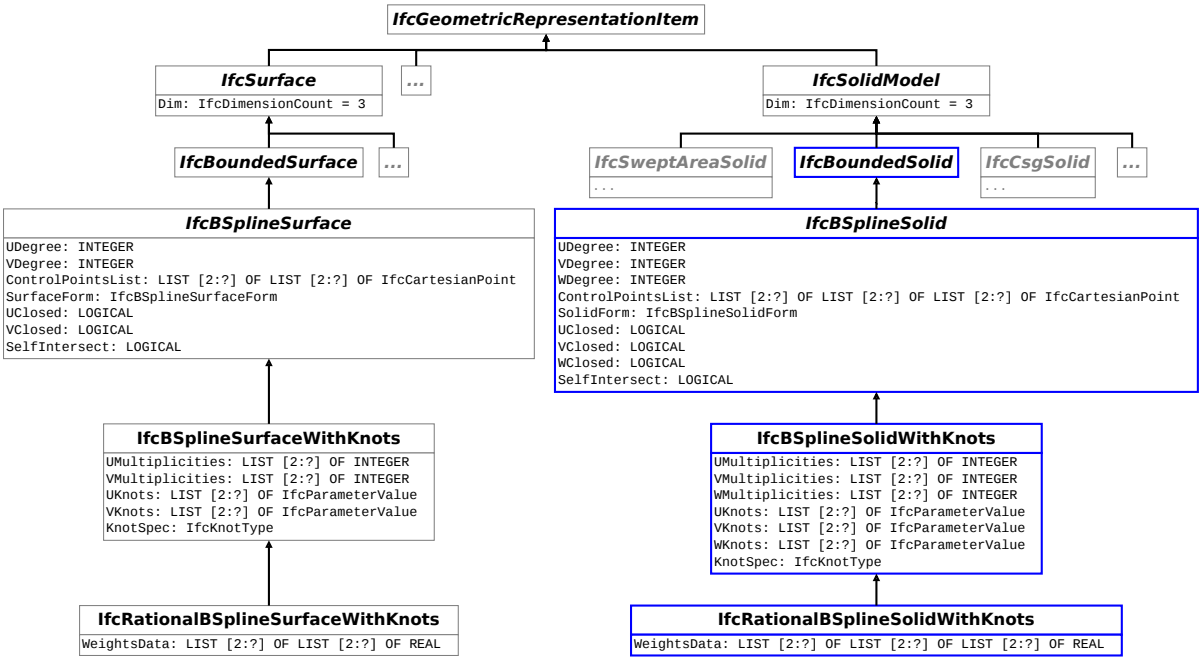


Figure 2.6: Extension of the IFC4 product data model with entities for NURBS solid representation. Existing entities are framed by grey rectangles, newly added ones by blue rectangles.

Modifications are not only required for CAD tools, but also for the IFC standard. The current version 4 was enhanced with entities for polynomial and rational, non-uniform B-spline curve and surface representations within the *IfcGeometryResource* schema on the resource layer. IFC solid representations however, which only exist in the *IfcGeometricModelResource* schema, do currently not have an expression for NURBS. A proposition for the respective extension of the standard within the existing formal structures is depicted in fig. 2.6, the corresponding EXPRESS schema is listed in appendix A.

Founding on the approach of procedurally generated models by Stein [154], it is also possible to rely on the already existing spline entity resources in IFC4 to express the shapes of building elements. This alternative to the direct storage of solid model data is depicted in fig. 2.7. The volumetric shape is expressed as a sequence of modeling operations on spline curves and surfaces. However, though all of the entities in fig. 2.7 exist, they are constrained in such a way, that the depicted combination of modeling operations and shape representation resources is not valid within the current IFC standard. Hence, also this approach would require a modification of the IFC4 schema. Another disadvantage of the procedural approach is given by Gerold [69, sec. 3.3.7]. With a building element's shape representation being only implicitly defined, it is not always possible to attach result data or other properties unambiguously to the geometry of

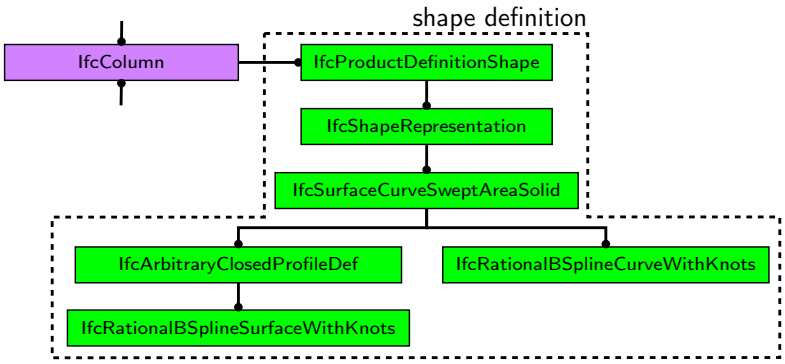


Figure 2.7: Alternative shape representation to fig. 2.5, a sequence of modeling operations defines the volumetric shape of the column.

the object. In consequence, the direct storage of NURBS solid data, for which afore mentioned limitation does not apply, is retained for the building model.

To retain all existing geometry representations of the IFC standard and to only approximate the volumetric shapes with analysis suitable formulations as a kind of post-BIM model mapping operation is an approach that was not yet pursued. Though there exist sophisticated fitting algorithms for NURBS (cf. Piegl and Tiller [129]), this option has a lot of implications. One of them is the geometric compatibility at the interfaces of the individual building elements. With shape approximation, this is unlikely to be granted, yet it is a vital requirement for the applied analysis procedure.

Using the architectural shape representations as the basis of an integrated model does, of course, not only affect CAD applications and BIM tools. Implications concern also the product model itself. Before, it was necessary to store sections and their mechanical properties as an integral part of the structural analysis model view. With volumetric models, this becomes superfluous. All the information is contained in the shape and can be mathematically recovered whenever required. Similarly, most load data does not need to be stored explicitly in the product model. Obviously, the structural weight can be calculated from the solid shape and the associated material data. But also the collateral load and in part even the live load can be automatically inferred from the building information model. Fundamental are the consequences for the structural analysis process. Basing the analysis on solid models and using spline formulations as mathematical basis are two issues that necessitate changes in the simulation procedures and their underlying algorithms. These were already outlined in sect. 1.2, key aspects will be discussed in chapters 4 to 6.

Chapter 3

Isogeometric analysis

3.1 Introduction

Upon giving the motivation for the use of isogeometric analysis in the introductory chapter 1, this chapter presents the basic equations along with an introduction to the notation. The properties and definitions explained here, will be used for reference throughout this work.

First, the general boundary value problem of linear elasticity is derived. Subsequently it is shown, how this problem can be solved with the classic finite element method. Next, spline geometry is introduced by initially discussing univariate Bézier, B-spline, and NURBS curves together with their respective underlying basis functions. Afterwards, the univariate curves are extended to multivariate formulations. With the definition of the different basis functions at hand, their incorporation in the finite element method is illustrated and the fundamental properties of the resultant isogeometric analysis concept are presented.

The information contained in the following sections serves as a basis for this work but does not provide a complete discussion on the individual topics. For this purpose, references to fundamental publications and monographs are given in the respective sections. The content provided here stems from these references but due to its basic nature, is in general not cited explicitly.

3.2 Governing equations of linear elasticity

Any material point P within a deformable, continuous body $\mathcal{B}$ is referenced by its initial position vector $\boldsymbol{x}$ in Euclidean space $\mathbb{R}^d$. The space covered by the body, i.e. its domain in global coordinates is $\Omega \subset \mathbb{R}^d$. The boundary of the body is denoted Γ. For simplicity, the body is assumed to be undeformed in its initial configuration.

Under the action of any externally applied load, the configuration of the body will change.[1] The position of a material point P is then given by $\boldsymbol{X}$. This change of the body's configuration may be expressed by the difference of the position vectors of all points P.

$$\boldsymbol{u}(P) = \boldsymbol{X}(P) - \boldsymbol{x}(P) \qquad \forall P \in \Omega \tag{3.1}$$

Equation (3.1) defines the displacement field $\boldsymbol{u}$ of the body $\mathcal{B}$ under the action of the applied load.[2] With $\boldsymbol{e}_i$ being the orthonormal standard basis of $\mathbb{R}^n$, the displacement vector $\boldsymbol{u}$ can be

[1] This is a direct consequence of Newton's 1^{st} law of motion stating that a body remains at rest (or moves with constant velocity) unless an external force acts on that body.

[2] Using the Lagrangian formulation, with x denoting the reference frame.

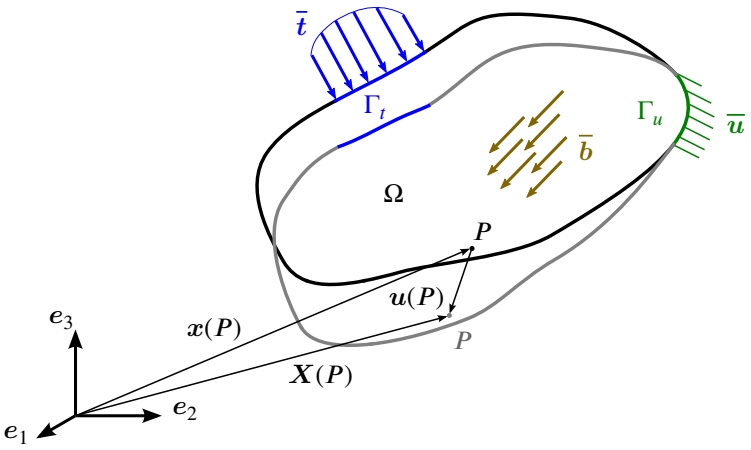


Figure 3.1: Basic continuum mechanics symbols and relations for a body under the action of externally applied loads. The undeformed configuration is symbolized by the black shape whereas the grey shape marks the deformed configuration.

expressed as the sum of the basis vectors scaled by the scalar components of $\boldsymbol{u}$.

$$\boldsymbol{u} = \sum_{i=1}^{n} u_i \boldsymbol{e}_i \tag{3.2}$$

Using the Einstein summation convention, that implies summation over repeated indices within a single term, eq. (3.2) can be shortened to

$$\boldsymbol{u} = u_i \boldsymbol{e}_i \tag{3.3}$$

This convention will be used in the remainder of this thesis, wherever omitting the summation symbol improves the readability. Just like the displacement vector, the position vectors $\boldsymbol{x}$ and $\boldsymbol{X}$ can be expressed by the sum of their components:

$$\boldsymbol{x} = x_i \boldsymbol{e}_i \qquad\qquad \boldsymbol{X} = X_i \boldsymbol{e}_i \tag{3.4}$$

A deformation of the body is caused only by relative displacements between material points. These relative displacements lead to strains and eventually to stresses within the body. The displacement gradient $\boldsymbol{H}$ defined in eq. (3.5) provides a basis for the local strain information.

$$\boldsymbol{H} = \nabla \boldsymbol{u} = u_{i,j} \boldsymbol{e}_i \otimes \boldsymbol{e}_j \tag{3.5}$$

with $u_{i,j}$ being the partial derivative of the scalar displacement vector components i with respect to the initial position vector components j

$$u_{i,j} = \frac{\partial u_i}{\partial x_j} \quad \text{for} \quad i, j = 1, 2, ..., n \tag{3.6}$$

The displacement gradient is used for the definition of the Green-Lagrange strain tensor field of the body $\mathcal{B}$:

$$\boldsymbol{E} = \frac{1}{2}\left[\boldsymbol{H} + \boldsymbol{H}^T + \boldsymbol{H}^T \cdot \boldsymbol{H}\right] \qquad \forall P \in \Omega \tag{3.7}$$

Under the assumption of small displacements and small displacement gradients, i.e. $\|\boldsymbol{E}\| \ll 1$, the higher order terms in the definition of the Green-Lagrange strain tensor can be neglected, leading to a linear strain tensor field $\boldsymbol{\epsilon}$, also called infinitesimal strain tensor. The definition of $\boldsymbol{\epsilon}$ corresponds to the symmetric part of the displacement gradient tensor.

$$\begin{aligned}\boldsymbol{\epsilon}(P) = \boldsymbol{H}^{(s)} &= \frac{1}{2}\left[\boldsymbol{H} + \boldsymbol{H}^T\right] \\ &= \frac{1}{2}\left[\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right] \qquad \text{for} \quad i,j = 1,2,...,n, \forall P \in \Omega \\ &= \frac{1}{2}\left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right]\boldsymbol{e}_i \otimes \boldsymbol{e}_j\end{aligned} \tag{3.8}$$

Equation (3.8) constitutes the kinematic equation of the problem.

The deformation of the body leads to internal forces within that body. These internal forces are expressed via stresses i.e., force per differential area element. A stress vector, or traction, $\boldsymbol{t}$ represents the internal force state of any, possibly imaginary, surface with unit outward normal vector $\boldsymbol{n}$, cutting though the body at a material point P (eq. (3.9))

$$\boldsymbol{t}(\boldsymbol{n}, P) = t_i(\boldsymbol{n}, P)\,\boldsymbol{e}_i \quad \text{with} \quad t_i(\boldsymbol{n}, P) = \frac{df_i}{dA} \qquad \text{for} \quad i,j = 1,2,...,n \tag{3.9}$$

In this equation f_i stands for the scalar components of the contact force vector $\boldsymbol{f}$ of a differential surface element with size dA and t_i is the respective scalar stress vector component for the given basis. According to Cauchy's fundamental theorem (cf. [5, p. 144]), the knowledge of the stress vectors on n orthonormal planes in $\mathbb{R}^n$ suffices to uniquely define the stress state of P and therewith the stress vector of any plane running though that point. Hence, the stress state of the body can be expressed by the Cauchy stress field, defined in eq. (3.10).

$$\boldsymbol{T}(P) = \boldsymbol{t}(\boldsymbol{e}_i, P) \otimes \boldsymbol{e}_i = T_{ij}\boldsymbol{e}_i \otimes \boldsymbol{e}_j \qquad \text{for} \quad i,j = 1,2,...,n, \forall P \in \Omega \tag{3.10}$$

Here T_{ij} represents the scalar component j of the traction vector $\boldsymbol{t}_i$. This traction vector acts on a plane normal to $\boldsymbol{e}_i$ and the component j acts in direction of $\boldsymbol{e}_j$ on that plane.

For the case of the body's material being linear elastic, there exists a linear relationship between the stress and the strain tensor. This relation given in eq. (3.11) is the constitutive equation of the problem.

$$\boldsymbol{\sigma}(P) = \boldsymbol{D}(P) : \boldsymbol{\epsilon}(P) \tag{3.11}$$

$\boldsymbol{D}$ is the linear elastic material tensor and $\boldsymbol{\sigma}$ denotes the linearized version of the stress tensor.[3] Further assuming the material to be isotropic, the scalar components of the material tensor $\boldsymbol{D}$

[3] The Cauchy stress tensor (eq. (3.10)) is formulated with reference to the current configuration of the body $\mathcal{B}$, whereas the Green-Lagrange strain tensor (eq. (3.7)) is based on the initial configuration. Yet, in the context of the linearized theory for small displacements, the various stress measures coincide. Namely, the difference between the 2^{nd} Piola-Kirchhoff stress tensor, which is formulated with reference to the initial configuration of the body, as is the Green-Lagrange strain tensor, and the Cauchy stress tensor vanishes, cf. [126, p. 127]. Therefore, the stress tensor notation is rewritten for the given context to

$$\boldsymbol{\sigma} = \sigma_{ij}\boldsymbol{e}_i \otimes \boldsymbol{e}_j \qquad \text{for} \quad i,j = 1,2,...,n, \forall P \in \Omega \tag{3.12}$$

can be expressed in terms of the two Lamé constants λ and μ characterizing the isotropic linear elastic material.

$$D_{ijkl} = \lambda\delta_{ij}\delta_{kl} + \mu\left(\delta_{ik}\delta_{kl} + \delta_{il}\delta_{jk}\right) \quad \text{with} \quad \delta_{ij} = \begin{cases} 1 & \text{for} \quad i = j \\ 0 & \text{otherwise} \end{cases} \tag{3.13}$$

The two Lamé constants can be deduced from the more commonly used elastic constants Young's modulus E and poisson ratio ν.

$$\lambda = \frac{\nu E}{(1+\nu)(1-2\nu)} \qquad \mu = \frac{E}{2(1+\nu)} \tag{3.14}$$

The body load (density) $\boldsymbol{b}$ at any material point $\boldsymbol{P}$ is defined as a function of the body's material mass density ρ and an external field $\boldsymbol{g}$, the latter, e.g. a gravitational field, being the source of the force acting on the body.

$$\boldsymbol{b}(\boldsymbol{P}) = -\rho(\boldsymbol{P})\,\boldsymbol{g}(\boldsymbol{P}) \tag{3.15}$$

Following Newton's third law of motion, static equilibrium between internal and external forces can then be expressed through eq. (3.16), in which the bar on the body load symbol $\boldsymbol{b}$ highlights the external nature of the body load.

$$\nabla \cdot \boldsymbol{\sigma}(\boldsymbol{P}) + \overline{\boldsymbol{b}}(\boldsymbol{P}) = \boldsymbol{0} \qquad \forall \boldsymbol{P} \in \Omega \tag{3.16}$$

The partial differential equations (3.8) and (3.16) together with the constitutive relation given in eq. (3.11) form the governing equations of the linear elastic boundary value problem (BVP). Solving such a BVP for the unknown displacement field $\boldsymbol{u}$ of a given body in a known load state requires the additional formulation of boundary conditions. These can be given as mixed boundary conditions with imposed displacements $\overline{\boldsymbol{u}}$ on the boundary Γ_u, named Dirichlet or essential boundary conditions, and externally applied tractions $\overline{\boldsymbol{t}}$ on the boundary Γ_t, referred to as Neumann or natural boundary conditions. Γ_u and Γ_t are disjoint but fulfill the condition $\Gamma_u \cup \Gamma_t = \Gamma$. Mathematically, these conditions are stated in eqs. (3.17) and (3.18).

$$\boldsymbol{u} = \overline{\boldsymbol{u}} \qquad \forall \boldsymbol{P} \in \Gamma_u \tag{3.17}$$

$$\boldsymbol{n} \cdot \boldsymbol{\sigma} = \overline{\boldsymbol{t}} \qquad \forall \boldsymbol{P} \in \Gamma_t \tag{3.18}$$

A closed form analytical solution for the formulated boundary value problem is only known to be possible for cases with rather simple geometry and loading conditions. Examples of such problems will be given later on in this work, when their analytical solutions will be used as a reference for comparison with results of respective numerical analyses.

For a more thorough treatment of the topic the reader is referred to one of numerous textbooks available. Linear elasticity is especially treated in Maceri [110] and Sadd [142], whereas books focusing on continuum mechanics provide a broader view, see e.g. Altenbach [5], Lai et al. [101] or Mase and Mase [112].

3.3 Finite element method

3.3.1 Introduction

As already stated before, it is not possible to solve the BVP outlined in sect. 3.2 for a general problem setting with analytical methods. Instead, numerical methods in combination with today's computer technology are employed. Probably the most widely used technique for solving large engineering problems is, as of its generality, the method of finite elements. A thorough insight into this method can be gained by reading the books of Bathe [17], Hughes [82], or Zienkiewicz et al. [174]. This section here shall be used to briefly introduce the displacement based version of the method, in which the primary unknown is solely the displacement field of the body.

The basis of the finite element method is a reformulation of the BVP given in eqs. (3.8), (3.11), and (3.16) to (3.18).

$$\int_{\Omega} \boldsymbol{\epsilon}(\delta\boldsymbol{u}) : \boldsymbol{\sigma}(\boldsymbol{u})\,\mathrm{d}\Omega = \int_{\Omega} \delta\boldsymbol{u}\cdot\overline{\boldsymbol{b}}\,\mathrm{d}\Omega + \int_{\Gamma_t} \delta\boldsymbol{u}\cdot\overline{\boldsymbol{t}}\,\mathrm{d}\Gamma_t \tag{3.19}$$

$$\boldsymbol{u} = \overline{\boldsymbol{u}} \qquad \forall \boldsymbol{P} \in \Gamma_u \tag{3.20}$$

$$\delta\boldsymbol{u} = \boldsymbol{0} \qquad \forall \boldsymbol{P} \in \Gamma_u \tag{3.21}$$

Equations (3.19) to (3.21) are known as the principle of virtual displacements, the variational formulation of the problem. The δ denotes a variation of the unknown field variable $\boldsymbol{u}$ that may in general be arbitrary but must be sufficiently smooth to evaluate $\boldsymbol{\epsilon}$ on Ω and it must be zero on the Dirichlet boundary. The left term of the equation represents the internal virtual work of the body, which must be in equilibrium with the external virtual work, expressed by the term on the right of the equality sign. The equivalence of this formulation with the one given in sect. 3.2 is shown in Bathe [17, pp. 156-158].

For the ease of presentation, the space of Ω is subsequently restricted to $\mathbb{R}^3$ and the basis is defined to be of Cartesian type. This allows to express the relevant tensors of the governing equations by the respective vectors or matrices of their scalar components. In particular, position and displacement tensors can be rewritten as

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \qquad \boldsymbol{X} = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \qquad \boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{3.22}$$

The components of all other spatial vectors, as e.g. $\boldsymbol{b}$, $\boldsymbol{n}$, and $\boldsymbol{t}$ have, without change of meaning, either lower indices 1, 2, and 3 or x, y and z.

The infinitesimal strain and stress tensors read as

$$\boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_{11} & \epsilon_{12} & \epsilon_{13} \\ \epsilon_{21} & \epsilon_{22} & \epsilon_{23} \\ \epsilon_{31} & \epsilon_{32} & \epsilon_{33} \end{bmatrix} = \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{yx} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{zx} & \epsilon_{zy} & \epsilon_{zz} \end{bmatrix} \tag{3.23}$$

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} = \begin{bmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz} \end{bmatrix} \tag{3.24}$$

which can, due to the symmetry of the tensors, also be expressed as vectors.

$$\underline{\boldsymbol{\epsilon}} = \begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ 2\epsilon_{23} \\ 2\epsilon_{13} \\ 2\epsilon_{12} \end{bmatrix} = \begin{bmatrix} \epsilon_{11} \\ \epsilon_{22} \\ \epsilon_{33} \\ \gamma_{23} \\ \gamma_{13} \\ \gamma_{12} \end{bmatrix} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \epsilon_4 \\ \epsilon_5 \\ \epsilon_6 \end{bmatrix} = \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ 2\epsilon_{yz} \\ 2\epsilon_{xz} \\ 2\epsilon_{xy} \end{bmatrix} = \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \epsilon_{zz} \\ \gamma_{yz} \\ \gamma_{xz} \\ \gamma_{xy} \end{bmatrix} \tag{3.25}$$

$$\underline{\boldsymbol{\sigma}} = \begin{bmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \sigma_{23} \\ \sigma_{13} \\ \sigma_{12} \end{bmatrix} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{bmatrix} = \begin{bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \sigma_{zz} \\ \sigma_{yz} \\ \sigma_{xz} \\ \sigma_{xy} \end{bmatrix} \tag{3.26}$$

The vector representation of the strain and stress tensors is known as Voigt notation, which is marked by the underline symbol above.

Utilizing its symmetry, also the material tensor of isotropic, linear elastic materials can be rewritten as a 6×6 matrix. The matrix complies with the Voigt notation of the strain and stress tensors.

$$\underline{\boldsymbol{D}} = \begin{bmatrix} 2\mu + \lambda & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & 2\mu + \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & 2\mu + \lambda & 0 & 0 & 0 \\ 0 & 0 & 0 & \mu & 0 & 0 \\ 0 & 0 & 0 & 0 & \mu & 0 \\ 0 & 0 & 0 & 0 & 0 & \mu \end{bmatrix} \tag{3.27}$$

And finally, a differential operator matrix $\boldsymbol{\mathcal{L}}$ can be defined (eq. (3.28)), that allows the expression of eqs. (3.8) and (3.16) in matrix and vector notation.

$$\boldsymbol{\mathcal{L}} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 & 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ 0 & \frac{\partial}{\partial y} & 0 & \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ 0 & 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \end{bmatrix}^T \tag{3.28}$$

3.3.2 Discretization

In the finite element method, the domain Ω of the continuous body $\mathcal{B}$ is partitioned into non-overlapping subdomains Ω_e, each of them denoted a finite element. As the shape of the subdomains is generally restricted, the union of all subdomains is only an approximation of the original domain (eq. (3.29)).

$$\Omega \approx \Omega^h = \bigcup \Omega_e \tag{3.29}$$

$$\Omega_{e_i} \cap \Omega_{e_j} = \emptyset \qquad \forall \Omega_{e_i}, \Omega_{e_j} \in \Omega^h \quad \text{with} \quad \Omega_{e_i} \neq \Omega_{e_j} \tag{3.30}$$

The boundary of a finite element Ω_e is denoted Γ_e. $\Gamma_{e,t}$ and $\Gamma_{e,u}$ represent those parts of the element boundary that coincide with the body's Neumann and Dirichlet boundaries respectively.

$$\Gamma_{e,t} = \Gamma_e \cap \Gamma_t^h \qquad \text{and} \qquad \Gamma_{e,u} = \Gamma_e \cap \Gamma_u^h \tag{3.31}$$

A specific number n_{en} of nodal points $\hat{P}$, with $\hat{P} \subset P \in \Omega_e$, is defined for each element e, with neighboring elements sharing the nodal points on their common boundary.

With the definition of an appropriate set of n_{en} functions N, the displacement field $\boldsymbol{u}$ within each element e can be expressed as a linear combination of these functions with the displacement values $\hat{\boldsymbol{u}}$ at the element's nodal points.

$$\boldsymbol{u}(P) \approx \boldsymbol{u}^h(P) = \sum_{a=1}^{n_{en}} N^{(a)}(P)\ \hat{\boldsymbol{u}}^{(a)} \qquad \forall P \in \Omega_e \tag{3.32}$$

with

$$\mathscr{U}^h = \operatorname{span}\left\{ N^{(i)}(P) \right\}_{i=1}^{n_{en}} \tag{3.33}$$

denoting the finite dimensional function space of $\boldsymbol{u}^h$, i.e. $\boldsymbol{u}^h \in \mathscr{U}^h$.

Clearly, the element's displacement field $\boldsymbol{u}^h$ expressed by eq. (3.32) can only be an approximation to the true displacement field $\boldsymbol{u}$, its quality strongly depending on the function space $\mathscr{U}^h$.

Storing all n_{en} functions N of an element in a matrix $\boldsymbol{N}$ and denoting the vector containing the displacements of all nodal points of that element $\tilde{\boldsymbol{u}}^e$, the notation of eq. (3.32) simplifies to

$$\boldsymbol{u}^h = \boldsymbol{N}\tilde{\boldsymbol{u}}^e \quad \text{with} \tag{3.34}$$

$$\boldsymbol{N} = \begin{bmatrix} N^{(1)} & 0 & 0 & N^{(2)} & 0 & 0 & \dots & N^{(n_{en})} & 0 & 0 \\ 0 & N^{(1)} & 0 & 0 & N^{(2)} & 0 & \dots & 0 & N^{(n_{en})} & 0 \\ 0 & 0 & N^{(1)} & 0 & 0 & N^{(2)} & \dots & 0 & 0 & N^{(n_{en})} \end{bmatrix} \tag{3.35}$$

and

$$\tilde{\boldsymbol{u}}^e = \begin{bmatrix} \hat{u}^{(1)} & \hat{v}^{(1)} & \hat{w}^{(1)} & \hat{u}^{(2)} & \hat{v}^{(2)} & \hat{w}^{(2)} & \dots & \hat{u}^{(n_{en})} & \hat{v}^{(n_{en})} & \hat{w}^{(n_{en})} \end{bmatrix}^T . \tag{3.36}$$

In the same manner, the geometry of the elements themselves can be expressed with the help of the interpolating functions N and the geometry, i.e. position vectors, at discrete nodal points.

$$\boldsymbol{x}(P) \approx \boldsymbol{x}^h(P) = \sum_{a=1}^{n_{en}} N^{(a)}(P)\, \hat{\boldsymbol{x}}^{(a)} = \boldsymbol{N}\tilde{\boldsymbol{x}}^e \qquad \forall P \in \Omega_e \tag{3.37}$$

The geometry mapping in eq. (3.37) explains the shape limitation of the finite elements noted in the context of eq. (3.29). Again, the limitation depends on the functions N and the number of nodal points. Using the identical set of functions for N in eqs. (3.32) and (3.37) is known as the isoparametric concept; the functions N are generally referred to as shape functions, basis

functions, or interpolation functions. In the context of the classic finite element method, the term shape function will be used subsequently.

As the nodal points $\hat{P}$ are a subset of all points P in the element domain Ω_e, the mapping in eqs. (3.32) and (3.37) at any given nodal point should result in the original value for that point. Therefore, the shape functions must fulfill the restrictions formulated in eqs. (3.38), which state that each nodal point is associated with one shape function that evaluates to one at this node. All other functions have to evaluate to zero. The associated function is interpolatory at its node.

$$\begin{aligned} N^{(a)}(\hat{P}^{(a)}) &= 1 \qquad \text{with} \quad a = 1, 2, ..., n_{en} \\ N^{(a)}(\hat{P}^{(b)}) &= 0 \qquad \text{with} \quad a, b = 1, 2, ..., n_{en} \quad \text{and} \quad a \neq b \end{aligned} \tag{3.38}$$

Rewriting the kinematic equation (3.8) in matrix notation for the $\mathbb{R}^3$ Cartesian space, results in

$$\underline{\boldsymbol{\epsilon}} = \boldsymbol{\mathcal{L}}\boldsymbol{u}. \tag{3.39}$$

The strain field can then be approximated by replacing $\boldsymbol{u}$ with $\boldsymbol{u}^h$

$$\underline{\boldsymbol{\epsilon}} \approx \underline{\boldsymbol{\epsilon}}^h = \boldsymbol{\mathcal{L}}\boldsymbol{u}^h, \tag{3.40}$$

and for a single element be expressed in terms of the nodal point displacements by substitution of $\boldsymbol{u}^h$ with eq. (3.34)

$$\underline{\boldsymbol{\epsilon}}^h(P) = \boldsymbol{\mathcal{L}}\boldsymbol{N}\tilde{\boldsymbol{u}}^e = \boldsymbol{B}\tilde{\boldsymbol{u}}^e \qquad \forall P \in \Omega_e, \tag{3.41}$$

where the application of the differential operator matrix $\boldsymbol{\mathcal{L}}$ on the matrix of element shape functions $\boldsymbol{N}$ defines the strain displacement matrix $\boldsymbol{B}$. It is to be observed, that all shape functions in $\boldsymbol{N}$ must be evaluated at the element point of interest.

Therewith, also an approximate field of element stresses can be expressed in terms of nodal point displacements. Using the constitutive relation of eq. (3.11), rewriting it in matrix notation and substituting the strain field with eq. (3.41) leads to

$$\underline{\boldsymbol{\sigma}}(P) \approx \underline{\boldsymbol{\sigma}}^h(P) = \underline{\boldsymbol{D}}\boldsymbol{B}\tilde{\boldsymbol{u}}^e \qquad \forall P \in \Omega_e. \tag{3.42}$$

Returning to the variational form of the boundary value problem of eq. (3.19), i.e.

$$\int_\Omega \underline{\boldsymbol{\epsilon}}(\delta\boldsymbol{u})^T\, \underline{\boldsymbol{\sigma}}(\boldsymbol{u}) d\Omega = \int_\Omega \delta\boldsymbol{u}^T\, \overline{\boldsymbol{b}} d\Omega + \int_{\Gamma_t} \delta\boldsymbol{u}^T\, \overline{\boldsymbol{t}} d\Gamma_t,$$

it is easy to see, that the continuous stress field $\boldsymbol{\sigma}$ can be replaced with its discretized approximation defined in eq. (3.42). By further applying the Galerkin method, the functions N previously used for the interpolation of the discretized displacement field $\boldsymbol{u}^h$ are also used to interpolate the variational field. Thus,

$$\delta\boldsymbol{u} \approx \delta\boldsymbol{u}^h = \boldsymbol{N}\delta\tilde{\boldsymbol{u}}^e \qquad \forall P \in \Omega_e \tag{3.43}$$

which expresses the same as the restriction of the virtual displacement field's function space $\mathscr{V}^h$ to that of the discretized displacement field, i.e.

$$\delta \boldsymbol{u}^h \in \mathscr{V}^h = \mathscr{U}^h \tag{3.44}$$

In consequence, the discretized form of the variational problem formulation can be written as

$$\bigcup_{\Omega_e \in \Omega^h} \int_{\Omega_e} \boldsymbol{B}^T \delta \tilde{\boldsymbol{u}}^e \underline{\boldsymbol{D}} \boldsymbol{B} \tilde{\boldsymbol{u}}^e \, \mathrm{d}\Omega_e = \bigcup_{\Omega_e \in \Omega^h} \int_{\Omega_e} (\boldsymbol{N} \delta \tilde{\boldsymbol{u}}^e)^T \, \bar{\boldsymbol{b}} \, \mathrm{d}\Omega_e + \bigcup_{\Gamma_{e,t} \in \Gamma_t^h} \int_{\Gamma_{e,t}} (\boldsymbol{N} \delta \tilde{\boldsymbol{u}}^e)^T \, \bar{\boldsymbol{t}} \, \mathrm{d}\Gamma_{e,t} \tag{3.45}$$

which, after reordering the terms in the equation and further assuming all element matrices were assembled and correctly arranged in global matrices, finally results in

$$\delta \tilde{\boldsymbol{u}} \int_{\Omega^h} \boldsymbol{B}^T \underline{\boldsymbol{D}} \boldsymbol{B} \tilde{\boldsymbol{u}} \, \mathrm{d}\Omega^h = \delta \tilde{\boldsymbol{u}} \int_{\Omega^h} \boldsymbol{N} \bar{\boldsymbol{b}} \, \mathrm{d}\Omega^h + \delta \tilde{\boldsymbol{u}} \int_{\Gamma_t^h} \boldsymbol{N} \bar{\boldsymbol{t}} \, \mathrm{d}\Gamma_t^h. \tag{3.46}$$

And since the contents of the global vector of virtual nodal point displacements $\delta \tilde{\boldsymbol{u}}$ are arbitrary, eq. (3.46) only holds, when

$$\boldsymbol{K} \tilde{\boldsymbol{u}} = \tilde{\boldsymbol{f}} \tag{3.47}$$

is fulfilled. This equation states to global problem that needs to be solved in order to find the unknown displacement field $\boldsymbol{u}^h$, expressed in terms of nodal point displacements $\tilde{\boldsymbol{u}}$. The known components of this equation are the global stiffness matrix $\boldsymbol{K}$ and the global vector of equivalent nodal forces $\tilde{\boldsymbol{f}}$. Each of the components is the sum of their respective element contributions:

$$\boldsymbol{K} = \sum_e \boldsymbol{K}^e \qquad \tilde{\boldsymbol{u}} = \sum_e \tilde{\boldsymbol{u}}^e \qquad \tilde{\boldsymbol{f}} = \sum_e \tilde{\boldsymbol{f}}^e, \tag{3.48}$$

with $\boldsymbol{K}^e$ being the element stiffness matrix

$$\boldsymbol{K}^e = \int_{\Omega_e} \boldsymbol{B}^T \underline{\boldsymbol{D}} \boldsymbol{B} \, \mathrm{d}\Omega_e, \tag{3.49}$$

and $\tilde{\boldsymbol{f}}^e$ the element vector of equivalent nodal forces

$$\tilde{\boldsymbol{f}}^e = \int_{\Omega_e} \boldsymbol{N}^T \bar{\boldsymbol{b}} \, \mathrm{d}\Omega_e + \int_{\Gamma_{t,e}} \boldsymbol{N}^T \bar{\boldsymbol{t}} \, \mathrm{d}\Gamma_{t,e}. \tag{3.50}$$

Equations (3.48) imply that the size of the element matrices is adjusted to match the total number of nodal points with entries corresponding to nodal points not belonging to the considered element being zero.

As eq. (3.47) is an integral formulation, it is to be observed, that the static equilibrium equation (3.16) is not satisfied for every point of the discretized domain Ω^h exactly, but rather in an integral or weak form over the element domain Ω_e, over which the integration is performed (see eqs. (3.49) and (3.50)). Additionally, the static equilibrium is also satisfied at all nodal points, which is the essence of eq. (3.47).

3.3.3 Isoparametric continuum element formulation

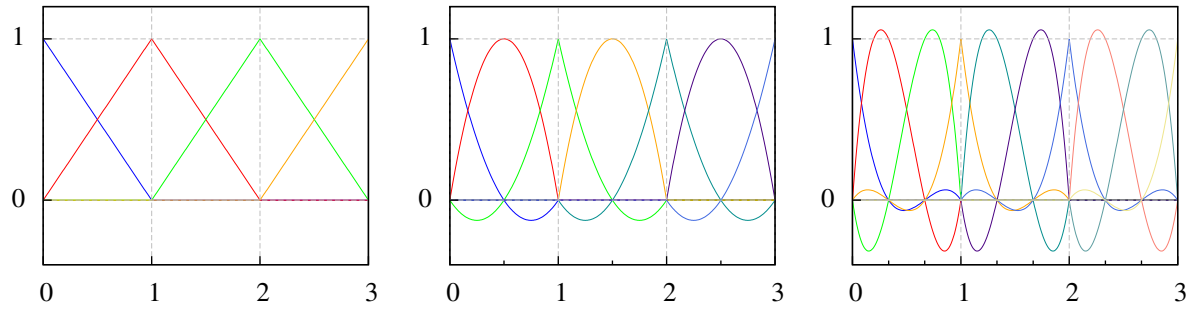
The practical evaluation of the element stiffness matrix as defined in eq. (3.49) depends on the element formulation used. For the purpose of a subsequent comparison with the isogeometric formulation, some aspects of the evaluation of a displacement based, quadrilateral element shall be discussed here.

Natural coordinates

As was shown in sect. 3.3.2, shape functions are utilized for expressing the element displacement field and the element geometry in terms of nodal point displacements and coordinates respectively. In order to derive suitable functions for this purpose, they are defined in a Cartesian coordinate system in which the element extends over the domain of $\tilde{\Omega}_e = [-1, 1]^n$. This is referred to as the element's natural coordinate system with the set of natural coordinates denoted $\boldsymbol{r}$, that is $\begin{bmatrix} r & s & t \end{bmatrix}^T$ in $\mathbb{R}^3$. With this premise, all elements of a kind have identical shape functions, regardless of their mapped geometry or position in the global coordinate space.

Shape functions in the natural coordinate system

A basic property of the shape functions was already given in eq. (3.38), they must be interpolatory at their associated nodes. The polynomials have to be constructed for the desired layout and number of nodes in an element such, that they respect this restriction. Clearly, a larger number of element nodes allows for a higher polynomial degree of the shape functions and thus for better field interpolations. However, it also increases the required computational work in terms of basis function evaluations and, more relevant, due to an increased number of unknowns, a larger global problem to solve (cf. eq. (3.47)).



(a) linear **(b)** quadratic **(c)** cubic

Figure 3.2: Lagrangian polynomials of different order as shape functions for one dimensional elements. Functions of each polynomial order are plotted over three adjacent elements, each element having unit length. Intermediate node locations are marked by minor tics on the x-axis.

Commonly used shape functions, i.e. Lagrangian polynomials of different degree, are plotted for one dimensional elements in fig. 3.2. Figure 3.2(a) depicts three elements with linear shape functions, where each element has two nodes and the neighboring elements share the connecting node. The functions have C^1 continuity over the element domain but are C^0 on the element

boundaries. It is obvious that the functions sum to unity at any point in the element domain. Also clearly visible, they fulfill the restriction given in eq. (3.38) – for each function there is a node, where the function value is unity and all other functions are zero. The elements in fig. 3.2(b) are of quadratic type, as the polynomial degree of their shape functions is two. Therefore, three nodes are required per element and the continuity within each element increases to C^2. However, on the element boundaries it remains at C^0. This does not change for the cubic elements depicted in fig. 3.2(c) with the element continuity being C^3 and the boundary remaining at C^0. Not quite as obvious, but also guaranteed for the quadratic and cubic elements, is the summation of all functions at any given point to be one. This property, expressed in eq. (3.51), is named *partition of unity*. Another thing to note is the locality of the shape functions. They assume nonzero values only in those elements that are defined through their associated nodes. This characteristic leads to a sparse global stiffness matrix and therefore reduces the required computational effort to solve the global problem.

$$\sum_{a=1}^{n_{en}} N^{(a)}(\boldsymbol{r}) = 1 \qquad \forall \boldsymbol{r} \in \tilde{\Omega}_e \tag{3.51}$$

As an example, the shape functions of the one dimensional quadratic element depicted in fig. 3.2(b) are given in eq. (3.52). Note that node 3 is the center node. Also note that separate function definitions are required, depending on the node location within the element. Shape function derivatives are available by differentiating these functions with respect to their natural coordinate r.

$$\begin{aligned} N^{(1)}(r) &= \frac{1}{2}\left(r^2 - r\right) \\ N^{(2)}(r) &= \frac{1}{2}\left(r^2 + r\right) \qquad \forall r \in \tilde{\Omega}_e \\ N^{(3)}(r) &= \left(1 - r^2\right) \end{aligned} \tag{3.52}$$

Defining the shape functions in the natural coordinate system of the element leads to a definition of the element geometry in terms of the natural coordinates. The mapping $\boldsymbol{x}^h : \tilde{\Omega}_e \rightarrow \Omega_e$ of element coordinates from the natural to the global coordinate system is given as

$$\boldsymbol{x}^h(\boldsymbol{r}) = \sum_{a=1}^{n_{en}} N^{(a)}(\boldsymbol{r})\, \hat{\boldsymbol{x}}^{(a)} \qquad \forall \boldsymbol{r} \in \tilde{\Omega}_e \tag{3.53}$$

and likewise the displacement field is expressed as

$$\boldsymbol{u}^h(\boldsymbol{r}) = \sum_{a=1}^{n_{en}} N^{(a)}(\boldsymbol{r})\, \hat{\boldsymbol{u}}^{(a)} \qquad \forall \boldsymbol{r} \in \tilde{\Omega}_e. \tag{3.54}$$

Transformation I

The strain displacement matrix defined in eq. (3.41) requires the first derivatives of the shape functions with respect to global coordinates. As the functions are defined in natural coordinates, a transformation is necessary to express the global derivatives with respect to the natural coordinate derivatives. Since the mapping (3.53) denotes that $\boldsymbol{x}$ is a function of $\boldsymbol{r}$, and also

the inverse relation exists, the chain rule can be applied to express the derivative of the shape functions $N^{(a)}$ with respect to the natural coordinate r in Cartesian $\mathbb{R}^3$ space as

$$\frac{\partial N^{(a)}}{\partial r} = \frac{\partial N^{(a)}}{\partial x}\frac{\partial x}{\partial r} + \frac{\partial N^{(a)}}{\partial y}\frac{\partial y}{\partial r} + \frac{\partial N^{(a)}}{\partial z}\frac{\partial z}{\partial r}. \tag{3.55}$$

Doing this for all three coordinates and writing in matrix notation, the following relation is obtained

$$\begin{bmatrix} \frac{\partial N^{(a)}}{\partial r} \\ \frac{\partial N^{(a)}}{\partial s} \\ \frac{\partial N^{(a)}}{\partial t} \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} & \frac{\partial z}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} & \frac{\partial z}{\partial s} \\ \frac{\partial x}{\partial t} & \frac{\partial y}{\partial t} & \frac{\partial z}{\partial t} \end{bmatrix} \begin{bmatrix} \frac{\partial N^{(a)}}{\partial x} \\ \frac{\partial N^{(a)}}{\partial y} \\ \frac{\partial N^{(a)}}{\partial z} \end{bmatrix} = \boldsymbol{J} \begin{bmatrix} \frac{\partial N^{(a)}}{\partial x} \\ \frac{\partial N^{(a)}}{\partial y} \\ \frac{\partial N^{(a)}}{\partial z} \end{bmatrix}, \tag{3.56}$$

with $\boldsymbol{J}$ being the Jacobian matrix. Its contents are readily available from the differentiation of eq. (3.53) with respect to local coordinates. After building the inverse of the Jacobian matrix, there exists an expression that relates the shape function derivatives with respect to global coordinates to the known derivatives with respect to the natural coordinates.

$$\begin{bmatrix} \frac{\partial N^{(a)}}{\partial x} \\ \frac{\partial N^{(a)}}{\partial y} \\ \frac{\partial N^{(a)}}{\partial z} \end{bmatrix} = \boldsymbol{J}^{-1} \begin{bmatrix} \frac{\partial N^{(a)}}{\partial r} \\ \frac{\partial N^{(a)}}{\partial s} \\ \frac{\partial N^{(a)}}{\partial t} \end{bmatrix} \tag{3.57}$$

With the relation above, the strain-displacement matrix $\boldsymbol{B}$ can be constructed as a function of natural coordinates but the contained shape function derivatives constitute an expression that is equivalent to the differentiation with respect to global coordinates.

Transformation II

For the integration of the element matrices and vectors, which is of the general form

$$\int_{\Omega^e} \boldsymbol{G}(\boldsymbol{P})\,\mathrm{d}\Omega^e, \tag{3.58}$$

another transformation is required. The integration domain is the global coordinate space, but $\boldsymbol{G}$ is formulated as a function of the natural coordinates. Therefore, the differential element size in natural coordinates must be related to the actual size in global coordinates. By means of vector algebra it can be shown that

$$\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z = \det \boldsymbol{J}\,\mathrm{d}r\,\mathrm{d}s\,\mathrm{d}t, \tag{3.59}$$

and thus,

$$\int_{\Omega^e} \boldsymbol{G}(P)\,\mathrm{d}\Omega^e = \iiint_{\boldsymbol{x}} \boldsymbol{G}(x,y,z)\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z = \iiint_{\boldsymbol{r}} \boldsymbol{G}(r,s,t)\det \boldsymbol{J}(r,s,t)\,\mathrm{d}r\,\mathrm{d}s\,\mathrm{d}t. \tag{3.60}$$

Replacing $\boldsymbol{G}$ with the appropriate terms of the element stiffness matrix in eq. (3.49), the stiffness matrix can be rewritten in terms of natural coordinates as

$$\boldsymbol{K}^e = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \boldsymbol{B}^T(\boldsymbol{r})\,\underline{\boldsymbol{D}}(\boldsymbol{r})\,\boldsymbol{B}(\boldsymbol{r})\det \boldsymbol{J}(\boldsymbol{r})\,\mathrm{d}r\,\mathrm{d}s\,\mathrm{d}t. \tag{3.61}$$

Numerical integration

The analytical integration of eq. (3.61) is not feasible, but in a few cases of simple element geometry. The standard procedure is to apply numerical integration instead. This assumes that the integral can be approximated by the sum of weighted function values at a number of sampling points n_{ip}.

$$\int_a^b \boldsymbol{G}(\boldsymbol{r})\,\mathrm{d}\boldsymbol{r} = \sum_{i=1}^{n_{ip}} w_i \boldsymbol{G}(\boldsymbol{r}^{(i)}) + \boldsymbol{R} \tag{3.62}$$

In the context of the finite element method, the most commonly used procedure for numerical integration is Gauss quadrature or more precisely Gauss-Legendre quadrature. With this integration type, the weights w_i and the sampling point coordinates $\boldsymbol{r}^{(i)}$, also referred to as Gauss or integration points, are optimized to accurately integrate polynomials of a given degree. Since the domain over which the integration extends is well known, i.e. $\tilde{\Omega}_e = [-1,1]^3$, the coordinates and weights can be precomputed and tabulated. Gauss point coordinates of different element domains and varying numbers of integration points, as well as their associated weights, can be found in various text books, e.g. in Zienkiewicz et al. [174, pp. 162-166].[4] Note that using Gauss quadrature with n sampling points and undistorted elements yields exact results ($\boldsymbol{R} = 0$) for integrands with a polynomial degree of $2n-1$ or less. When integrating elements with curved boundaries or otherwise distorted elements, an integration error will arise. In these cases, the Jacobian matrix and its determinant will be non-constant and thus the integrand is a rational function that cannot be integrated exactly. However, the error should be small for elements with reasonable distortion and size.

Applying numerical integration, the element stiffness matrix can be rewritten as

$$\boldsymbol{K}^e = \sum_{i=1}^{n_{ip}} w_i\,\boldsymbol{B}^T(\boldsymbol{r}^{(i)})\,\underline{\boldsymbol{D}}(\boldsymbol{r}^{(i)})\,\boldsymbol{B}(\boldsymbol{r}^{(i)})\det \boldsymbol{J}(\boldsymbol{r}^{(i)}), \tag{3.63}$$

where the product of all terms following the weights w_i constitutes the integrand $\boldsymbol{G}(\boldsymbol{r})$ whose polynomial degree in $\boldsymbol{r}$ determines the number of required Gauss points. By similar considerations, also the element vector of equivalent nodal forces in eq. (3.50) can be expressed as a summation series

$$\boldsymbol{f}^e = \boldsymbol{f}^e_{body} + \boldsymbol{f}^e_{surface} \tag{3.64}$$

[4] The first publication of the Gauss point coordinates and weights originates from Lowan et al. [108].

with

$$\boldsymbol{f}^e_{body} = \sum_{i=1}^{n_{ip}} w_i \boldsymbol{N}^T(\boldsymbol{r}^{(i)})\, \bar{\boldsymbol{b}}(\boldsymbol{r}^{(i)}) \det \boldsymbol{J}(\boldsymbol{r}^{(i)}) \tag{3.65}$$

$$\boldsymbol{f}^e_{surface} = \sum_{j=1}^{n^S_{ip}} w_j \boldsymbol{N}^T(\boldsymbol{r}^{(j)})\, \bar{\boldsymbol{t}}(\boldsymbol{r}^{(j)}) \det \boldsymbol{J}^S(\boldsymbol{r}^{(j)}) \tag{3.66}$$

where the upper index S in $\boldsymbol{J}^S$ represents the Jacobian of the underlying surface of the surface load vector, and n^S_{ip} is the number of integration points required to integrate this surface.

3.4 Spline geometry

3.4.1 Introduction

In the classical finite element method, a body that has its geometry already defined is approximated with a mesh of finite elements. The quality of this approximation depends on several factors, in particular on the original shape of the body, the size and number of the elements, and the shape functions used in the element formulation. With an isoparametric formulation, these functions were originally chosen to express the unknown field variable. They "only happen" to approximate the geometry by virtue of the method.

The concept of isogeometric analysis is, at its heart, just the classical finite element method with a different type of shape functions used for the element formulation. However, the initial approach is a different one. The original geometry of the body is not approximated but defined by a certain type of functions. These functions are then used as the shape functions of the element formulation, where it is provided that they are suited to do so. As a consequence, the analysis bases by definition on the correct geometry.

Since the first days of computer graphics (CG) and computer aided design in the 1960s, spline geometry was an essential means of shape description. After several evolutionary milestones, non-uniform rational B-splines turned into the de facto standard in the CAD industry (cf. Rogers [135, p. xv]). As NURBS also fulfill the requirements for the shape functions in the setting of a finite element formulation, they were the functions of choice when the concept of isogeometric analysis was introduced in 2005 by Hughes et al. [83]. They are also the basis of the isogeometric formulations in this thesis.

As the geometry description is essential for isogeometric analyses, this section shall give a brief overview of the different formulations prior to incorporating them in the context of the finite element method. Since NURBS are best understood when explained by the steps of its evolution, *Bézier* curves and standard *B-spline* curves will be briefly addressed before turning the attention to NURBS curves. After discussing the one dimensional formulations, the extension to higher dimensions, i.e. to surfaces and volumes will be presented.

A comprehensive portrayal of historical developments and the current state of NURBS based geometry can be found in the book of Rogers [135]. A more technical and extensive presentation on NURBS is given by Piegl and Tiller [129]. Farin finally provides an overview of a variety of curve and surface formulations used in computer aided graphical design in [62].

3.4.2 Parametric curves in general

Before introducing the individual curve formulations, it is advantageous to make a few general remarks. Two-dimensional curves can mathematically be expressed by one of the following formulations:

$$\text{explicit:} \qquad y = f(x) \tag{3.67}$$

$$\text{implicit:} \qquad f(x, y) = 0 \tag{3.68}$$

$$\text{parametric:} \qquad \boldsymbol{C}(\xi) = \begin{pmatrix} x(\xi) \\ y(\xi) \end{pmatrix} \quad \text{for} \quad a \leq \xi \leq b \tag{3.69}$$

For explicit representations, it is difficult to handle multi-dimensional objects and unfeasible to associate multiple y-values with a single x-value, as would be required for the description of a full circle. Therefore, they are rarely used for computer graphics. With implicit formulations, both is possible. However, the curve definition depends, just like it does for explicit representations, on the specific coordinate system. Therefore, geometric transformations are difficult to handle. Parametric curve definitions, which are a function of an independent parameter ξ, do not depend on a specific coordinate system. This can easily be seen on the example of the parametric representation of a straight line

$$\boldsymbol{C}(\xi) = \boldsymbol{P}^{(1)} + (\boldsymbol{P}^{(2)} - \boldsymbol{P}^{(1)})\xi \qquad \text{for} \quad 0 \leq \xi \leq 1, \tag{3.70}$$

in which $\boldsymbol{P}^{(i)}$ denotes the position vector of point i. This example also demonstrates further properties of parametric curves: Extending the definition to a higher dimension is as simple as appending the respective components of the additional dimension to the position vectors. Also do parametric curves possess a well-defined beginning and end, as well as a natural order of traversal, all determined by the independent parameter ξ. Consequently, curves with infinite extension are better expressed by implicit representations. Implicit and explicit representations also differ in the complexity of fundamental geometric operations. For implicit curve definitions, it is cumbersome to compute the points of the curve, which is simple for parametric curves. Then again, determining whether a given point is located on a curve, is difficult for parametric expressions but not for implicit formulations.

Splines are parametric curves and therefore possess the named properties, advantages and disadvantages. They are defined in terms of an independent parameter ξ. The space of this parameter is referred to as *parameter space*, with coordinates in this space being denoted as $\boldsymbol{\xi}$, that is $\begin{bmatrix} \xi & \eta & \zeta \end{bmatrix}^T$ in $\mathbb{R}^3$.

The parametric equation (3.70) of the straight line can also be written as

$$\boldsymbol{C}(\xi) = \sum_{i=1}^{2} N^{(i)}(\xi)\, \boldsymbol{P}^{(i)} \qquad \forall \xi \in \hat{\Omega} \tag{3.71}$$

with

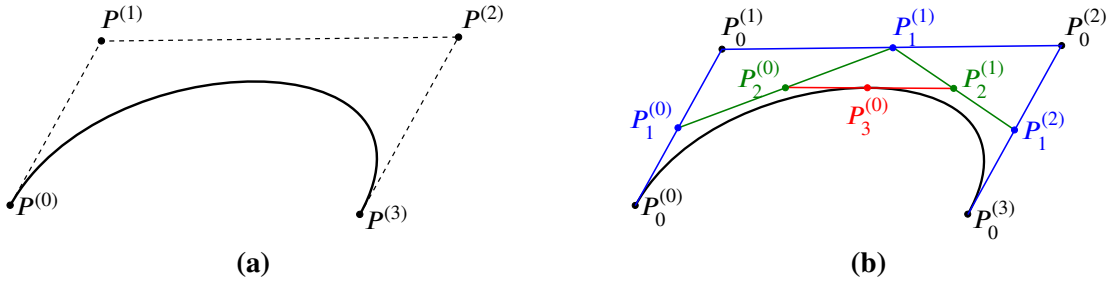
$$N^{(1)}(\xi) = 1 - \xi \qquad \text{and} \qquad N^{(2)}(\xi) = \xi,$$

where $\hat{\Omega}$ denotes the domain of the line in parameter space, being defined as $\hat{\Omega} = [0, 1]$ in this case. Equation (3.71) is the mapping of the parametric line or curve from parameter space to

the global coordinate space, i.e. $\boldsymbol{C}(\xi) : \hat{\Omega} \rightarrow \Omega \subset \mathbb{R}^3$. Contrasting the parameter space, the global coordinate space is in the context of isogeometric analysis also referred to as *physical space*, where it does not necessarily have to name the global system, but can also stand for a local system that preserves the physical extents of a given object. The image of the parameter domain in physical space is the physical domain Ω. The functions N performing the mapping are denoted *blending functions* or *basis functions* in the world of computer graphics. Here, the latter term will be used.

3.4.3 Bézier curves

The development of Bézier curves is credited to Pierre Bézier and Paul de Casteljau. Both worked for French car manufacturers, Bézier for Renault and de Casteljau for Citroën. In the early 1960s, they sought independently for a way to describe the hulls of their companies' automobiles mathematically for the use with the emerging methods of computer aided design and computer aided manufacturing (CAM) [135].



(a) **(b)**

Figure 3.3: Example of a Bézier curve. (a) Cubic Bézier curve with its control polygon connecting the control points, and (b) the evaluation of a point on that curve at $\xi = 0.5$ with the *de Casteljau* algorithm.

A Bézier curve as shown in fig. 3.3(a) is defined as

$$\boldsymbol{C}(\xi) = \sum_{i=0}^{n} N_p^{(i)}(\xi)\, \boldsymbol{P}^{(i)} \qquad \forall \xi \in [0,1]\,, \tag{3.72}$$

where the i^{th} basis function $N_p^{(i)}$ is the *Bernstein polynomial*[5] of degree p.[6]

$$N_p^{(i)}(\xi) = \binom{p}{i} \xi^i (1-\xi)^{p-i} \qquad \text{with} \qquad 0^0 \equiv 1 \qquad \text{and} \qquad \binom{p}{i} = \frac{p!}{i!\,(p-i)} \tag{3.73}$$

Bézier curves are polynomial curves that require $p+1$ control points $\boldsymbol{P}$ for a curve of degree p, hence, $n = p$ in the definition above. With the curve being defined on the parameter domain $\hat{\Omega} = [0,1]$, it will always start in control point $\boldsymbol{P}^{(0)}$ and end in $\boldsymbol{P}^{(n)}$, but will in general not pass through the other control points $\boldsymbol{P}^{(1)}$ to $\boldsymbol{P}^{(n-1)}$. The dashed lines connecting the control

[5] Named after the Russian mathematician Sergei Bernstein, who published the polynomial originally in the proof of Stone–Weierstrass approximation theorem [31]. In 1972, Robin Forrest [65] demonstrated the equivalence of the Bernstein polynomial with the work done by Paul Bézier [32].

[6] In the context of CG and CAD, the order of a polynomial is always equal to the polynomial degree $p+1$. This distinction is not made by people dealing with finite elements, therefore it is adopted in this work.

points of the curve in fig. 3.3(a) constitute the control polygon of that curve. Moving any of the control points will change the shape of the entire curve in an intuitive manner, as the curve will follow the modified shape of the control polygon. This type of control established the curve's popularity in the world of computer aided design. The geometric construction of a point on the curve with the help of the *de Casteljau* algorithm is depicted in fig. 3.3(b). The curve $\boldsymbol{C}(\xi)$ is evaluated at $\xi = 0.5$ by consecutively interpolating the lines connecting the control points at 0.5, creating new lines connecting the interpolated points and starting over until at iteration p the point $\boldsymbol{C}(0.5)$ is found.

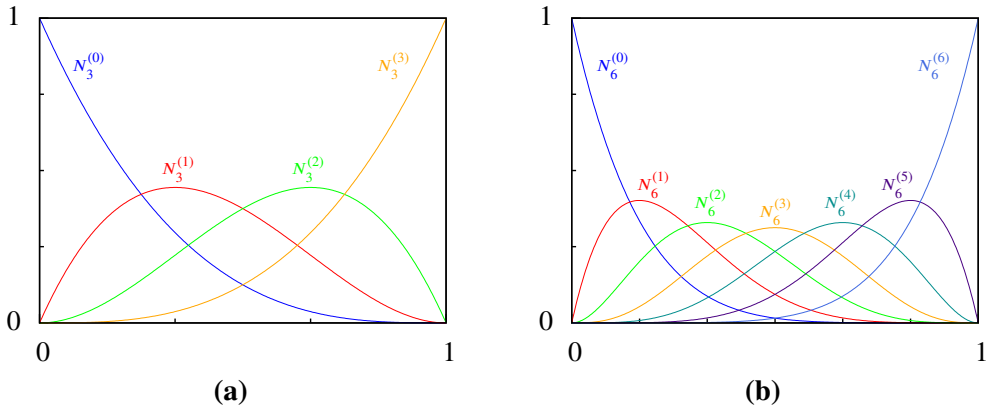


Figure 3.4: Bernstein polynomials of different degree plotted over the domain $[0, 1]$, i.e. the basis functions of a Bézier curve. (a) For the depicted curve in fig. 3.3(a) with degree 3; (b) the functions of a curve with seven control points ($p = 6$).

The basis functions of the curve in fig. 3.3(a) are plotted in fig. 3.4(a), and for the reason of comparison, the respective functions of a Bézier curve with seven control points, i.e. a curve of polynomial degree 6, are shown in fig. 3.4(b). The functions are non-negative and fulfill the partition of unity property. It is to be noted, that all functions are nonzero over the entire domain $]0, 1[$. This is the reason, why manipulating the position of a single control point influences the shape of the entire curve – irrespective of the number of control points of the curve. As none of the function values becomes one over $]0, 1[$, the curve does not pass through any of the control points $\boldsymbol{P}^{(1)}$ to $\boldsymbol{P}^{(n-1)}$ and reversely, passes through the start and end point, where the respective basis functions evaluate to one at $\xi = 0$ and $\xi = 1$.

3.4.4 B-spline curves

Owing to the use of the Bernstein polynomial as the basis functions of the Bézier curve, it is not possible to define a curve of low degree with a higher number of control points, i.e. more control points than $p + 1$. The basis also prohibits the application of local changes on a given curve.[7] These shortcomings were overcome with B-spline curves that use piecewise polynomials as their basis. Instead of defining the curve over $\hat{\Omega} = [0, 1]$, the domain of a B-spline curve in

[7] Both statements are only partially true, as they disregard the possibility of defining multiple connected Bézier curves and specifying continuity conditions between them (cf. [135, pp.31-32]). However, in mathematical terms this would not lead to single curve and it would impose severe restrictions on the curve when achieving higher order continuity at the connection is desired.

parameter space is given as $\hat{\Omega} = \left[\xi_{p+1}, \xi_{n+1}\right]$, where the variables ξ_{p+1} and ξ_{n+1} stem from a set of coordinates Ξ.

$$\Xi = \left\{\xi_1, \xi_2, \ldots, \xi_{n+p}, \xi_{n+p+1}\right\} \tag{3.74}$$

Ξ is referred to as the *knot vector* associated with a given B-spline curve. It has $n + p + 1$ entries, with n being the number of control points and p the polynomial degree of the B-spline basis functions. The coordinates in the knot vector, commonly referred to as *knots*, must not be decreasing, i.e.

$$\xi_i \leq \xi_{i+1} \tag{3.75}$$

must hold for all entries. These coordinates are used for the evaluation of the B-spline basis functions $N_p^{(i)}$ with the *Cox-de Boor* recursion formula,[8] which is given in eq. (3.76).

$$N_0^{(i)}(\xi) = \begin{cases} 1 & \text{for} \quad \xi_i \leq \xi < \xi_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{and for } p > 0 \tag{3.76}$$

$$N_p^{(i)}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} N_{p-1}^{(i)}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} N_{p-1}^{(i+1)}(\xi)$$

Note that $N_{p-1}^{(i)}$ and $N_{p-1}^{(i+1)}$ are required for the evaluation of $N_p^{(i)}$. The entries in the knot vector subdivide the parameter space into individual *knot spans*, with each basis function i beginning at ξ_i and spanning exactly $p + 1$ knot spans.

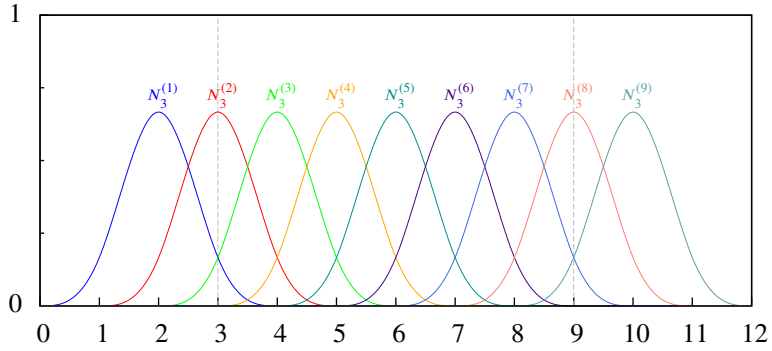


Figure 3.5: B-spline basis functions of degree 3 for the periodic uniform knot vector $\Xi = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$. The dashed grey lines at $x = 3$ and $x = 9$ symbolize the domain boundary of an associated B-spline curve.

Figure 3.5 depicts the basis functions with polynomial degree 3 for a periodic uniform knot vector $\Xi = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$. All basis functions are identical but shifted by one knot span, they all start at distinct knots and they also end at distinct knots. The domain of an associated B-spline curve evaluates to $\hat{\Omega} = [3, 9]$, marked by the vertical dashed lines in the figure. On this domain, the partition of unity property $\sum_{i=1}^{n} N_3^{(i)}(\xi) = 1 \ \forall \xi \in \hat{\Omega}$ is fulfilled.

[8] In the early 1970s, Maurice Cox [44] and Carl de Boor [47] independently published this method to efficiently evaluate the B-spline basis. Also other methods of defining the B-spline basis exist, but due to its efficiency, the recurrence relation by Cox and de Boor is frequently used in implementations (cf. [129, p. 50].

However, contrasting the Bézier basis, none of the functions evaluates to one at the domain boundaries. Therefore, a B-spline curve associated with this basis and the corresponding knot vector does neither begin nor end at one of its control points. None of the control points is fully interpolated. This characteristic renders the curve difficult to handle. Consequently, periodic uniform knot vectors are rarely used.

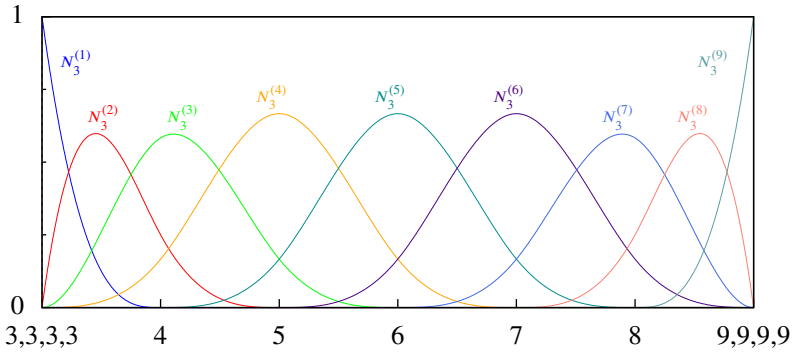


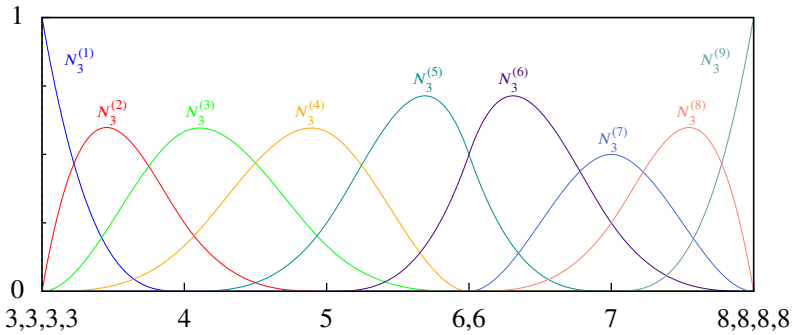
Figure 3.6: B-spline basis functions for $\Xi = \{3, 3, 3, 3, 4, 5, 6, 7, 8, 9, 9, 9, 9\}$, referred to as open uniform knot vector. The polynomial degree is again 3. The domain of an associated B-spline curve equals the space covered by the basis functions.

The repetition of the first and last value in the knot vector $p+1$ times causes the first and respectively the last p knot spans to be of length zero, leading to a concentration of basis functions around the beginning and the end of the parameter space. This can be observed in fig. 3.6, where the cubic basis functions are plotted for

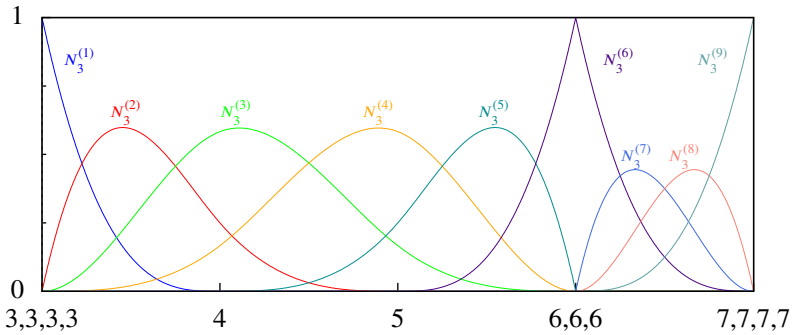
$$\Xi = \{\underbrace{3, 3, 3, 3}_{p+1}, 4, 5, 6, 7, 8, \underbrace{9, 9, 9, 9}_{p+1}\}.$$

In this example, the basis function $N_3^{(1)}$ still covers $p + 1$ knot spans, but just one span has nonzero length. The next function, $N_3^{(2)}$, covers two nonzero length spans. This scheme proceeds until the function $N_3^{(p+1)}$ is finally the same regular function as those in fig. 3.5. With this modification of the knot vector, the first and last basis functions evaluate to unity at the domain boundary where all other function values become zero. Hence, an associated B-spline curve is interpolatory at the first and last control points. The partition of unity property is fulfilled for the entire domain. Knot vectors with $p + 1$ identical values at their beginning and their end are named *open knot vectors*. These knot vectors are, for the given reasons, the standard in CAD.

The knot vector provides even more control over the basis. It can be used to influence its continuity. The piecewise polynomial basis functions for an open uniform knot vector, depicted in fig. 3.6, are of continuity C^p over the length of the knot spans. Exactly on the knots, where in the standard case one pair of polynomials connect, the continuity is C^{p-1}. Increasing the multiplicity of a knot value that is not on the boundary of the parameter space further reduces the continuity at that knot and increases the number of connecting polynomials. Nonetheless, the continuity is preserved over the knot spans. In general, the continuity at a given knot is C^{p-m} with m denoting the multiplicity of the given knot value. An example for reduced continuity is given in fig. 3.7.



(a) B-spline basis functions for $\Xi = \{3, 3, 3, 3, 4, 5, 6, 6, 7, 8, 8, 8, 8\}$



(b) B-spline basis functions for $\Xi = \{3, 3, 3, 3, 4, 5, 6, 6, 6, 7, 7, 7, 7\}$

Figure 3.7: B-spline basis functions of polynomial degree 3 with reduced continuity due to an increased multiplicity of the knot value 6. In (a) the continuity is $C^{(3-2)} = C^1$ at the repeated knot, and in (b) it is C^0.

With the knot vector of eq. (3.74) and the B-spline basis functions given by the Cox-de Boor recursion formula in eq. (3.76), a B-spline curve is defined as

$$\boldsymbol{C}(\xi) = \sum_{i=1}^{n} N_p^{(i)}(\xi)\, \boldsymbol{P}^{(i)} \qquad \forall \xi \in \hat{\Omega}, \tag{3.77}$$

which resembles the definition of the Bézier curve. And in fact, for an open knot vector with just one nonzero knot span, i.e. $\Xi = \{0, 0, 0, 0, 1, 1, 1, 1\}$ in the cubic case, the B-spline curve and its basis are identical to the Bézier curve. Therefore, the Bézier curves are included as a special case of the B-spline curves.

For an identical set of control points, but three different knot vectors and therefore different basis functions, three B-spline curves are displayed in fig. 3.8. The knot vectors and basis functions are the ones given in figs. 3.6 and 3.7. Observe the full interpolation of $\boldsymbol{P}^{(6)}$ for the variant that is C^0 at the knot value $\xi = 6$.

3.4.5 Rational B-spline curves

Compared with Bézier curves, B-spline curves provide more flexibility for geometric modeling. However, as both types of curves base upon polynomial basis functions, one drawback persists. Many conic sections that are frequently used in geometric modeling, such as ellipses, circles,

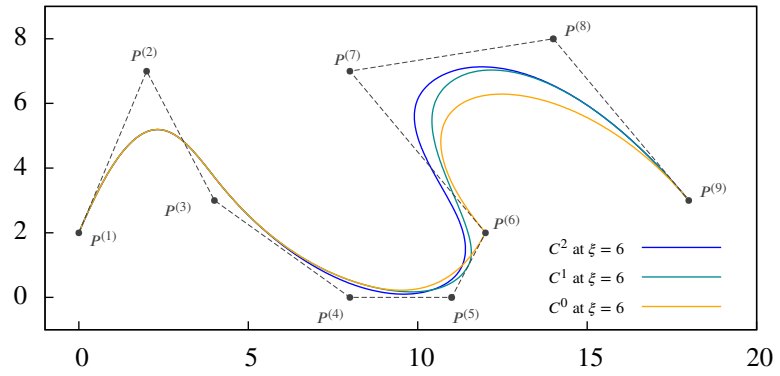


Figure 3.8: Three B-spline curves with polynomial degree 3, evaluated with identical sets of control points, but with basis functions that have different continuity at knot value $\xi = 6$ (cf. figs. 3.6 and 3.7).

and hyperbolas, cannot be represented exactly. This limitation was overcome with the use of rational basis functions for B-spline curves. Rational B-spline curves, known as *NURBS*, were first discussed in the thesis of Versprille [162]. They are defined as polynomial B-spline curves in $\mathbb{R}^{d+1}$ homogeneous space and then pulled back by projective transformation to $\mathbb{R}^d$ physical space.

If a point in 4D homogeneous space is given by its position vector

$$\boldsymbol{P}^w = \begin{bmatrix} xw & yw & zw & w \end{bmatrix}^T \tag{3.78}$$

then its projection through the origin onto the hyperplane defined by $w = 1$ results in

$$\boldsymbol{P} = \begin{bmatrix} x & y & z \end{bmatrix}^T, \tag{3.79}$$

which is the respective point in 3D physical space. Accordingly, for a (non-rational) B-spline curve in homogeneous space

$$\boldsymbol{C}^w(\xi) = \sum_{i=1}^{n} N_p^{(i)}(\xi) \boldsymbol{P}^{w(i)} \qquad \forall \xi \in \hat{\Omega}, \tag{3.80}$$

its projection into physical space is given by

$$\left(\boldsymbol{C}(\xi)\right)_j = \frac{\left(\boldsymbol{C}^w(\xi)\right)_j}{\sum_{i=1}^{n} N_p^{(i)}(\xi) w^{(i)}} \qquad \forall \xi \in \hat{\Omega} \quad \text{and} \quad j = 1, \ldots, d, \tag{3.81}$$

which states that the homogeneous coordinate components of any point on the curve evaluated at ξ are divided by the value of w at ξ. In eqs. (3.80) and (3.81), $N_p^{(i)}$ are the B-spline basis functions defined in eq. (3.76), the lower index j denotes one of the coordinate components 1 through d, and $w^{(i)}$ is the $(d+1)^{th}$ scalar component of the position vector of the homogeneous control point $\boldsymbol{P}^{w(i)}$, commonly also referred to as the *weight* of the control point i, for which $w^{(i)} \neq 0$ must apply. Reformulating eq. (3.81) with reference to the control points in physical coordinates, leads to the following definition of a NURBS curve

$$\boldsymbol{C}(\xi) = \sum_{i=1}^{n} R_p^{(i)}(\xi) \boldsymbol{P}^{(i)} \qquad \forall \xi \in \hat{\Omega} \tag{3.82}$$

with the rational NURBS basis functions

$$R_p^{(i)}(\xi) = \frac{N_p^{(i)}(\xi) w^{(i)}}{\sum_{\hat{i}=1}^{n} N_p^{(\hat{i})}(\xi) w^{(\hat{i})}}. \tag{3.83}$$

As can be seen from eq. (3.83), the weights provide another possibility to modify the basis functions and thus to alter the NURBS curve. To illustrate this, the open uniform knot vector and the control points of the B-spline curve in fig. 3.8 were reused to plot the cubic[9] NURBS curves in fig. 3.9. In this figure, the weight of the control point $\boldsymbol{P}^{(6)}$ was varied for plot of the four different curves, while the weights of all other control points were kept constant at $w^{(i)} = 1, i \neq 6$. The comparison of the resultant curves demonstrates how the weight loosens ($w^{(i)} < 1$) or strengthens ($w^{(i)} > 1$) the local influence of a control point on the curve. The rational NURBS basis functions that underlie the four different curves of fig. 3.9 are shown in fig. 3.10.

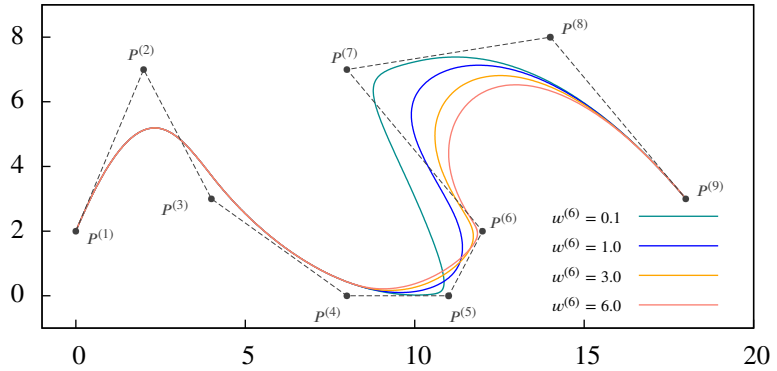
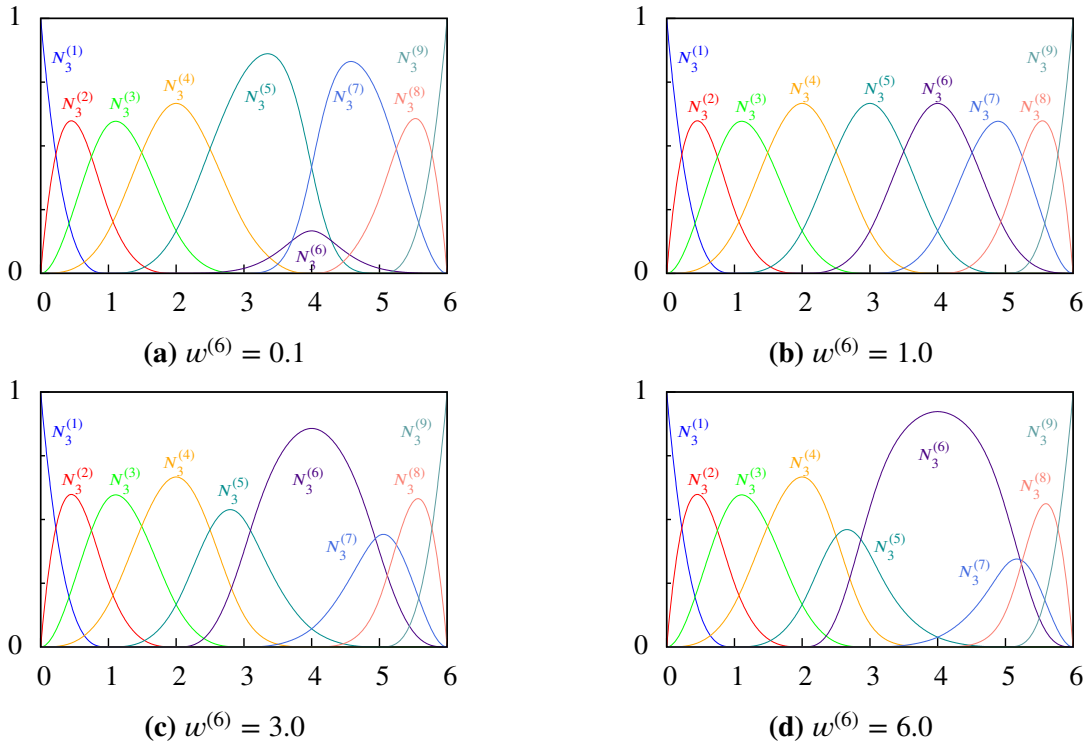


Figure 3.9: Four cubic NURBS curves with open uniform knot vector. For each curve plot, the weight of control point $\boldsymbol{P}^{(6)}$ was varied.

For the case of all weights being equal to unity, the NURBS curve in fig. 3.9 is identical to the B-spline curve in fig. 3.8. This also follows directly from eq. (3.83) and the partition of unity property of the B-spline basis. As can easily be seen, in the case of all weights taking the same value, $R_p^{(i)} = N_p^{(i)}$ and consequently, the NURBS basis remains a polynomial. Hence, B-spline curves are a special case of NURBS curves.

Like the B-spline basis, the NURBS basis fulfills the partition of unity property and any basis function $R_p^{(i)}$ is nonzero only on the interval $\left[\xi_i, \xi_{i+p+1}\right]$, which means that its support and thus the influence of a single control point is restricted to $p+1$ knot spans. Also, the results from the discussion about the continuity at knots and knot spans in sect. 3.4.4 pertain to NURBS in a similar manner. Further properties, that have not yet been explicitly mentioned, but apply equally to B-spline and NURBS curves, is the non-negativity of the basis, the linear independence of the basis, and the variation diminishing property, the later stating that the curve will not intersect any line or plane more often than its control polygon does. Thus, it will not oscillate as Lagrangian polynomials do when interpolating discontinuous data. Transforming a curve in physical space, i.e. translating, rotating, shearing or scaling it, is achieved by solely

[9] Though the curve is expressed by a rational function, it is still associated with the polynomial degree of the underlying B-spline basis function.



(a) $w^{(6)} = 0.1$ **(b)** $w^{(6)} = 1.0$

(c) $w^{(6)} = 3.0$ **(d)** $w^{(6)} = 6.0$

Figure 3.10: Cubic NURBS basis functions for nine control points, with open uniform knot vector $\Xi = \{0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 6, 6, 6\}$ and varying weight $w^{(6)}$.

transforming the position vectors of the control points. Therefore, the curve is invariant to affine transformations.

Before ending the discussion about the three different spline curves, two geometric operations shall be introduced, that will be of upmost importance when returning to the topic of finite element analysis.

Knot insertion

The instrument of *knot insertion* can be utilized to gain more control over a given NURBS curve. Inserting k new knots $\bar{\xi}_i$ into the knot vector of that curve increases not only the number of knot spans but also refines the control polygon, which, due to the knot insertion, is defined by k additional control points. Since the geometry of the curve itself is not to be modified by this operation, the new control points must be calculated adequately. The existing curve in homogeneous coordinate space is denoted $\boldsymbol{C}^w(\xi)$ and after knot insertion the curve is identified by $\overline{\boldsymbol{C}^w}(\xi)$. When the associated knot vectors are Ξ and $\overline{\Xi}$ respectively, then $\Xi \subset \overline{\Xi}$. It must be required that $\boldsymbol{C}^w(\xi) = \overline{\boldsymbol{C}^w}(\xi)$. Using the relation in eq. (3.80), this requirement can be expanded to

$$\sum_{i=1}^{n} N_p^{(i)}(\xi)\boldsymbol{P}^{w(i)} = \sum_{\bar{i}=1}^{n+k} \overline{N}_p^{(\bar{i})}(\xi)\overline{\boldsymbol{P}^w}^{(\bar{i})}. \tag{3.84}$$

Evaluating eq. (3.84) $n+k$ times for adequate values of ξ leads to a linear system of equations, that can be solved for the unknown new control points $\overline{\boldsymbol{P}^{w(\bar{i})}}$. A more sophisticated approach

of retrieving the new control points, as well as an efficient implementation algorithm can be found in Piegl and Tiller [129, pp. 141-151]. The knot insertion algorithm is demonstrated in fig. 3.11. Figure 3.11(a) shows the original curve and the control points, fig. 3.11(c) depicts the curve after knot insertion, and fig. 3.11(b) contains the basis functions before and after. With steady refinement by knot insertion, the control polygon converges to the shape of the curve.

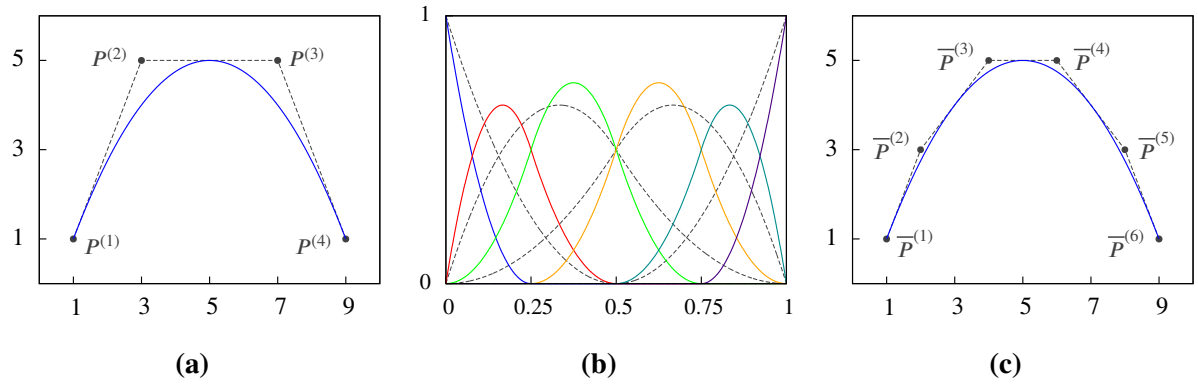


Figure 3.11: Knot insertion algorithm for a simple curve of polynomial degree 2. (a) The original NURBS curve with $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$, (b) enriched basis $\overline{N}_p^{(\bar{i})}$ with the original basis as dashed lines in the background, (c) the NURBS curve after inserting knots 0.25 and 0.75 into Ξ.

Degree elevation

The second geometric operation to be briefly discussed is known as *degree elevation*. With this operation, the basis is globally enriched by raising the polynomial degree of the basis functions from p to $\overline{p} = p + 1$. Like knot insertion, also degree elevation requires a recalculation of the control points, but here, the new knot vector $\overline{\Xi}$ is not initially known. However, as the shape of the curve is to be identical before and after degree elevation, a new knot vector can be constructed by continuity considerations. In order to retain the full interpolation property of the start and end points of the curve, the new knot vector must also be an open knot vector and therefore, the first and last knot values in $\overline{\Xi}$ must have a multiplicity $\overline{m} = \overline{p} + 1$. For all other knots in the original knot vector, the continuity of the curve at these coordinates in parameter space is to be preserved. Hence, $p - m_i = \overline{p} - \overline{m}_i$ must hold for all $\xi_i \in \{\xi_{p+2}, \ldots, \xi_n\}$. This suffices to define $\overline{\Xi}$ and with the size of $\overline{\Xi}$ being $\overline{n} + \overline{p} + 1$, the number of new control points is also known. With a similar relation as given in eq. (3.84), the new control points can be calculated from

$$\sum_{i=1}^{n} N_p^{(i)}(\xi) \boldsymbol{P}^{w(i)} = \sum_{\bar{i}=1}^{\overline{n}} \overline{N}_{p+1}^{(\bar{i})}(\xi) \overline{\boldsymbol{P}^w}^{(\bar{i})}. \tag{3.85}$$

Again, more efficient algorithms exist and can be found in [129]. A simple NURBS curve before and after degree elevation is shown in fig. 3.12. Note that in fig. 3.12(b), the number of knot spans with nonzero length remains constant with degree elevation.

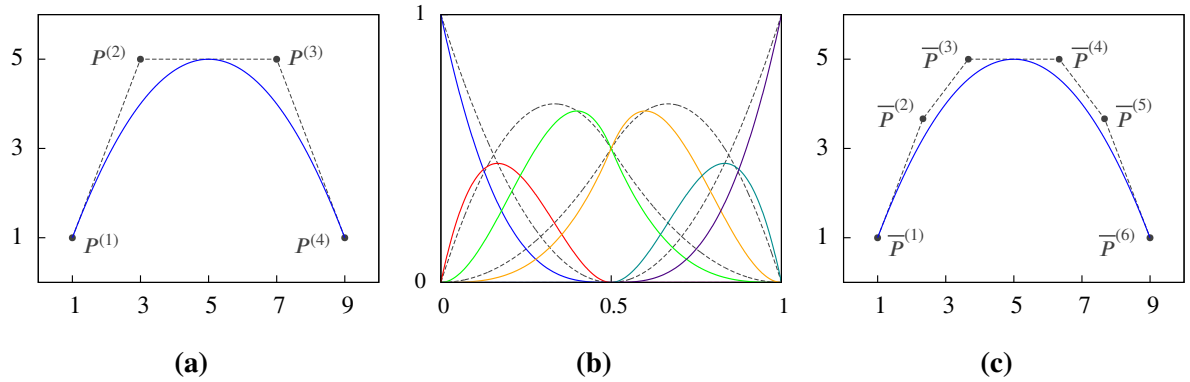


Figure 3.12: Degree elevation algorithm for a simple curve of polynomial degree 2. (a) The original NURBS curve with $\Xi = \{0, 0, 0, 0.5, 1, 1, 1\}$, (b) enriched basis $\overline{N}^{(\bar{i})}_{p+1}$ with the original basis as dashed lines in the background, (c) the NURBS curve after raising the polynomial degree of the basis function to 3 and the knot vector now being $\overline{\Xi} = \{0, 0, 0, 0, 0.5, 0.5, 1, 1, 1, 1\}$.

3.4.6 Surface and volume representations

Advancing from one dimensional NURBS curves to higher dimensional surface and volume representations is achieved with the construction of the tensor product of two or three curves, respectively. This results in a NURBS surface being defined as

$$\boldsymbol{S}(\xi, \eta) = \sum_{i=1}^{n} \sum_{j=1}^{m} R^{(i,j)}_{p,q}(\xi, \eta)\, \boldsymbol{P}^{(i,j)} \qquad \forall (\xi, \eta) \in \hat{\Omega}^2 \tag{3.86}$$

with the bivariate NURBS basis functions

$$R^{(i,j)}_{p,q}(\xi, \eta) = \frac{N^{(i)}_p(\xi)\, N^{(j)}_q(\eta)\, w^{(i,j)}}{\sum_{\hat{i}=1}^{n} \sum_{\hat{j}=1}^{m} N^{(\hat{i})}_p(\xi)\, N^{(\hat{j})}_q(\eta)\, w^{(\hat{i},\hat{j})}} \tag{3.87}$$

and the domain in parameter space being

$$\hat{\Omega}^2 = \left[\xi_{p+1}, \xi_{n+1}\right] \times \left[\eta_{q+1}, \eta_{m+1}\right].$$

Likewise, a NURBS solid or volume is expressed by

$$\boldsymbol{V}(\xi, \eta, \zeta) = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{l} R^{(i,j,k)}_{p,q,r}(\xi, \eta, \zeta)\, \boldsymbol{P}^{(i,j,k)} \qquad \forall (\xi, \eta, \zeta) \in \hat{\Omega}^3. \tag{3.88}$$

Then, the trivariate NURBS basis functions are

$$R^{(i,j,k)}_{p,q,r}(\xi, \eta, \zeta) = \frac{N^{(i)}_p(\xi)\, N^{(j)}_q(\eta)\, N^{(k)}_r(\zeta)\, w^{(i,j,k)}}{\sum_{\hat{i}=1}^{n} \sum_{\hat{j}=1}^{m} \sum_{\hat{k}=1}^{l} N^{(\hat{i})}_p(\xi)\, N^{(\hat{j})}_q(\eta)\, N^{(\hat{k})}_r(\zeta)\, w^{(\hat{i},\hat{j},\hat{k})}} \tag{3.89}$$

while the domain in parameter space is

$$\hat{\Omega}^3 = \left[\xi_{p+1}, \xi_{n+1}\right] \times \left[\eta_{q+1}, \eta_{m+1}\right] \times \left[\zeta_{r+1}, \eta_{l+1}\right].$$

In the equations above ξ, η, and ζ denote the coordinates in the parameter space of the respective one dimensional curves. $\boldsymbol{\Xi}$, $\mathbf{H}$, and $\mathbf{Z}$ are the associated knot vectors and p, q, and r are the polynomial degrees of the univariate B-spline basis functions. The number of control points in the directions of the respective parametric coordinates is n, m, and l. Due to the tensor product structure, the total number of control points is $n \times m$ for a surface and $n \times m \times l$ for a solid and accordingly, the control polygon becomes a *control net* in the case of a surface and a *control lattice* for a solid. Obviously, this type of structure allows for independent polynomial degrees, number of knot spans and therewith also independent number of control points for the different parametric directions. Without regard to the number of spatial dimensions, a single NURBS object is referred to as a NURBS *patch*.

The properties of the multivariate formulations follow directly from their univariate counterparts. The basis functions fulfill the partition of unity. The support of a function $R^{(i,j,k)}_{p,q,r}$ is limited to $[\xi_i, \xi_{i+p+1}] \times [\eta_j, \eta_{j+q+1}] \times [\zeta_k, \zeta_{k+r+1}]$ and therewith local, as is the influence of a control point on the solid. The full interpolation of the two control points bounding the control polygon of a curve based on an open knot vector, i.e. the mapping of ξ_{p+1} to $\boldsymbol{P}^{(1)}$ and of ξ_{n+1} to $\boldsymbol{P}^{(n)}$, leads to an equivalent interpolation of the control points $\boldsymbol{P}^{(i,j,k)}$ $\forall\ i \in \{1, n\} \wedge j \in \{1, m\} \wedge k \in \{1, l\}$ when the parametric coordinates simultaneously assume values on their respective boundaries.

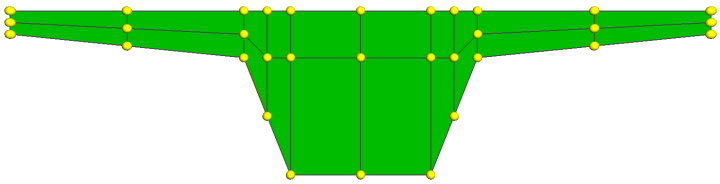


Figure 3.13: Bicubic NURBS surface with the sharp corners obtained by repeated coordinate values in one of the knot vectors, which are $\boldsymbol{\Xi} = \{0, 0, 0, 1, 1, 1\}$ and $\mathbf{H} = \{0, 0, 0, 1, 1, 3, 3, 5, 5, 7, 7, 8, 8, 8\}$. The figure displays the NURBS patch, the control points and the control net.

It is to note that a NURBS solid is bounded by six NURBS surfaces which are in turn bounded by twelve NURBS curves in total. Each of these bounding surfaces (and likewise the curves), can be recovered from the volume representation by fixing one of the parametric coordinates to its upper or lower bound. The evaluation of the bounding surface then only depends on the outmost control points of the chosen parametric direction. As an example, one of the bounding surfaces of the NURBS solid $\boldsymbol{V}$ is given as

$$\boldsymbol{V}(\xi_{p+1}, \eta, \zeta) = \sum_{j=1}^{m} \sum_{k=1}^{l} R^{(1,j,k)}_{p,q,r}(\xi_{p+1}, \eta, \zeta)\, \boldsymbol{P}^{(1,j,k)} \qquad \forall (\xi_{p+1}, \eta, \zeta) \in \hat{\Omega}^3 \tag{3.90}$$

which is exactly the same as

$$\boldsymbol{S}(\eta, \zeta) = \sum_{j=1}^{m} \sum_{k=1}^{l} R^{(j,k)}_{q,r}(\eta, \zeta)\, \boldsymbol{P}^{(1,j,k)} \qquad \forall (\eta, \zeta) \in \hat{\Omega}^2, \tag{3.91}$$

when in both equations $\boldsymbol{P}^{(i,j,k)}$ refers to the same set of control points and the domain of η and ζ to the same knot vectors.

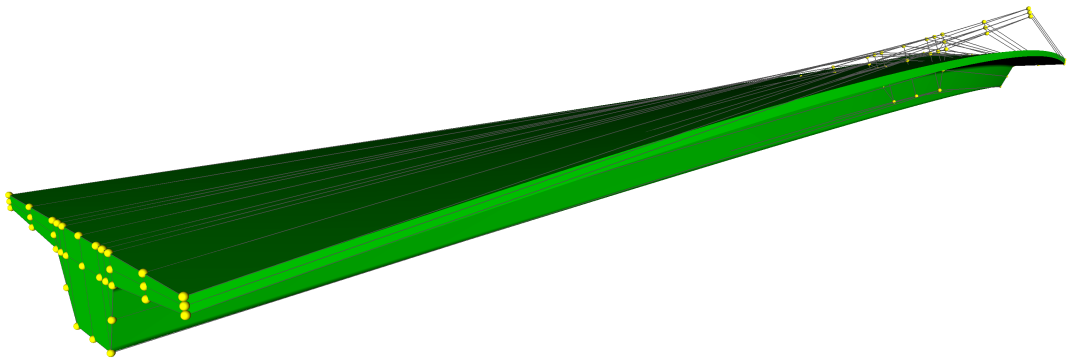


Figure 3.14: Tricubic NURBS solid representing a curved bridge deck, built from the tensor product of the surface in fig. 3.13 and a NURBS curve with three aligned control points and a knot vector $\mathbf{Z} = \{0, 0, 0, 1, 1, 1\}$. After building the tensor product solid, the control points in the middle of the deck were rotated around the longitudinal bridge axis to obtain the visible twist of the deck. The figure displays the NURBS patch, the control points and the control lattice.

An example of a bicubic NURBS surface defined by 3×11 control points and two knot vectors $\boldsymbol{\Xi}$ and $\mathbf{H}$ is shown fig. 3.13. Extruding that surface along a straight line defined by a cubic NURBS curve with 3 control points and a knot vector $\mathbf{Z} = \{0, 0, 0, 1, 1, 1\}$ and collectively rotating the center control points afterwards, leads to the NURBS volume representing a curved bridge deck depicted in fig. 3.14.

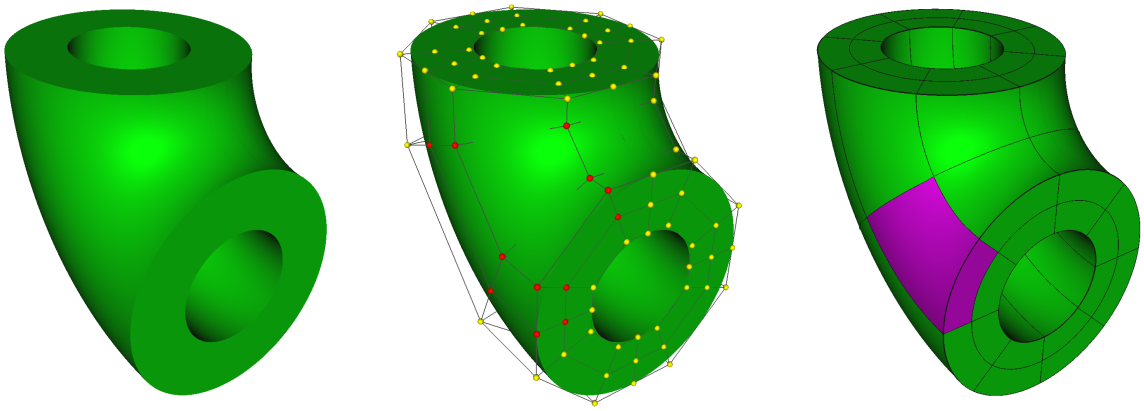
3.5 Analysis based on spline geometry

3.5.1 The mesh equivalent

This section on the use of non-uniform rational B-splines in the context of a finite element analysis begins with a discussion on the bent pipe NURBS solid adopted from [43]. The NURBS volume depicted in fig. 3.15 is modeled with quadratic functions in longitudinal and circumferential directions and linear functions in the through thickness direction. The knot vectors for the respective directions are $\boldsymbol{\Xi}$, $\mathbf{H}$, and $\mathbf{Z}$. For the purpose of clarity, there are more knot spans and control points than are actually required to define the geometry. The attention is to be focused on fig. 3.15(c) where the tensor product of the three knot vectors is projected via eq. (3.88) into physical space. This projection leads to a partitioning of the pipe into three-dimensional hexahedral cells with curved boundaries. These cells are the multi-dimensional equivalents of the knot spans – in physical space. Obviously, this resembles the partitioning of a body's domain Ω into non-overlapping subdomains Ω_e expressed in eqs. (3.29) and (3.30) in the context of the finite element discretization. And that is exactly, what this projection is used for in isogeometric analysis. The only difference is, that the partitioning must not be actively conducted by meshing the body, but it is inherent to the geometry defining the body. Consequently, eq. (3.29) becomes in the case of isogeometric analysis

$$\Omega = \Omega^h = \bigcup \Omega_e, \tag{3.92}$$

as no approximation is involved in the partitioning of the original geometry.



(a) (b) (c)

Figure 3.15: A section of a bent pipe modeled with a single NURBS patch. (a) Geometry of the bent pipe, cf. [43]. (b) Control lattice with control points defining the geometry of the NURBS solid in physical space. Those points colored in red are the (visible) control points that have support in the highlighted cell. (c) Projection of the tensor product of the knot vectors into physical space partitioning the shape into individual cells.

3.5.2 Field interpolations

Studying again the NURBS basis functions of fig. 3.10, it is apparent that in each knot span exactly $p+1$ basis functions are nonzero and that the basis functions exclusively start and end at knot values defined in the knot vector. With that in mind, the control points influencing the geometry of a given knot span, or of a given cell in the three-dimensional case, can easily be identified from eqs. (3.82) and (3.88), respectively. For the highlighted cell of the bent pipe example in fig. 3.15(c), they are shown in fig. 3.15(b).

Referring to the relevant control points and basis functions for a given cell, i.e., those functions whose product does not vanish, requires a unique numbering scheme. In order to define such a scheme for the trivariate basis functions of a NURBS solid, the patch global indices (i, j, k) of the basis functions' three parametric directions with $i \in 1, \dots, n,\ j \in 1, \dots, m,$ and $k \in 1, \dots, l$ are collapsed into

$$a = nm(k-1) + n(j-1) + i. \tag{3.93}$$

Since the number and structure of the basis functions is equivalent to the number and structure of the control points, the scheme applies to both. In a similar manner, a cell local numbering for the nonzero NURBS basis functions within a single cell can be established as

$$\hat{a} = (p+1)(q+1)(\mathfrak{k}-1) + (p+1)(\mathfrak{j}-1) + \mathfrak{i}, \tag{3.94}$$

with local indices $\mathfrak{i} \in 1, \dots, p+1,\ \mathfrak{j} \in 1, \dots, q+1,$ and $\mathfrak{k} \in 1, \dots, r+1$ and the total number of nonzero basis functions within a cell being

$$n_{en} = (p+1) \times (q+1) \times (r+1). \tag{3.95}$$

With these definitions, it is easy to construct an array that relates the 1 through n_{en} nonzero basis functions of any cell e to the patch wide numbering scheme and therefrom to the patch global

indices. Retrieving the indices (i, j, k) from that array is, depending on the input, denoted $f(\hat{a}, e)$ or $f(a)$. However, for brevity this notation will generally not be used and is to be assumed from the context. Consequently, all the nonzero functions $R^{(i,j,k)}_{p,q,r}$ of a given cell can be stored in matrix form as

$$\boldsymbol{N} = \begin{bmatrix} R^{(1)} & 0 & 0 & R^{(2)} & 0 & 0 & \dots & R^{(n_{en})} & 0 & 0 \\ 0 & R^{(1)} & 0 & 0 & R^{(2)} & 0 & \dots & 0 & R^{(n_{en})} & 0 \\ 0 & 0 & R^{(1)} & 0 & 0 & R^{(2)} & \dots & 0 & 0 & R^{(n_{en})} \end{bmatrix}. \tag{3.96}$$

If likewise the components of the control points' position vectors with support in that cell are stored as

$$\tilde{\boldsymbol{x}}^e = \begin{bmatrix} \hat{x}^{(1)} & \hat{y}^{(1)} & \hat{z}^{(1)} & \hat{x}^{(2)} & \hat{y}^{(2)} & \hat{z}^{(2)} & \dots & \hat{x}^{(n_{en})} & \hat{y}^{(n_{en})} & \hat{z}^{(n_{en})} \end{bmatrix}^T, \tag{3.97}$$

the geometry of the cell is mapped from parameter to physical space by

$$\boldsymbol{x}(\xi, \eta, \zeta) = \boldsymbol{x}^h(\xi, \eta, \zeta) = \sum_{\hat{a}=1}^{n_{en}} R^{f(\hat{a},e)}_{p,q,r} \boldsymbol{P}^{f(\hat{a},e)} = \boldsymbol{N}\tilde{\boldsymbol{x}}^e \qquad \forall (\xi, \eta, \zeta) \in \hat{\Omega}_e, \tag{3.98}$$

which is very similar to the geometry mapping of eq. (3.37) in the finite element discretization – with the difference of not having to deal with an approximation of the true geometry. Reversing the isoparametric paradigm and thus using the functions defining the geometry also for the interpolation of the unknown solution field of the boundary value problem, eqs. (3.32) and (3.34) of the finite element discretization become

$$\boldsymbol{u}(\xi, \eta, \zeta) \approx \boldsymbol{u}^h(\xi, \eta, \zeta) = \sum_{\hat{a}=1}^{n_{en}} R^{f(\hat{a},e)}_{p,q,r} \boldsymbol{Q}^{f(\hat{a},e)} = \boldsymbol{N}\tilde{\boldsymbol{q}}^e \qquad \forall (\xi, \eta, \zeta) \in \hat{\Omega}_e \tag{3.99}$$

in the context of isogeometric analysis. The *control variables* $\boldsymbol{Q}^{(i,j,k)}$ store, for a displacement based finite element method, the displacement equivalents to the position vectors of the control points. As these are, with the exception of the corners, not fully interpolated by the basis functions, the values in $\boldsymbol{Q}$ do not have a direct physical meaning. The vector $\tilde{\boldsymbol{q}}^e$ contains the scalar components of all $\boldsymbol{Q}$s with support in the cell. They are the unknown degrees of freedom (DOFs). As can be seen from fig. 3.15, the control points and variables with support in a cell are geometrically not necessarily part of that cell. Furthermore, they have support in multiple cells, what becomes obvious after inspection of their associated basis functions.

In analogy to eq. (3.47), the discretized global problem reads

$$\boldsymbol{K}\tilde{\boldsymbol{q}} = \tilde{\boldsymbol{g}} \tag{3.100}$$

where $\tilde{\boldsymbol{g}}$ is the equivalent to the global load vector evaluated for location of the the control points. The individual control point load vectors contained in $\tilde{\boldsymbol{g}}$ are in general not fully interpolated by the basis functions and thus, just as the control variables $\boldsymbol{Q}^{(i,j,k)}$, they have no direct physical meaning.

Replacing the traditional polynomial shape functions discussed in sect. 3.3.2, that interpolate the element geometry and displacement field from the nodal point values, with those field interpolations defined above, which build on NURBS basis functions and their control points and control variables, establishes the concept of isogeometric analysis as a flavor of the finite

element method. Therewith, the knot spans or cells resulting from the projection of the knot vectors into physical space constitute the elements of this finite element method.[10]

3.5.3 Element matrices

The assembly of the element matrices and eventually of the global matrices in eq. (3.100) proceeds in a similar manner as was previously shown for the finite element method. However, a notable difference in the evaluation of the matrices results from the definition of the basis functions in parameter space but the numerical integration taking place in the elements natural coordinate space. When evaluating the strain-displacement matrix $\boldsymbol{B}$, the derivatives of the NURBS basis functions with respect to the physical coordinates are required. These are obtained by differentiating the basis functions with respect to the parametric coordinates and then using the inverse of the Jacobian matrix built for the relation $\partial \boldsymbol{x} / \partial \boldsymbol{\xi}$ to establish an expression for the basis functions derivatives with respect to the physical coordinates. This constitutes the same procedure as the one outlined in eq. (3.57) for the traditional finite element method, merely the physical space is not related to the natural coordinate space but to the parameter space. As however the numerical integration is still performed in the natural coordinate space, it is not sufficient to relate the differential element size between the physical and the parametric spaces. Rather, the gradient $\partial \boldsymbol{\xi} / \partial \boldsymbol{r}$ has to be evaluated as well and then both gradients are composed to form the Jacobian matrix

$$\boldsymbol{J} = \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\xi}} \frac{\partial \boldsymbol{\xi}}{\partial \boldsymbol{r}}, \tag{3.101}$$

where the determinant of $\boldsymbol{J}$ as defined in eq. (3.101) relates the differential element size in the elements natural coordinate space to the size in physical space. The relation between the natural coordinate space, the parametric space, and the physical space for a given surface element is depicted in fig. 3.16.

Regarding the numerical integration of the elements, it has to be noted, that Gauss quadrature bases on fitting a polynomial through the results of the function evaluations at the integration points. In case of isogeometric analysis, these functions are in general rational and therefore cannot be exactly reproduced with a fitted polynomial. Hence, numerical integration with Gauss quadrature is expected to be an approximation – independent of the number of integration points used. Nonetheless, it proves to work satisfactorily when the number of points is determined according to the polynomial degree of the underlying B-spline basis and the mesh is reasonably refined (cf. Hughes et al. [83]).

Furthermore it must be noted, that Gauss quadrature solely bases on the information available for an individual element. This is appropriate for Lagrangian elements with C^0 continuity across element boundaries. In the case of isogeometric analysis, continuity across element boundaries is increased to C^{p-m}. Therefore, additional information from neighboring elements can be incorporated into a patch wide integration scheme. Thus, the number of integration

[10] Naming the knot spans or its multi-dimensional equivalents "elements" results from the fact that they constitute the basis for the numerical integration. However, as the basis functions are defined over an entire NURBS patch and their support extends beyond the limits of a knot span, the entire NURBS patch is also referred to as *macro element*.

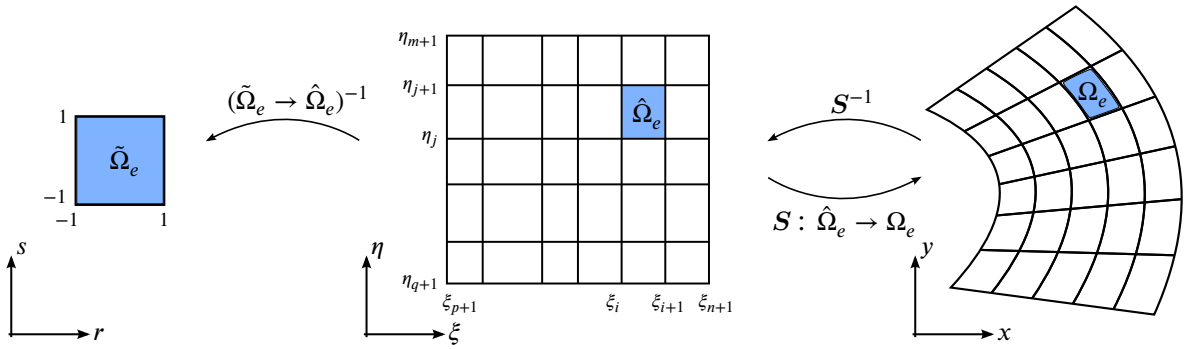


Figure 3.16: Relation between element spaces for a NURBS surface patch. The element is defined by the knot values in parametric space and transferred to physical space via the mapping $\boldsymbol{S}$. For the evaluation of the strain-displacement matrix it is pulled back from physical to parameter space and for numerical integration it is further pulled back to the elements natural coordinate space.

points per element may be reduced and in consequence, the efficiency of the numerical integration could be increased. However, as the computational effort for evaluating the element matrices is small compared with solving the global system of equations and in addition, the integration can effectively be parallelized, standard Gauss quadrature is used in this work. Further information on improved integration schemes for NURBS based isogeometric analysis is provided by Adam et al. [2], Auricchio et al. [12], and Hughes et al. [85].

3.5.4 NURBS basis derivatives

Using NURBS as a basis for structural analysis also requires the evaluation of the basis functions' first derivatives to express the kinematic relation in eq. (3.8). Efficient algorithms to evaluate the derivatives of any order for curves and surfaces are proposed by Piegl and Tiller [129]. The extension to solids is straightforward. The basic equations for first order derivatives are presented below.

The trivariate NURBS basis function given in eq. (3.89) and repeated here for convenience,

$$R^{(i,j,k)}_{p,q,r}(\xi,\eta,\zeta) = \frac{N^{(i)}_p(\xi)\, N^{(j)}_q(\eta)\, N^{(k)}_r(\zeta)\, w^{(i,j,k)}}{\sum_{\hat{i}=1}^{n}\sum_{\hat{j}=1}^{m}\sum_{\hat{k}=1}^{l} N^{(\hat{i})}_p(\xi)\, N^{(\hat{j})}_q(\eta)\, N^{(\hat{k})}_r(\zeta)\, w^{(\hat{i},\hat{j},\hat{k})}}$$

can be expressed as

$$R^{(a)}(\boldsymbol{\xi}) = \frac{A^{(a)}(\boldsymbol{\xi})}{W(\boldsymbol{\xi})}. \tag{3.102}$$

The expressions for the numerator and denominator respectively, are the same in the upper and lower relation. For reasons of clarity, the polynomial degrees were dropped in the lower relation, the patch global numbering scheme was applied, and the three parametric coordinates were expressed by vector notation. Differentiating this expression with the help of the quotient rule results in

$$R^{(a)}_{,\alpha}(\boldsymbol{\xi}) = \frac{A^{(a)}_{,\alpha}(\boldsymbol{\xi})\, W(\boldsymbol{\xi}) - A^{(a)}(\boldsymbol{\xi})\, W_{,\alpha}(\boldsymbol{\xi})}{W(\boldsymbol{\xi})^2} \tag{3.103}$$

where , α denotes the partial differentiation with respect to the parametric coordinate expressed by α. Reinserting eq. (3.102) leads to

$$R^{(a)}_{,\alpha}(\boldsymbol{\xi}) = \frac{A^{(a)}_{,\alpha}(\boldsymbol{\xi}) - R^{(a)}(\boldsymbol{\xi})\, W_{,\alpha}(\boldsymbol{\xi})}{W(\boldsymbol{\xi})}. \tag{3.104}$$

Both terms that require differentiation constitute the (sum of the) product of the multivariate B-spline basis function $N^{(a)}$ and the homogeneous coordinate $w^{(a)}$, with the latter being constant with respect to α. Since the multivariate function is a product of three univariate B-spline basis functions and two of the functions are constant with respect to α as well, the third is readily differentiated with

$$N^{(i)}_{p\,,\alpha}(\alpha) = \frac{p}{\alpha_{i+p} - \alpha_i} N^{(i)}_{p-1}(\alpha) - \frac{p}{\alpha_{i+p+1} - \alpha_{i+1}} N^{(i+1)}_{p-1}(\alpha), \tag{3.105}$$

which is the first derivative of the univariate B-spline basis function in eq. (3.76). Hence, when exemplifying eq. (3.104) by setting $\alpha = \xi$, then $A^{(a)}_{,\alpha}(\boldsymbol{\xi})$ and $W_{,\alpha}(\boldsymbol{\xi})$ become

$$A^{(i,j,k)}_{,\xi}(\boldsymbol{\xi}) = N^{(i)}_{p\,,\xi}(\xi)\, N^{(j)}_{q}(\eta)\, N^{(k)}_{r}(\zeta)\, w^{(i,j,k)} \tag{3.106}$$

and

$$W_{,\xi}(\boldsymbol{\xi}) = \sum_{i=1}^{n} \sum_{j=1}^{m} \sum_{k=1}^{l} N^{(i)}_{p\,,\xi}(\xi)\, N^{(j)}_{q}(\eta)\, N^{(k)}_{r}(\zeta)\, w^{(i,j,k)} = \sum_{a=1}^{nml} A^{(a)}_{,\xi}(\boldsymbol{\xi}) \tag{3.107}$$

and thus, all terms of the partial derivative $R^{(a)}_{,\alpha}$ can be evaluated. The effort to do so for all quadrature points of a volumetric domain is clearly increased, when compared with the rather simple derivatives of the Lagrange polynomials.

3.5.5 Refinement strategies

The isogeometric notion of an element results from the projection of the knot vectors into physical space. Increasing the number of knots in such a vector clearly increases the number of finite elements and thereby reduces their size in physical space. The corresponding operation of knot insertion is presented in sect. 3.4.5 as a means to gain refined control over a given curve. The actual result of this operation is an enriched basis that allows a more specific manipulation of the curve. Due to this property, knot insertion is the isogeometric equivalent of *h-refinement*. Typically, a new knot is inserted with a distinct value into an existing knot span. The respective element is thereby split into two new elements with a continuity of C^{p-1} at the new interface. Inserting an additional knot with an already existing value does not split the element but reduces the continuity of the already existing element interface. Raising the multiplicity of a knot further until it equals $p + 1$ splits the NURBS patch into two.

Owing to the tensor product structure, any univariate parametric coordinate represents a hyperplane of a multivariate NURBS patch. Inserting this coordinate as a new knot consequently splits all elements along that hyperplane. This effect, which is depicted in fig. 3.17, constitutes a fundamental disadvantage of the tensor product structure, as it results in an increased computational demand when solving the discretized boundary value problem. The severity of that disadvantage grows with the spatial dimension of the NURBS patch.

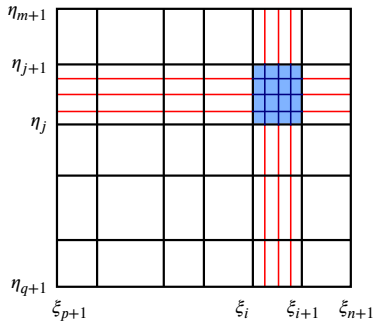


Figure 3.17: Parametric representation of a NURBS surface. Refinement of element $[\xi_i, \xi_{i+1}] \times [\eta_j, \eta_{j+1}]$ propagates through the entire NURBS patch causing numerous extra elements that may be undesired as they entail additional computational effort.

The second operation discussed in sect. 3.4.5 is degree elevation, which also enriches the basis but does not produce any new elements. As the enrichment results from raising the polynomial order of the underlying B-spline basis, this operation is the equivalent of *p-refinement* in the classical finite element method. The enrichment does not happen elementwise but is patch global. Raising the degree by one in a single parametric direction produces one additional basis function for that direction. However, as in the case of h-refinement, the tensor product structure causes the total number of additional basis functions to be significant for multivariate patches. Yet, there is also a positive aspect to the tensor product structure of multivariate patches: It is possible to selectively raise the order for a single parametric direction, a direction which frequently corresponds to a distinct physical extent of the body under analysis.

Combining the aforementioned operations gives rise to a third strategy that was named *k-refinement* by Hughes et al. [83]. Since knot insertion and degree elevation are not commutative, it is important to firstly raise the degree of the basis and only then perform h-refinement. This way, the order of continuity at any new element boundary that is created during the h-refinement process is equal to the original degree p of the basis, whereas it would be $p - 1$ when the order of refinement is reversed. Also, the total number of basis functions is considerably smaller for k-refinement than it is for its reversed counterpart.

Cottrell et al. provide a thorough study of refinement methods for NURBS-based isogeometric analysis in [42]. In [84], Hughes et al. compare p-refinement of C^0 continuous finite element meshes with the k-refinement method of isogeometric analysis for problems in structural dynamics.

3.5.6 Conclusion

In a variety of publications NURBS-based isogeometric analysis was shown to be successfully applied to problems of structural mechanics. Sound convergence rates for two and three dimensional continuum element formulations using NURBS of different order were already shown in the initial paper on the topic [83]. In [42], Cottrell et al. study the effects that result from the smoothness of the NURBS basis at element boundaries. A comparison of convergence rates for NURBS and Lagrange polynomials under h-refinement in a structural mechanics setting is conducted by Echter and Bischoff [60]. Advanced structural shell formulations based on the sophisticated NURBS basis are proposed by Benson et al. for large deformations within

the Reissner-Mindlin shell theory [26, 27] and by Kiendl et al. for the Kirchhoff-Love shell [91, 92]. Also in other fields of engineering, the concept of using NURBS as a basis for analysis led to an extensive research activity, e.g. in fluid dynamics and fluid-structure interaction [18, 19, 21, 22], vibrations and wave propagation [41, 42, 85, 168], shape and topology optimization [93, 151, 163], and electromagnetics [35].

For many of these examples, the NURBS based formulations proved to deliver superior results in terms of accuracy versus degrees of freedom over their Lagrangian counterparts. This is mainly credited to the smoothness and increased continuity of the NURBS basis, which may avoid jumps and kinks in the fields of the derivatives of the solution variable. NURBS based discretizations also show a higher robustness toward mesh distortion [107]. However, two rather fundamental drawbacks exist when compared with the classic element formulations.

Combining several NURBS patches to analyze more complex shapes than can be defined with a single patch is a rather involved task. One possible approach to do so is to glue multiple patches by imposing weakly enforced interface constraints on their shared boundaries. Thus, a common analysis domain is formed out of the individual patches. This topic is covered in chapter 4 and is therefore not further discussed here. A different approach is taken with the fictitious domain methods [106, 116], which embed the domain of the actual problem in a larger, fictitious domain of typically very simple geometry. The easily discretized fictitious domain then provides the basis for a numerical solution. Embedding multiple "connected" patches within such a domain also allows to compute the numerical solution for problems with a complex, multi-patch geometry. In that sense, the finite cell method [56, 127] was recently extended to the context of isogeometric analysis [133, 146, 166]. Furhtermore, it is shown in [98, 137, 138] that the finite cell method is also a possible approach to deal with trimmed patches that are frequently created in standard CAD applications.

Another previously mentioned drawback is the tensor product structure of NURBS, which prohibits local refinement. When modeling the geometry in CAD, this is only inconvenient: unnecessary control points must be introduced in order to express a local feature. For an analysis, local mesh refinement in regions of special interest or large error is a frequently used procedure. In this case though, the addition of unnecessary control points is more than an inconvenience. Due to their sheer number or rather the number of their associated degrees of freedom, they can have a major influence on the computational resources required to solve the problem.

Though the patch coupling technique discussed in chapter 4 provides a technique to prevent the propagation of mesh refinement through the entire domain, the topic of local refinement is beyond the scope of this work. Nonetheless, this is an ongoing research topic with promising results to which some references shall be made. T-splines introduced by Sederberg et al. [150] are a generalization of NURBS and therefore also base on a grid structure. Yet, for T-splines the grid may be incomplete rendering local refinement possible. Many publications on their use in isogeometric analysis exist, see e.g. [20, 24, 45, 53, 149, 161] and the references therein. Polynomial splines over hierarchical T-meshes (PHT-splines) [50] and their rational counterpart RHT-splines [165] also allow for local refinement – based on a considerably simpler algorithm than T-splines do. The refinement of (R)PHT-splines always starts from an underlying NURBS patch but the continuity on element boundaries is restricted to C^1. Examples of the application of (R)PHT-splines in an analysis context are found in [118, 119, 120, 165]. Further concepts motivated by the lack of local refinement in NURBS are the polynomial splines over locally refined box-partitions [52] and hierarchical B-splines [99, 145, 146] that are possibly truncated

and then named THB-splines [34, 70, 95]. In Zander et al. [170], a similar hierarchical refinement concept is formulated for 3D problems that are solved with the *hp*-version of the finite element method. Yet another refinement variant is provided with the locally refined B-splines (LR B-splines) by Johannessen et al. [88, 89].

Chapter 4

Multiple patches and domain coupling

4.1 Introduction

Examples of physical bodies represented by NURBS objects were shown in figs. 3.14 and 3.15. In both cases, the body consists of a single NURBS patch, whose shape is restricted to the mapping of a rectangular cuboid in 3D parameter space to physical space. For many practical situations, the shape limitation of single patch would be too restrictive to express the desired geometry and thus multiple patches have to be used. In this case, each individual patch denotes a subdomain $\Omega^{(i)}$ of the body and the complete body is made up by the union of all n_{sub} subdomains. Accordingly, the notation of eq. (3.92) has to be extended to

$$\Omega = \Omega^h = \bigcup_{i=1}^{n_{sub}} \Omega^{(i)}, \tag{4.1}$$

where each subdomain

$$\Omega^{(i)} = \bigcup \Omega_e^{(i)}. \tag{4.2}$$

While the use of multiple patches is a standard procedure within CAD, it can be a rather demanding task in a numerical analysis setting. Coupling conditions have to be specified in order to unite the various subdomains in a single analysis. The coupling discussed in this work assumes that two adjacent subdomains share a common boundary in physical space and that each subdomain is part of at least one such boundary. Thus, the union of all patches constitutes a set of interconnected subdomains. For the neighboring subdomains i and j, the shared boundary is denoted $\Gamma_c^{(i,j)}$, where the index (i, j) is dropped whenever the associated subdomains are apparent from the context. Likewise to the finite element domains (cf. eq. (3.30)), it is required that

$$\Omega^{(i)} \cap \Omega^{(j)} = \emptyset \qquad \forall i, j \in 1 \dots n_{sub} \quad \text{with} \quad i \neq j. \tag{4.3}$$

Equations (4.1) and (4.3) require the geometry model to be watertight, i.e. without any gaps or overlaps between connected patches. In real world situations however, adjacent patches created with commercial CAD applications often show (small) gaps and overlaps on their common boundary [39]. For the purpose of statically displaying the geometry, this is irrelevant as long as a critical magnification level is not exceeded. For more involved applications, e.g. deforming

patches in animations or, precisely, structural analysis, it is an issue.[1] One approach of resolving this problem is the use of global watertight parametrizations that are possible with T-splines [20]. They come at the price of losing the regular structure of NURBS and require to actually connect the individual geometric objects to a single global patch. For different materials, a global patch is not likely to be the optimal solution. And, as will be seen in chapter 5, multiple patches also have advantages for parallelization on multi-CPU computers. Within AECO, commonly used shapes are very regular. With the renouncement of trimmed patches and the awareness of a subsequent analysis during geometric modeling, watertightness is also achievable with NURBS. For the topic of analysis-aware modeling, the reader is once again referred to the work of Cohen et al. [39]. In this work, the subject of inaccurate geometry representations is addressed by loosening the strict requirements on watertightness with the definition of a threshold value R_{Γ_c}. When the corresponding mappings from parameter to physical space for the two subdomains i and j are

$$\boldsymbol{x}^{(i)} = \boldsymbol{V}(\boldsymbol{\xi}) : \hat{\Omega}^{(i)} \rightarrow \Omega^{(i)} \quad \text{and} \quad \boldsymbol{x}^{(j)} = \boldsymbol{V}(\boldsymbol{\xi}) : \hat{\Omega}^{(j)} \rightarrow \Omega^{(j)} \tag{4.4}$$

then contact of the two subdomains requires that

$$\left\| \boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)} \right\|_2 \leq R_{\Gamma_c} \qquad \forall \boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)} \in \Gamma_c^{(i,j)}. \tag{4.5}$$

Thus, a boundary of adjacent patches is considered a shared boundary, when their distance at any point is less or equal to R_{Γ_c}.

In numerical analysis, domain decomposition methods are used to divide the analysis domain into smaller parts that can be solved independently and in parallel. The methods must ensure that the results on the shared boundaries of the respective subdomains correspond to each other. Bridging domain methods represent the reverse situation: Different subdomains with usually non-conforming meshes are to be employed in single, interconnected analysis. Traditionally, these situations arise in multi-scale problems or when different physical problems are to be coupled. Domain decomposition methods and bridging domain methods base on the same principles and therefore are closely related. Previously it was shown that the finite element discretization of NURBS is inherent to the patches and thus cannot be freely chosen. In consequence, the mesh of NURBS patches that are to be coupled is usually non-conforming and one method out of the afore mentioned group of methods has to be applied.

4.2 Domain coupling methods

Conditions for the coupling of subdomains can be specified in strong or in weak form. Strong coupling requires the difference in the primal field variables of the coupled subdomains, i.e. the displacements for the displacement-based FEM, to vanish at any point on the shared boundary.

$$\boldsymbol{u}^{(i)}(\boldsymbol{x}) - \boldsymbol{u}^{(j)}(\boldsymbol{x}) = \boldsymbol{0} \qquad \forall \boldsymbol{x} \in \Gamma_c \tag{4.6}$$

[1] This is not only relevant in the context of IGA but in an equivalent manner also for traditional finite element analyses. Geometric representations created with common CAD systems often show geometric inaccuracies that are frequently the cause of failure during automated mesh generation or of undesired mesh topologies.

This restriction is eased in the case of weak coupling, where condition (4.6) must be fulfilled only in an integral sense such that

$$\int_{\Gamma_c} \boldsymbol{u}^{(i)}(\boldsymbol{x}) - \boldsymbol{u}^{(j)}(\boldsymbol{x})\, d\boldsymbol{x} = \boldsymbol{0} \qquad \forall \boldsymbol{x} \in \Gamma_c. \tag{4.7}$$

The implication of reducing the requirements for the coupling conditions is the loss of displacement compatibility. With condition (4.7), this is only achievable on average over Γ_c.

Strong coupling is achieved with constraint equations. The control variables associated with patch displacement can be distinguished by whether or not they have support on a shared boundary. This is denoted by subscript s for the variables with support and subscript n for all others. The control variables $\tilde{\boldsymbol{q}}$ of two adjacent patches i and j are thus

$$\tilde{\boldsymbol{q}}^{(i)} = \begin{bmatrix} \tilde{\boldsymbol{q}}_n^{(i)} \\ \tilde{\boldsymbol{q}}_s^{(i)} \end{bmatrix} \quad \text{and} \quad \tilde{\boldsymbol{q}}^{(j)} = \begin{bmatrix} \tilde{\boldsymbol{q}}_n^{(j)} \\ \tilde{\boldsymbol{q}}_s^{(j)} \end{bmatrix}. \tag{4.8}$$

Then,

$$\tilde{\boldsymbol{q}}_s^{(i)} = f\left(\tilde{\boldsymbol{q}}_s^{(j)}\right) \tag{4.9}$$

is a general formulation of constraint equations for the two patches that are to be coupled. It is used to eliminate the control variables $\tilde{\boldsymbol{q}}_s^{(i)}$ from the global problem given in eq. (3.100). Thereby, the global problem size is reduced by the number of constrained DOFs. A procedure for the determination of function f in eq. (4.9) and the subsequent assembly of the reduced global system of equations is outlined in the monograph by Cottrell et al. [43] and discussed in Kleiss et al. [97]. It ensures C^0 continuity across subdomain boundaries. However, the procedure has a severe drawback that renders it almost useless for the intended use in a framework that requires the flexible coupling of patches, which represent structural components: The coupling interface of any two patches must not only match geometrically but also the parametrization, the polynomial degree and the control points of the shared boundary must be identical on the coarsest refinement level and the refinement history must be known [43]. Otherwise, f cannot be determined.

As constraint equations cannot be established for patches of arbitrary parametrizations, weak coupling has to be employed. Thus, displacement incompatibilities at the coupling interfaces have to be accepted. In recent years different methods have been investigated for the purpose of weakly coupling NURBS patches in the context of IGA. Namely, there are the Nitsche method, the mortar method with Lagrange multipliers (LMs), and recently, the weak substitution method. Nitsche's method is discussed in [9, 117, 138]. It was originally proposed to impose Dirichlet boundary conditions in weak form. This is achieved by adding terms to the variational formulation (eq. (3.19)) that express the flux on the constrained boundary or the coupling interfaces respectively in terms of the field of primal unknowns. Furthermore, stabilizing terms are added to the formulation that ensure the positive definiteness of the resulting stiffness matrix. The final system of equations has the same number of unknowns as the originally uncoupled problem. The weak substitution method was proposed by Dornisch et al. [54]. The method makes use of a weak ansatz to express the primal variables of the slave side of the shared subdomain boundary with those of the master side. Equivalent to the constraint equations, the primal variables of the slave side are subsequently condensed out of the global

system of equations. Results are reported to be comparable to those of the Nitsche and the mortar method. A comparison of weak coupling methods in the context of IGA is given by Apostolatos et al. [9].

The integrated structural analysis framework that is developed within this work uses the mortar method for the patch coupling. Reasons and details of the method are explained in the following section.

4.3 The mortar method

The mortar method was first published by Maday et al. [111] as a domain decomposition approach for non-conforming meshes in the setting of the spectral element method. Compatibility along the coupling interface was originally achieved with a reduced space of shape functions [30, 111]. Later a variant was introduced, that uses a field of Lagrange multipliers for that purpose, cf. [25, 169]. In this case, the Lagrange multiplier field represents an approximation to the normal derivatives of the primal unknowns on the interface.

The mortar method has been successfully applied to solid mechanics problems. Arbitrarily meshed 3D subdomains were coupled and solved with the classic FEM by Puso [132]. Hesch and Betsch [78] used it in the context of IGA by coupling non-conforming patches. And finally, Temizer et al. [157] as well as De Lorenzis et al. [48] applied it to contact problems in IGA.

Applying the standard form of the mortar method to 3D solid mechanical problems leads to a rigid coupling of the involved subdomains. It is noted that this property is not always desirable, e.g. in the case of hinged connections. An approach to circumvent this limitation is demonstrated by Horger et al. [79] who implicitely incorporate an elastic bearing in the mortar formulation in the context of a modal analysis with the *hp*-version of the finite element method.

The mortar method with reduced function spaces leads to a positive definite problem formulation, as do the Nitsche method and the weak substitution method. This is not the case for the mortar method with Lagrange multipliers. It yields a saddle point problem, which is more problematic to solve, but it retains the block diagonal structure of the uncoupled patches and therefore provides attractive parallelization options. The augmented Lagrangian approach also leads to a saddle point problem but in contrast to the non-augmented variant additionally introduces coupling terms for the patches that are in contact. This feature renders the parallelization not as straightforward. Since the integrated structural analysis framework deals with volumetric objects, the resulting system of equations has the potential to become very large and thus, parallelization is a necessity. Therefore, the approach of the mortar method with Lagrange multipliers is pursued in this work.

Extension of the continuum mechanics problem

In the setting of multiple subdomains, the virtual work equation (3.19) reads as

$$G = \sum_{i}^{n_{sub}} G_{int}^{(i)} - G_{ext}^{(i)} = 0 \tag{4.10}$$

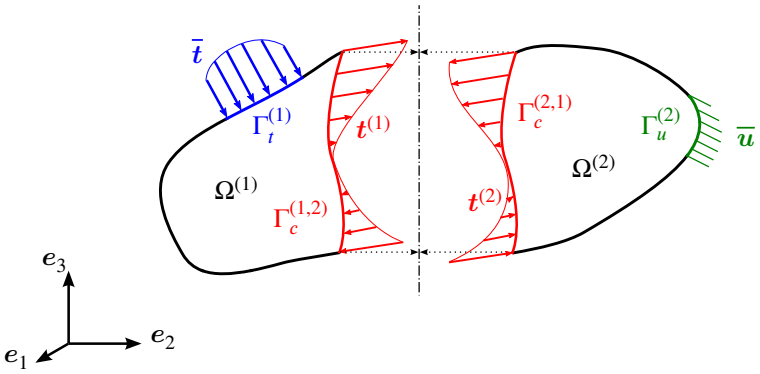


Figure 4.1: Decomposition of the body $\mathcal{B}$ into two subdomains $\Omega^{(1)}$ and $\Omega^{(2)}$. The action of one subdomain onto the other is expressed by the field of interface or coupling tractions $\boldsymbol{t}^{(1)}$ or $\boldsymbol{t}^{(2)}$ respectively.

with

$$G_{int}^{(i)} = \int_{\Omega^{(i)}} \epsilon(\delta \boldsymbol{u}) : \boldsymbol{\sigma}(\boldsymbol{u}) \, \mathrm{d}\Omega^{(i)} \tag{4.11}$$

$$G_{ext}^{(i)} = \int_{\Omega^{(i)}} \delta \boldsymbol{u} \cdot \bar{\boldsymbol{b}} \, \mathrm{d}\Omega^{(i)} + \int_{\Gamma_t^{(i)}} \delta \boldsymbol{u} \cdot \bar{\boldsymbol{t}} \, \mathrm{d}\Gamma_t^{(i)} \tag{4.12}$$

The mechanical action of one subdomain onto another at their common boundary can be expressed by tractions. As no additional external forces are to be introduced at these interfaces, the tractions at any point of such an interface must be equal in size and opposed in direction, thus

$$\boldsymbol{t}^{(i)}(P) = -\boldsymbol{t}^{(j)}(P) \qquad \forall P \in \Gamma_c^{(i,j)}. \tag{4.13}$$

The virtual work performed by these tractions on the coupling interfaces is then given by

$$G_c^{(i)} = \int_{\Gamma_c^{(i)}} \boldsymbol{t} \cdot \delta \boldsymbol{u} \, \mathrm{d}\Gamma_c^{(i)}, \tag{4.14}$$

a term that is essential for the formulation of the mortar method. It is thus added to the virtual work balance in eq. (4.10), which then results in

$$G = \sum_i^{n_{sub}} G_{int}^{(i)} - G_{ext}^{(i)} - G_c^{(i)} = 0. \tag{4.15}$$

Denoting any two subdomains with a shared boundary $\Omega^{(i)}$ and $\Omega^{(j)}$ and considering their virtual work contributions at once, allows the formulation (4.14) also to be expressed as

$$G_c^{(i)} + G_c^{(j)} = \int_{\Gamma_c} \boldsymbol{\lambda} \left(\delta \boldsymbol{u}^{(i)} - \delta \boldsymbol{u}^{(j)} \right) \mathrm{d}\Gamma_c, \tag{4.16}$$

where

$$\boldsymbol{\lambda} = \boldsymbol{t}^{(i)} = -\boldsymbol{t}^{(j)}. \tag{4.17}$$

In addition to the modification of the virtual work equation, weak constraints have to be imposed on the shared boundary of any two adjacent subdomains, i.e.

$$\int_{\Gamma_c} \delta\boldsymbol{\lambda} \left(\boldsymbol{u}^{(i)} - \boldsymbol{u}^{(j)}\right) \mathrm{d}\Gamma_c = 0 \qquad \forall i, j \in 1 \dots n_{sub} \quad \text{with} \quad i \neq j, \tag{4.18}$$

where the variation of $\boldsymbol{\lambda}$ is used as test functions thus rendering this formulation as the conjugate to eq. (4.16).

With the afore stated extensions, the global boundary value problem in weak form now comprises the modified virtual work principle in eq. (4.15), the original boundary conditions in eqs. (3.20) and (3.21) as well as the new boundary conditions given in eq. (4.18). In addition to the existing unknown displacement field $\boldsymbol{u}$, a second unknown field variable, the field of LMs $\boldsymbol{\lambda}$ representing the interface tractions at the shared subdomain boundaries was introduced. The BVP must be fulfilled for arbitrary virtual states $\delta\boldsymbol{u}$ and $\delta\boldsymbol{\lambda}$. It is solved with the finite element method.

Discretization

The approximations of the displacement fields of the individual subdomains takes place as outlined in sects. 3.3.2 and 3.5.2 for the entire domain. Equation (3.99) denotes the displacement interpolation associated with a single finite element. Using that notation and storing all basis functions associated with a subdomain i in a single matrix $\boldsymbol{N}^{(i)}$ and the corresponding unknown degrees of freedom in the vector $\tilde{\boldsymbol{q}}^{(i)}$, the displacement field of that subdomain and likewise its virtual counterpart are written as

$$\begin{aligned} \boldsymbol{u}^{(i),h}(\boldsymbol{\xi}) &= \boldsymbol{N}^{(i)}(\boldsymbol{\xi})\, \tilde{\boldsymbol{q}}^{(i)} \\ \delta\boldsymbol{u}^{(i),h}(\boldsymbol{\xi}) &= \boldsymbol{N}^{(i)}(\boldsymbol{\xi})\, \delta\tilde{\boldsymbol{q}}^{(i)} \end{aligned} \qquad \forall \boldsymbol{\xi} \in \hat{\Omega}^{(i)}. \tag{4.19}$$

In an equal manner to the displacements, the field of LMs on any coupling interface is approximated by

$$\boldsymbol{\lambda}^{(i,j),h}(\boldsymbol{P}) = \boldsymbol{N}^{c,(i,j)}(\boldsymbol{P})\, \tilde{\boldsymbol{\lambda}}^{c,(i,j)} \qquad \forall \boldsymbol{P} \in \Gamma_c^{(i,j)}, \tag{4.20}$$

where $\boldsymbol{\lambda}^{(i,j),h}$ is the approximated field of LMs on the interface $\Gamma_c^{(i,j)}$, $\tilde{\boldsymbol{\lambda}}^{c,(i,j)}$ is the vector of discrete LM values for that interface and $\boldsymbol{N}^{c,(i,j)}$ are the associated interpolation functions. The corresponding virtual field is likewise defined as

$$\delta\boldsymbol{\lambda}^{(i,j),h}(\boldsymbol{P}) = \boldsymbol{N}^{c,(i,j)}(\boldsymbol{P})\, \delta\tilde{\boldsymbol{\lambda}}^{c,(i,j)} \qquad \forall \boldsymbol{P} \in \Gamma_c^{(i,j)}. \tag{4.21}$$

The interpolations functions are not interface global, but in the manner of the finite element method local to some kind of interface segmentation. In order to make a distinction from the surface elements of the coupled patches, these areas of local interpolation are referred to as

mortar segments. The discrete LM values are then associated with support points on these segments. $\boldsymbol{N}^{c,(i,j)}$ and $\tilde{\boldsymbol{\lambda}}^{(i,j)}$ constitute the collection of all segment contributions, i.e.

$$\boldsymbol{N}^{c,(i,j)} = \sum_{seg} \boldsymbol{N}^{c,(i,j)}_{seg} \qquad\qquad \tilde{\boldsymbol{\lambda}}^{c,(i,j)} = \sum_{seg} \tilde{\boldsymbol{\lambda}}^{c,(i,j)}_{seg} \tag{4.22}$$

where it is again implied, that matrix sizes match the total of the number of support points with entries not having support in a given segment being considered zero. There are several options for the definition of the segments, interpolation functions and support point locations that are discussed in sect. 4.4.1. At this point, explaining the method is continued without a detailed definition.

Introducing eqs. (4.19) and (4.20) into the virtual work statement for the coupling interface in eq. (4.16) leads to

$$\boldsymbol{G}_c^{(i,j)} = \int_{\Gamma_c^{(i,j)}} \boldsymbol{N}^{c,(i,j)}\, \tilde{\boldsymbol{\lambda}}^{c,(i,j)} \left(\boldsymbol{N}^{(i)}\, \delta\tilde{\boldsymbol{q}}^{(i)} - \boldsymbol{N}^{(j)}\, \delta\tilde{\boldsymbol{q}}^{(j)} \right) \mathrm{d}\Gamma_c^{(i,j)}, \tag{4.23}$$

which after reordering and omitting the superscript (i, j) denoting the respective interface becomes

$$\boldsymbol{G}_c = \left(\int_{\Gamma_c} \boldsymbol{N}^{(i)^T} \boldsymbol{N}^c\, \tilde{\boldsymbol{\lambda}}^c \, \mathrm{d}\Gamma_c \right)^T \delta\tilde{\boldsymbol{q}}^{(i)} - \left(\int_{\Gamma_c} \boldsymbol{N}^{(j)^T}\, \boldsymbol{N}^c\, \tilde{\boldsymbol{\lambda}}^c \, \mathrm{d}\Gamma_c \right)^T \delta\tilde{\boldsymbol{q}}^{(j)}. \tag{4.24}$$

For arbitrary virtual displacement fields, the entire virtual work equation (4.15) can only evaluate to zero, when that equation becomes zero with only the parenthesized terms in the previous expression being considered for the interface virtual work contribution.

Introducing eqs. (4.19) and (4.21) into the weak constraints formulation in eq. (4.18), reordering and omitting indices leads to the similar relation

$$0 = \left(\int_{\Gamma_c} \boldsymbol{N}^{c^T} \boldsymbol{N}^{(i)}\, \tilde{\boldsymbol{q}}^{(i)} \mathrm{d}\Gamma_c \right)^T \delta\tilde{\boldsymbol{\lambda}}^c - \left(\int_{\Gamma_c} \boldsymbol{N}^{c^T} \boldsymbol{N}^{(j)}\, \tilde{\boldsymbol{q}}^{(j)} \mathrm{d}\Gamma_c \right)^T \delta\tilde{\boldsymbol{\lambda}}^c, \tag{4.25}$$

which again can only hold for an arbitrary virtual Lagrange multiplier field, when the sum of the parenthesized terms becomes zero.

The integration in eqs. (4.24) and (4.25) is independent of the discrete $\tilde{\boldsymbol{\lambda}}^c$, $\tilde{\boldsymbol{q}}^{(i)}$ and $\tilde{\boldsymbol{q}}^{(j)}$ values and thus, mortar matrices can be defined as

$$\boldsymbol{m}^{(i)} = \int_{\Gamma_c} \boldsymbol{N}^{c^T} \boldsymbol{N}^{(i)} \, \mathrm{d}\Gamma_c \qquad\qquad \boldsymbol{m}^{(j)} = -\int_{\Gamma_c} \boldsymbol{N}^{c^T} \boldsymbol{N}^{(j)} \, \mathrm{d}\Gamma_c, \tag{4.26}$$

which allows to write the virtual work contribution for a given coupling interface as

$$\boldsymbol{G}_c = \left(\boldsymbol{m}^{(i)^T} \tilde{\boldsymbol{\lambda}}^c \right)^T \delta\tilde{\boldsymbol{q}}^{(i)} + \left(\boldsymbol{m}^{(j)^T} \tilde{\boldsymbol{\lambda}}^c \right)^T \delta\tilde{\boldsymbol{q}}^{(j)} \tag{4.27}$$

and the corresponding constraints condition as

$$0 = \left(\boldsymbol{m}^{(i)}\tilde{\boldsymbol{q}}^{(i)}\right)^T \delta\tilde{\boldsymbol{\lambda}}^c + \left(\boldsymbol{m}^{(j)}\tilde{\boldsymbol{q}}^{(j)}\right)^T \delta\tilde{\boldsymbol{\lambda}}^c. \tag{4.28}$$

The system matrices and vectors associated with the internal and external work contributions remain unchanged. The procedure for their establishment was previously presented in chapter 3. Assuming the problem domain to consist of the two subdomains i and j, the uncoupled, subdomain-specific systems of equations read as

$$\boldsymbol{K}^{(i)}\,\tilde{\boldsymbol{q}}^{(i)} = \tilde{\boldsymbol{g}}^{(i)} \qquad \text{and} \qquad \boldsymbol{K}^{(j)}\,\tilde{\boldsymbol{q}}^{(j)} = \tilde{\boldsymbol{g}}^{(j)}. \tag{4.29}$$

Provided that the two subdomains share a common boundary, the two subsystems can be coupled with the statements given in eqs. (4.27) and (4.28). For the entire domain, the global system of equations then expands to

$$\begin{bmatrix} \boldsymbol{K}^{(i)} & \boldsymbol{0} & \boldsymbol{m}^{(i)^T} \\ \boldsymbol{0} & \boldsymbol{K}^{(j)} & \boldsymbol{m}^{(j)^T} \\ \boldsymbol{m}^{(i)} & \boldsymbol{m}^{(j)} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{q}}^{(i)} \\ \tilde{\boldsymbol{q}}^{(j)} \\ \tilde{\boldsymbol{\lambda}}^{c,(i,j)} \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{g}}^{(i)} \\ \tilde{\boldsymbol{g}}^{(j)} \\ \boldsymbol{0} \end{bmatrix}, \tag{4.30}$$

which is the saddle point problem that needs to be solved in order to obtain the displacement fields of the coupled subdomains. An additional result obtained with the solution is the field of Lagrange multipliers, which, unless the coupling tractions are of special interest, is not used any further.

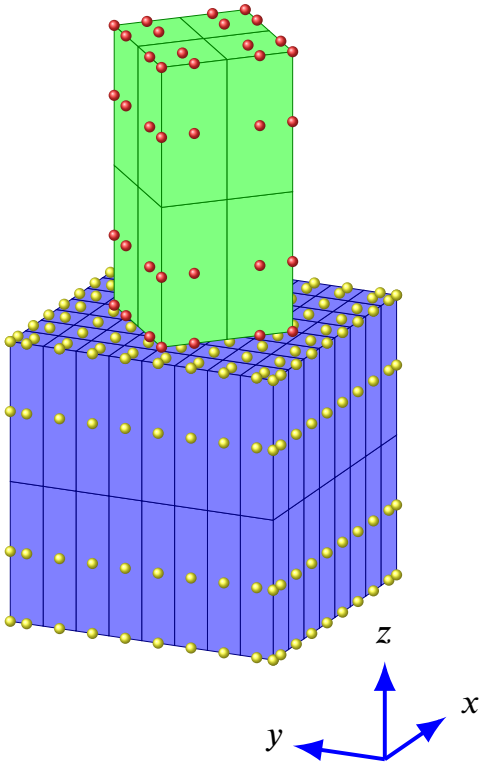
4.4 Prerequisites and implementation details

4.4.1 Lagrange multiplier interpolation

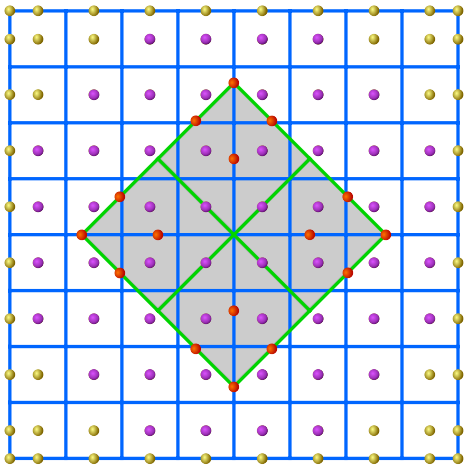
The details of the Lagrange multiplier interpolation were skipped in the previous sect. 4.3, as they require special attention. From the definition of the mortar matrices in eq. (4.26) it is obvious, that the two sides of a coupling interface stemming from two different subdomains are treated differently and therefore must be uniquely identified as master and slave or as mortar and non-mortar side respectively throughout an analysis. In the case of solid NURBS both interface surfaces are discretized from their respective underlying NURBS parametrizations.[2] It is convenient to use one of the two discretizations also for the Lagrange multipliers. By convention, the side chosen as the source of the interface discretization is labeled non-mortar side.

The NURBS parametrization of a surface always yields rectangular elements in parameter space. However, as a result of the mapping in eq. (3.86), the element edges may be curved

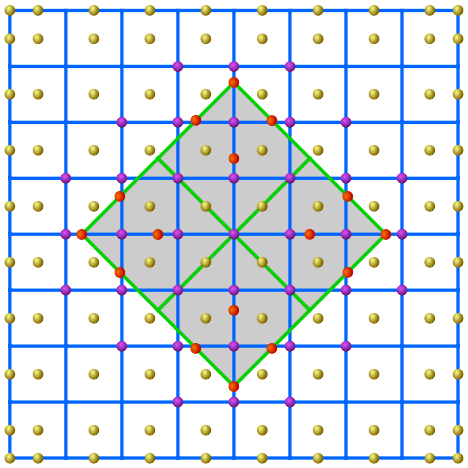
[2] In sect. 3.4.6 it is shown, how a boundary surface discretization can be recovered from the original volume representation of the given subdomain. Of course, this also applies to the coupling surfaces of any two adjacent subdomains.



(a) Coupling problem



(b) Biquadratic NURBS



(c) Bilinear Lagrangian

Figure 4.2: Mortar coupling of two NURBS patches. The coupling problem and patch parametrization is depicted in (a), all basis functions are of quadratic type. The lower patch is selected as non-mortar side, thus its parametrization is used for the interpolation of the Lagrange multiplier field. (b) displays the interface parametrization for the case that the Lagrange multiplier field is interpolated by the inherited NURBS surface basis functions of the non-mortar side, (c) depicts the interpolation with bilinear Lagrangian functions. LM support point locations are denoted by magenta nodes, lower patch control points by yellow nodes and upper patch control points by red nodes. In (b) LM nodes and lower patch control points coincide.

in physical space. They are expressed by rational functions whose underlying polynomial degree is the degree of the solid in the respective parametric direction. Hesch and Betsch [78] use the mapped element corners in physical space as support points for the interpolation of the Lagrange multiplier field. The interpolation is done with element local bilinear Lagrangian shape functions. With this approach, the mortar segments for the interpolation of the Lagrange multiplier field do not coincide with the surface elements of the non-mortar side for the case of curved element edges. Hesch and Betschs reason to use the bilinear interpolation scheme nonetheless, is to avoid overconstrained DOFs on the coupling interface, which might result from the increased number of support points for the higher degree interpolation. In this work,

this and a second, alternative approach are implemented. The latter makes use of the bivariate NURBS basis functions of the non-mortar side for the interpolation of the Lagrange multiplier field. Thus, the field of Lagrangian multipliers, i.e. the field of interface stress vectors, is of higher interpolation quality than the general stress state of the domain that is derived from the displacement field. Interface tractions are interpolated with basis functions of degree $p \times q$ whereas the general stress field only has degree $(p-1) \times (q-1)$. In the original approach, the interface tractions are of degree 1×1. For the various numerical examples, the implications of the alternative approach is evaluated.

The parametrization of a mortar interface is exemplified in fig. 4.2. The coupling situation is depicted in fig. 4.2(a) showing two cuboids that share a common boundary. Each cuboid's geometry is represented by a NURBS patch with quadratic basis functions and open uniform knot vectors. The interpolation of the Lagrange multiplier field with NURBS basis functions is depicted in fig. 4.2(b) and fig. 4.2(c) illustrates the respective discretization for the bilinear Lagrangian interpolation. In this example, the biquadratic NURBS interpolation requires 52 support points compared to 37 for the bilinear Lagrangian interpolation. If the coarser coupling surface of the upper patch were selected as non-mortar side, then there would have been only 16 support points required for the NURBS and nine for the Lagrangian interpolation.

4.4.2 Mortar matrix evaluation

The mortar matrices given in (4.26) must be evaluated by numerical integration. For convenience, the definition of the matrices for the two-patch coupling problem pictured in fig. 4.2 is repeated here. In this definition, $\Omega^{(1)}$ denotes the domain of the non-mortar side patch and $\Omega^{(2)}$ the domain of the mortar side patch.

$$\boldsymbol{m}^{(1)} = \int_{\Gamma_c} \boldsymbol{N}^{c\,T} \boldsymbol{N}^{(1)} \,\mathrm{d}\Gamma_c \qquad\qquad \boldsymbol{m}^{(2)} = -\int_{\Gamma_c} \boldsymbol{N}^{c\,T} \boldsymbol{N}^{(2)} \,\mathrm{d}\Gamma_c, \tag{4.31}$$

The matrix $\boldsymbol{N}^c$ contains interpolation function evaluations of the Lagrange multiplier field. If this field is interpolated by NURBS surface basis functions, the values in $\boldsymbol{N}^c$ depend on the parametric coordinate $\boldsymbol{\xi}^{(1)}$ of the underlying non-mortar side patch. In the alternative case of bilinear Lagrangian interpolation, the interpolation functions are local to their respective mortar segment and thus depend on the local segment coordinate $\boldsymbol{r}_{seg}$. The matrices of NURBS patch basis functions $\boldsymbol{N}^{(1)}$ and $\boldsymbol{N}^{(2)}$ for the interpolation of the patch displacement fields depend on the respective patch parametric coordinate $\boldsymbol{\xi}^{(1)}$ or $\boldsymbol{\xi}^{(2)}$. Performing the numerical integration requires a parametrization that is common to all fields. This is achieved by a triangulation of the coupling interface that respects the different field parametrizations, i.e. the edges of the coupling surface elements of both patches. The triangulation leads to a defined number of integration cells, where each individual cell is associated with exactly one mortar segment, one surface element on the mortar side and one surface element on the non-mortar side. The details of this triangulation process are discussed in the next section (sect. 4.4.3).

If the domain of such a cell in physical space is denoted Ω_{ic}, it is required that

$$\Gamma_c = \bigcup \Omega_{ic} \tag{4.32}$$

and

$$\Omega_{ic_i} \cap \Omega_{ic_j} = \emptyset \qquad \forall \Omega_{ic_i}, \Omega_{ic_j} \in \Gamma_c \quad \text{with} \quad \Omega_{ic_i} \neq \Omega_{ic_j}. \tag{4.33}$$

The mortar matrices for a given coupling interface are then given by the sum of the individual integration cell contributions

$$\boldsymbol{m}^{(1)} = \sum_{ic} \boldsymbol{m}_{ic}^{(1)} \qquad \text{and} \qquad \boldsymbol{m}^{(2)} = \sum_{ic} \boldsymbol{m}_{ic}^{(2)}. \tag{4.34}$$

Each triangular integration cell is defined by its three corner nodes. At this point, it is assumed that the physical coordinates of these nodes as well as their representation in the parametric coordinate spaces $\boldsymbol{\xi}^{(1)}$ and $\boldsymbol{\xi}^{(2)}$ and in the local mortar segment coordinate space $\boldsymbol{r}_{seg}$ are known. Again, the reader is referred to sect. 4.4.3 for determination of these nodal values.

With the integration cell's local interpolation functions

$$N_{ic}^{(1)} = 1 - r - s \qquad N_{ic}^{(2)} = r \qquad N_{ic}^{(3)} = s \qquad \forall (r, s) \in \tilde{\Omega}_{ic} \tag{4.35}$$

being defined in the natural coordinates $\boldsymbol{r}_{ic} = [r \;\; s]^T$ on the domain of the unit triangle $\tilde{\Omega}_{ic} = [0, 1]^2 \subset \mathbb{R}^2$, the physical domain of an integration cell is given be the linear mapping $\boldsymbol{x}^h : \tilde{\Omega}_{ic} \rightarrow \Omega_{ic}$. This mapping is defined as

$$\boldsymbol{x}^h(\boldsymbol{r}_{ic}) \;\; = \sum_{a=1}^{3} N_{ic}^{(a)}(\boldsymbol{r}_{ic})\, \hat{\boldsymbol{x}}_{ic}^{(a)} \;\; = \boldsymbol{N}^{ic}\, \tilde{\boldsymbol{x}}^{ic} \qquad \forall \boldsymbol{r}_{ic} \in \tilde{\Omega}_{ic} \tag{4.36}$$

where $\hat{\boldsymbol{x}}_{ic}^{(a)}$ represents the physical coordinate of the integration cell's node a and where $\tilde{\boldsymbol{x}}^{ic}$ contains all the nodal coordinates of that cell. In the same manner, the parametric and mortar segment coordinates of any point within a cell can be expressed via the respective mappings

$$\boldsymbol{\xi}^{(1),h}(\boldsymbol{r}_{ic}) = \sum_{a=1}^{3} N_{ic}^{(a)}(\boldsymbol{r}_{ic})\, \hat{\boldsymbol{\xi}}_{ic}^{(1),(a)} = \boldsymbol{N}^{ic}\, \tilde{\boldsymbol{\xi}}^{(1),ic} \qquad \forall \boldsymbol{r}_{ic} \in \tilde{\Omega}_{ic} \tag{4.37}$$

$$\boldsymbol{\xi}^{(2),h}(\boldsymbol{r}_{ic}) = \sum_{a=1}^{3} N_{ic}^{(a)}(\boldsymbol{r}_{ic})\, \hat{\boldsymbol{\xi}}_{ic}^{(2),(a)} = \boldsymbol{N}^{ic}\, \tilde{\boldsymbol{\xi}}^{(2),ic} \qquad \forall \boldsymbol{r}_{ic} \in \tilde{\Omega}_{ic} \tag{4.38}$$

$$\boldsymbol{r}_{seg}^{h}(\boldsymbol{r}_{ic}) \;\; = \sum_{a=1}^{3} N_{ic}^{(a)}(\boldsymbol{r}_{ic})\, \hat{\boldsymbol{r}}_{seg,ic}^{(a)} = \boldsymbol{N}^{ic}\, \tilde{\boldsymbol{r}}_{seg}^{ic} \qquad \forall \boldsymbol{r}_{ic} \in \tilde{\Omega}_{ic} \tag{4.39}$$

With these mappings, a single integration cell's contribution to the mortar matrix can be expressed as

$$\boldsymbol{m}_{ic}^{(1)} = \int_{\Omega_{ic}} \boldsymbol{N}^{c^T}\Big(\boldsymbol{r}_{seg}^{h}\left(\boldsymbol{r}_{ic}\right)\Big)\, \boldsymbol{N}^{(1)}\big(\boldsymbol{\xi}^{(1),h}\left(\boldsymbol{r}_{ic}\right)\big)\, \mathrm{d}\Omega_{ic} \tag{4.40}$$

$$\boldsymbol{m}_{ic}^{(2)} = -\int_{\Omega_{ic}} \boldsymbol{N}^{c^T}\Big(\boldsymbol{r}_{seg}^{h}\left(\boldsymbol{r}_{ic}\right)\Big)\, \boldsymbol{N}^{(2)}\big(\boldsymbol{\xi}^{(2),h}\left(\boldsymbol{r}_{ic}\right)\big)\, \mathrm{d}\Omega_{ic} \tag{4.41}$$

for the Lagrangian interpolation and correspondingly for the NURBS surface interpolation of the Lagrange multiplier field as

$$\boldsymbol{m}_{ic}^{(1)} = \int_{\Omega_{ic}} \boldsymbol{N}^{c^T}\left(\boldsymbol{\xi}^{(1),h}\left(\boldsymbol{r}_{ic}\right)\right) \boldsymbol{N}^{(1)}\left(\boldsymbol{\xi}^{(1),h}\left(\boldsymbol{r}_{ic}\right)\right) \, \mathrm{d}\Omega_{ic} \tag{4.42}$$

$$\boldsymbol{m}_{ic}^{(2)} = -\int_{\Omega_{ic}} \boldsymbol{N}^{c^T}\left(\boldsymbol{\xi}^{(1),h}\left(\boldsymbol{r}_{ic}\right)\right) \boldsymbol{N}^{(2)}\left(\boldsymbol{\xi}^{(2),h}\left(\boldsymbol{r}_{ic}\right)\right) \, \mathrm{d}\Omega_{ic}. \tag{4.43}$$

With these definitions, all terms of the integrand depend on the natural coordinate system of the cell. When Gauss quadrature as outlined in sect. 3.3.3 is applied, the mortar cell matrices become

$$\boldsymbol{m}_{ic}^{(1)} = \frac{1}{2}\sum_{i=1}^{n_{ip}} w_i \, \boldsymbol{N}^{c^T}\left(\boldsymbol{\xi}^{(1),h}\left(\boldsymbol{r}_{ic}^{(i)}\right)\right) \boldsymbol{N}^{(1)}\left(\boldsymbol{\xi}^{(1),h}\left(\boldsymbol{r}_{ic}^{(i)}\right)\right) \det \boldsymbol{J}(\boldsymbol{r}_{ic}^{(i)}) \tag{4.44}$$

$$\boldsymbol{m}_{ic}^{(2)} = -\frac{1}{2}\sum_{i=1}^{n_{ip}} w_i \, \boldsymbol{N}^{c^T}\left(\boldsymbol{\xi}^{(1),h}\left(\boldsymbol{r}_{ic}^{(i)}\right)\right) \boldsymbol{N}^{(2)}\left(\boldsymbol{\xi}^{(2),h}\left(\boldsymbol{r}_{ic}^{(i)}\right)\right) \det \boldsymbol{J}(\boldsymbol{r}_{ic}^{(i)}) \tag{4.45}$$

in the case of NURBS surface interpolation (Lagrangian interpolation is analogous). The numerical integration is performed on the unit triangle domain $\tilde{\Omega}_{ic}$ and the determinant of the Jacobian $\det \boldsymbol{J}$ relates differential element sizes on the unit domain to the actual sizes in physical space. For the linear interpolation functions however, the Jacobian and also its determinant are constant. Thus, in practice, the integration cell size can be evaluated directly from the nodal coordinates in physical space by the relation below.

$$A = \frac{1}{2}\left\|\left(\hat{\boldsymbol{x}}_{ic}^{(2)} - \hat{\boldsymbol{x}}_{ic}^{(1)}\right) \times \left(\hat{\boldsymbol{x}}_{ic}^{(3)} - \hat{\boldsymbol{x}}_{ic}^{(1)}\right)\right\|_2 \tag{4.46}$$

The number of required integration points n_{ip} clearly depends on the degree of the NURBS surface basis functions and, if applicable, on the Lagrangian mortar segment interpolation functions. Integration rules for the triangular domain are found in Dunavant [55].

It is to be noted, that the patch basis functions $\boldsymbol{N}^{(1)}$ and $\boldsymbol{N}^{(2)}$ and their associated parametric coordinates $\boldsymbol{\xi}^{(1)}$ and $\boldsymbol{\xi}^{(2)}$ are not local to a given surface element on the respective mortar or non-mortar side of the coupling interface but they are only local to the full patch surface of the given side. This would allow to triangulate the coupling interface without observing the existing element edges, when the NURBS surface interpolation of the Lagrange multiplier field is used (instead of the bilinear Lagrangian interpolation). However, there would be undesirable consequences for the implementation. An integration cell could be associated with more than one surface element on a given side. Thus, the number of nonzero patch surface basis functions and consequently the size of the mortar cell matrix were neither constant nor known in advance for all cells of a given mortar coupling. On the other side, there would be advantages for curved coupling interfaces. These could be discretized with higher order triangular integrations cells with curved cell edges, since the expensive identification of intersections with the also curved element edges of the mortar and non-mortar patches were unnecessary.

4.4.3 Coupling interface evaluation

4.4.3.1 General

In the context of the integrated analysis framework that is developed within this work, the coupling of the individual patches is to be carried out automatically. Input data other than the patch geometry mapped into physical space shall not be required. Thus, no additional information associated with the structural analysis or the specific numerical method has to be stored in the building information model. The implementation of the interface evaluation consists of two process components – interface detection and interface discretization. These are discussed subsequently.

4.4.3.2 Interface detection

Since the arbitrarily shaped solid objects are defined in parameter space and then only mapped to physical space, the automatic interface detection may be cumbersome and the numerical effort can be very high. Therefor it is important to use efficient algorithms that are suited for the specific domain of application. For this work, a hierarchically organized process that makes use of the convex hull property of NURBS patches is developed. This process is outlined below.

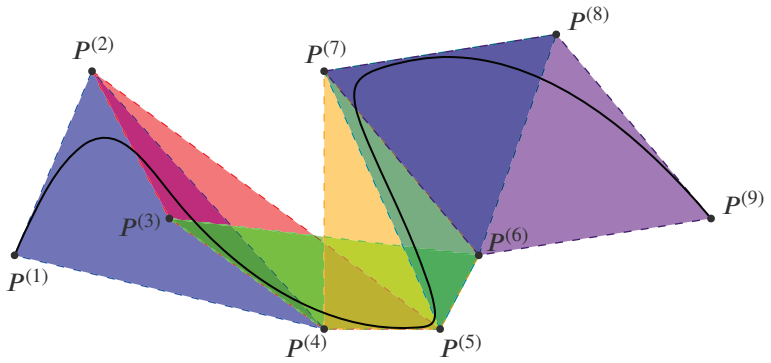


Figure 4.3: Convex hull property of a cubic NURBS curve. For all control points the convex hull of $3 + 1$ neighboring control points is pictured by the (overlapping) colored polygons. The curve is guaranteed to lie within the union of these convex hulls.

For a NURBS curve, the convex hull property states that any point on the curve lies within the union of the convex hulls formed by $p + 1$ neighboring control points of the curve's control polygon with p denoting the polynomial degree of the curve's basis functions. This is illustrated in fig. 4.3. As the convex hull property exists correspondingly for multivariate NURBS formulations, the control point coordinates can be used to construct bounding boxes that facilitate the process of contact detection. The bounding box of a given solid patch is determined by looping over the outmost control points of the patch's control lattice and storing the minimum and maximum coordinate values independently for the three coordinate directions in the two vectors $\boldsymbol{x}_{min}$ and $\boldsymbol{x}_{max}$. These two vectors define an axis aligned bounding box (AABB).

Likewise, the AABB can be defined by its center $\boldsymbol{x}_{cent}$ and the half-lengths of the box's edges $\boldsymbol{x}_{rad}$. The center and the edges' half-lengths can be calculated from $\boldsymbol{x}_{min}$ and $\boldsymbol{x}_{max}$.

$$\boldsymbol{x}_{rad} = \frac{1}{2}\left(\boldsymbol{x}_{max} - \boldsymbol{x}_{min}\right) \qquad \text{and} \qquad \boldsymbol{x}_{cent} = \boldsymbol{x}_{min} + \boldsymbol{x}_{rad}. \tag{4.47}$$

If two AABBs do not overlap, it is impossible for their underlying patches to share a common boundary. Candidate pairs of coupled patches can thus be identified with a simple overlap test. A positive result requires the axis projected distance between the two centers to be smaller than or equal to the sum of the two radii in all of the coordinate directions, i.e.

$$x_{diff,i} \geq 0 \qquad \forall i \in \{1,2,3\} \tag{4.48}$$

for

$$\boldsymbol{x}_{diff} = \boldsymbol{x}^{(1)}_{rad} + \boldsymbol{x}^{(2)}_{rad} - \text{abs}\left(\boldsymbol{x}^{(1)}_{cent} - \boldsymbol{x}^{(2)}_{cent}\right). \tag{4.49}$$

Algorithm 4.1 Evaluate coupled NURBS patches

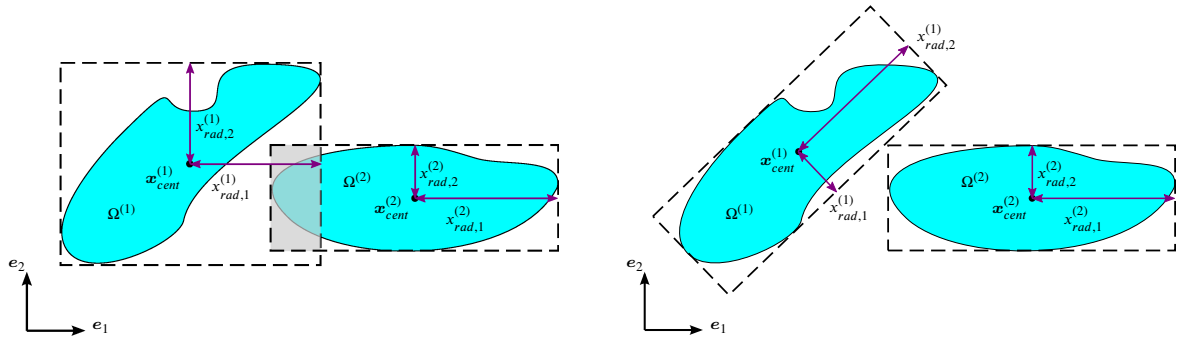
```
for NurbsPatch p1 ← 1 to NumPatches do
    for NurbsPatch p2 ← p1 + 1 to NumPatches do
        if AABB(p1).intersects(AABB(p2)) then
            pc ← NurbsPatchCouple(p1,p2)
            if not pc.isCouple() then                    ▷ see alg. 4.2
                delete pc
            end if
        end if
    end for
end for
```

In case an intersection of two AABBs is detected, the search for a common boundary of the associated patches is continued with the next hierarchical step. This is done by recovering the six boundary surface patches[3] for each of the two NURBS solids and performing an overlap test for all possible combinations of the AABBs associated with these surfaces.

A positive test result induces another hierarchical step in the interface search: an oriented bounding box (OBB) overlap test. OBBs are cuboids that are in contrast to the AABBs not aligned with the global coordinate axes but arbitrarily oriented. Thus, an OBB can enclose the underlying geometric object more tightly. In consequence OBB overlap tests have a higher significance with regard to the probability of a patch coupling. The improved performance comes at the price of higher numerical costs. The evaluation of the box properties and the execution of intersection tests is more expensive, since a rotation matrix is additionally required for the definition. Therefore, OBBs are only created for those surface patches that are already suspected to be in contact because of a positive AABB overlap test.

The determination of an appropriate rotation matrix is difficult as there is no unique procedure to do so. Gottschalk [72] outlines several methods for the construction of a rotation matrix and he also proposes an efficient OBB overlap test. This work follows the covariance-based approach

[3] The process of recovering the boundary surfaces from a solid patch was outlined in sect. 3.4.6.



(a) AABB **(b)** OBB

Figure 4.4: Comparison of axis aligned bounding boxes (AABBs) with oriented bounding boxes (OBBs). While the AABBs in (a) detect an intersection for the given example, the OBBs in (b) do not.

that uses a set of points as its basis. A covariance matrix is constructed for the coordinates of the given points. Together with the centroid of these points the matrix describes the approximate distribution and thus the shape of the points. The largest eigenvector of the covariance matrix points in the direction of the largest variance of the point's coordinates and thus likely in the direction of the largest extent of the geometric object that is the source of these points. All three eigenvectors form an orthogonal system and therefore they can be used to construct the rotation matrix. The grid structure of the NURBS control points makes an appropriately distributed set of points readily available, the centroid and the covariance matrix can easily be computed. With the rotation matrix at hand, the bounds and the center of a cuboid enclosing the control points can be determined in the rotated coordinate system. For details of the OBB creation and the algorithm of the intersection test with another OBB, the reader is referred to the original source [72].

After a positive OBB overlap test for a pair of patch boundary surfaces, a further step in the hierarchical determination of the coupling interface is executed. Therefor, the 2D control nets of the surfaces are decomposed into control net sections where each section consists of $(p+1)\times(q+1)$ control points. Those are the control points that form the convex hull whose union with all other respective hulls encloses the surface. (For a curve, the overlapping convex hulls were depicted in fig. 4.3.) For the control net of a surface with polynomial degrees p and q and $n \times m$ control points this leads to $(n-p)\times(m-q)$ control net sections. AABBs and if required also OBBs can then be constructed individually for the control net sections of the two patch surfaces that are presumed to be coupled. The respective union of these bounding boxes clearly encloses an arbitrary shaped surface tighter than a global box can do. Looping over the sections of one surface and performing AABB and optionally OBB overlap tests with all sections of the second surface identifies those sections that may contain coupled parts of the surface. Thus, the possible interface area is further narrowed down to specific pairs of control net sections. Since the control net sections of a surface overlap with each other,[4] those with a positive overlap test result can be condensed to describe one (or more) areas of possible contact by the indices of the control net.

Optionally, it is possible to create a copy of the patch surfaces that are believed to be in contact and to refine them by knot insertion. After reaching a desired level of refinement, the pre-

[4] neighboring sections have an offset of one position in the control net

Algorithm 4.2 Evaluate coupled patch boundary surfaces

```
function NURBSPATCHCOUPLE.ISCOUPLE()                    ▷ has access to variables in alg. 4.1

    Initialize:
    pbs_1[1 − 6] ← PatchBoundarySurface(p_1)            ▷ create boundary surfaces
    pbs_2[1 − 6] ← PatchBoundarySurface(p_2)
    foundCouple ← false
    for i ← 1 to 6 do
        for j ← 1 to 6 do
            if AABB( pbs_1[i] ).intersects( AABB(pbs_2[j] )) then
                if OBB( pbs_1[i] ).intersects( OBB(pbs_2[j] )) then
                    psc ← PatchSurfaceCouple( pbs_1[i], pbs_2[j] )
                    if psc.isCouple() then              ▷ see alg. 4.3
                        foundCouple ← true
                    else
                        delete psc
                    end if
                end if
            end if
        end for
    end for
    return foundCouple
end function
```

viously outlined procedure of control net section overlap tests can be performed for the now much smaller control net sections where sections already determined to be uncoupled may be excluded. This approach allows to narrow the potential coupling area to an arbitrary fine resolution of control net sections with the efficient bounding box overlap tests. However, the practical implementation showed that this is not a necessity for an adequate performance of the interface evaluation.

In the simplest case, a pair of condensed control net sections denotes a rectangular area of the control net for each of the two patch boundary surfaces of the potentially coupled patches. The two rectangular areas are identified by the four triples $(i, j, k)_P^{(I)}$ with $I \in \{1, 2\}$ and $P \in \{b, e\}$ where (i, j, k) denotes the index of a control point in the control lattice of the underlying solid patch, I denotes the subdomain and P the start (b) or the stop (e) index of the rectangle. Since these points are on a boundary surface, one of the three indices must be on the boundary of the control lattice and thus constant for a subdomain.

The physical coordinates of the four control points identified by $(i, j, k)_P^{(I)}$ can be projected back into the parametric space of their respective patch which defines a rectangular area in parameter space that corresponds to the condensed control net section. The next step in the interface evaluation procedure is to define a grid of parametric coordinates over that rectangular area of the potential non-mortar side. All of these coordinates are projected into physical space and then, each physical coordinate is projected on the opposing surface, the potential mortar side. The projection delivers a parametric coordinate in the parametric space of the mortar side and an associated physical coordinate. If the distance between the two physical coordinates is below a given threshold, patch contact exists and the non-mortar side parametric coordinate is

Algorithm 4.3 Evaluate coupled control net sections

```
function PATCHSURFACECOUPLE.ISCOUPLE()                    ▷ has access to variables in alg. 4.2

    Initialize:
    cns_1[1 − n_1][1 − m_1] ← ControlNetSections(pbs_1)        ▷ 2D array of control net sections
    cns_2[1 − n_2][1 − m_2] ← ControlNetSections(pbs_2)
    CNSCs ← CNSCs()                                            ▷ empty list of control net section couples
    for i_1 ← 1 to n_1 do
        for j_1 ← 1 to m_1 do
            for i_2 ← 1 to n_2 do
                for j_2 ← 1 to m_2 do
                    if AABB( cns_1[i_1][j_1] ).intersects( AABB( cns_2[i_2][j_2] )) then
                        if OBB( cns_1[i_1][j_1] ).intersects( OBB( cns_2[i_2][j_2] )) then
                            CNSCs.append(
                                ControlNetSectionCouple( cns_1[i_1][j_1], cns_2[i_2][j_2] ))
                        end if
                    end if
                end for
            end for
        end for
    end for

    condense( CNSCs )                                          ▷ build union of overlapping control net sections

    for cnsc ← CNSCs.first() to CNSCs.last() do
        if not cnsc.isCouple() then                                                ▷ see alg. 4.4
            delete cnsc
        end if
    end for

    if size( CNSCs ) > 0 then
        return true
    else
        return false
    end if
end function
```

associated with the newly found mortar side parametric coordinate to form a parametric point couple between the two parameter spaces of the underlying patches. Once this is done for all points of the grid and if there are at least two parametric point couples per parametric direction (to define a contact surface, not an edge) surface contact of the two patches is confirmed.

The final step of the interface detection is the correct description of the coupling interface's boundary. For each of the points in the grid of parametric points on the non-mortar side surface it is known whether or not it forms a couple with an opposing point on the mortar side. The interface area is defined by the set of contiguous point couples. The points on the boundary of the contiguous set are the best available description of the interface area's boundary. However, this is not the true boundary which is located somewhere between these boundary points and their respective neighboring point that does not have a match on the opposing mortar side. Thus,

Algorithm 4.4 Evaluate coupled parametric points

```
function CONTROLNETSECTIONCOUPLE.ISCOUPLE()        ▷ has access to variables in alg. 4.3

    Inputs:
    NurbsPatch p1, p2                              ▷ p1 and p2 are the underlying patches
    Initialize:
    ppstart1 ← ParametricPoint( p1.projectToParameter( cns1.start() ))
    ppstop1 ← ParametricPoint( p1.projectToParameter( cns1.stop() ))
    ppstep1 ← (ppstop1 − ppstart1) / numGridPoints
    PPCs = PPCs( numGridPoints)                    ▷ array of parametric point couple lists
    i ← 1
    for pp1[1] ← ppstart1[1] to ppstop1[1] incr ppstep1[1] do
        for pp1[2] ← ppstart1[2] to ppstop1[2] incr ppstep1[2] do
            physCoord1 ← p1.projectToPhysical( pp1 )
            pp2 ← p2.projectToParameter( physCoord1 )
            physCoord2 ← p2.projectToPhysical( pp2 )
            dist ← distance( physCoord1, physCoord2 )
            if dist < εc then                      ▷ εc denotes the coupling threshold
                PPCs[i].append( ParametricPointCouple( pp1, pp2 ))
            end if
        end for
        i ← i + 1
    end for

    if PPCs.isTwoDimensionalGrid() then
        PPCs.improveInterfaceBoundaryPoints()
        return true
    else
        return false
    end if
end function
```

the maximum error in the description of the interface boundary in parameter space corresponds to the grid size. With each iteration in a recursive approach, this error can by reduced by 50 per cent: A grid point that marks the current boundary forms a line segment with its neighboring grid point that is not part of the interface area. Bisecting this line segment introduces a new sub grid point for which the coupling status can be evaluated. If the new point forms a point couple, the interface area is extended, otherwise it is not. Either way, there will be two new line segments, each with half the original length. Only for one of these two segments, one end point is on the interface area and the other is off that area. For this segment the recursive bisecting process is repeated until the desired accuracy in the description of the coupling interface's boundary is reached.

4.4.3.3 Projection of physical coordinates to parameter space

Projecting coordinates from the parameter space of a NURBS patch to the physical space is a simple procedure that bases on the NURBS formulation. Performing the reverse projection

from physical to parameter space is not as straightforward. Yet, it is essential for the coupling algorithms. Therefore, the procedure is briefly discussed in this section.

The general mathematical description of the problem is to find the parametric coordinate $\boldsymbol{\xi}_p(\boldsymbol{P})$ such that the distance between the physical coordinate $\boldsymbol{x}(\boldsymbol{P})$ of the given point $\boldsymbol{P}$ and the NURBS projection $\boldsymbol{S}(\boldsymbol{\xi}_p(\boldsymbol{P}))$ is minimized, i.e.

$$\boldsymbol{\xi}_p(\boldsymbol{P}) \mid \left\| \boldsymbol{S}\left(\boldsymbol{\xi}_p(\boldsymbol{P})\right) - \boldsymbol{x}(\boldsymbol{P}) \right\|_2 \leq \|\boldsymbol{S}(\boldsymbol{\xi}) - \boldsymbol{x}(\boldsymbol{P})\|_2 \qquad \forall \boldsymbol{\xi}, \boldsymbol{\xi}_p \in \hat{\Omega}^2. \tag{4.50}$$

Piegl and Tiller [129, sec. 6.1] proposed a Newton-Raphson iteration based method to find an optimal value for $\boldsymbol{\xi}_p$ that equally works for physical points on the patch (point inversion) and off the target patch (point projection). The Newton-Raphson iteration requires the evaluation of the NURBS basis functions and its partial derivatives for the candidate point in each iteration. This can become numerically expensive when there are a lot of points to project. Therefore the grid size and the accuracy required for the projection in alg. 4.4 have to be chosen carefully. The numerical cost of the point projection also constitutes the motivation to use the previously described bounding box based hierarchical approach down to the level of control net sections. The bounding boxes are by far cheaper to evaluate than a point projection is.

The numerical cost of a single point projection is strongly influenced by the selection of the initial value of $\boldsymbol{\xi}_p$ which is used to start the Newton-Raphson iteration. The quality of that selection determines the number of necessary iterations and thus the overall performance of the method. With an inappropriate value, the Newton-Raphson iteration does not converge and thus the method may fail. This is particularly relevant for points close to the patch boundaries.[5]

The implementation in this work distinguishes two situations in the determination of the initial value for $\boldsymbol{\xi}_p$. When projecting control points onto the boundary surface of a NURBS patch, which is required for alg. 4.4, the initial value is determined from the local support of the basis functions associated with that control point. The bounds of these functions are exactly defined by the knot vectors. The median coordinate in parametric space is easily determined and constitutes an appropriate initial guess for patches with moderately high polynomial degrees of the basis functions. For control points on the edge of a boundary surface, one parametric coordinate value is directly known. Also, the bounds of the functions' support are used as the projection range, the rest of the parametric space is excluded. The second situation arises when the grid points are to be projected onto the opposing mortar boundary surface, see also alg. 4.4. In this case, the initial guess for $\boldsymbol{\xi}_p$ is inferred from the control net section that contains the grid point and its coupled counterpart on the mortar side. Then again, the support of the basis functions associated with the respective control points of the coupled counterpart is used to determine a rough estimate of the parametric coordinate. Once a coupled pair of parametric points in the two patch domains was found, the projected point plus an appropriate offset is used as the initial guess for the next grid point.

In case the point projection with the outlined technique fails, the more general approach by Ma and Hewitt [109] was implemented as a backup procedure. With this method, the boundary surface of a patch is subdivided into individual Bézier patches. Then, the relation of the point to project with the control points of the Bézier patches is used to determine candidate Bézier

[5] Due to the error-proneness paired with the numerical cost of their original approach [129], Piegl and Tiller proposed an alternative in [130]. However, as the original approach worked satisfactorily with the described method of selecting an initial value for $\boldsymbol{\xi}_p$, the alternative was not implemented.

patches that are closest to the given point. In a recursive process, the candidates are further subdivided for the control net to converge to the patch surface. When the patch size is sufficiently small and thus the control net is almost flat, a profound estimation of the parametric point is made by projecting it on the plane of the control net. This estimation is subsequently improved with a Newton-Raphson iteration.

4.4.3.4 Interface discretization

The 2D array of coupled parametric points evaluated in sect. 4.4.3.2 constitutes the input data for the interface discretization. Of these points, only those representing the corners of the coupling interface's polygonal boundary in parameter space are relevant for the discretization. The interior points are no longer required, non-coupled parametric regions within the bounding polygon are not supported by the current implementation.

The goal of the discretization process was previously discussed: the coupling interface is to be triangulated such that each triangle is associated with exactly one surface element on the mortar side and one surface element on the non-mortar side of the underlying patches (referred to as *single element constraint*). Though the resulting triangles denote individual integration domains, they do not constitute the finite elements for the Lagrange multiplier interpolation nor are the derivatives of their cell interpolation functions required at any point of the mortar formulation.[6] Thus, the numerical performance of the mortar method does not require the triangles to fulfill specific shape criteria.[7] In consequence, there is also no necessity to apply the standard procedure of Delaunay triangulation.[8] Any other procedure that is capable of coping with the single element constraints can be used instead. For this work, a method originally proposed by Oblonsek and Guid [123] for the reconstruction of surfaces from 3D scattered points was adapted to the requirements of the interface discretization. The method works with a defined set of points without the need of additionally introducing intermediate points that would again require point projections, it is robust with respect to the existing constraints and it has linear time complexity. The process of interface discretization together with key features and modifications of the surface reconstruction method are discussed subsequently, for the complete algorithm however, the reader is referred to the original paper.

The first step in the discretization process is the evaluation of those surface elements of the master and slave NURBS patches that are part of the coupling interface. By looping over the points denoting the polygonal interface boundary, the NURBS elements containing the boundary points are identified. The specific element IDs are easily determined from the two knot vectors of the respective boundary surface. If, in any parametric direction, the element containing the current point is not identical or adjacent to the one previously evaluated, an additional point is created midway in the parameter space of the current and the previous point. Then, the element is determined for the new point and the procedure is recursively repeated until all affected elements have a direct neighbor. Examining all boundary points results in a set of interconnected elements for which the coupling interface's boundary is known to run through them. Obviously, all other elements enclosed by these boundary elements must also

[6] Note that the determinant of the Jacobian in eqs. (4.44) and (4.45) is determined from geometric considerations and the shape function derivatives are not used.

[7] See Shewchuk [152] for shape criteria of linear elements in FEA.

[8] The goal of a Delaunay triangulation is to maximize the sum of the smallest angles of all triangles and thereby avoiding triangles with high aspect ratios.

be part of the coupling interface. Thanks to the structured grid layout of the elements, their identification is trivial.

All affected surface elements are stored as mortar segments and each segment is associated with its four corner points. Such a point is denoted a mortar node $\hat{P}_m$. These nodes are uniquely numbered over the entire interface, irrespective of their origin on the master or slave side, their total number is n_{mn}. A nodal point's parametric location $\boldsymbol{\xi}^{(\alpha)}(\hat{P}_m)$ is given by the nature of its definition, the physical location $\boldsymbol{x}(\hat{P}_m)$ is evaluated and the parametric location $\boldsymbol{\xi}^{(\alpha)}(\hat{P}_m)$ on the opposing interface surface is retrieved via point projection.[9] Also stored with each mortar node are two sets of associated elements $\mathcal{E}^{(\alpha)} = \{e_i \,|\, \boldsymbol{\xi}^{(\alpha)}(\hat{P}_m) \in \hat{\Omega}^{(\alpha)}_{e_i}\}_{i=1}^{n_e}$, separated for the mortar and the non-mortar boundary surface. For a typical node originating from an element corner, this would be the four surrounding elements in the 2D parametric plane and for the same node projected to the opposing surface this is likely to be only a single element containing that node. Eventually, all line segments corresponding to the mortar segment edges are stored. Here it is to be noted, that these line segments deviate from the element edges in physical space for elements with curved boundaries. This circumstance becomes relevant when later defined integration points are located within the area bounded by the four line segments but outside the true element area. In such a case, intermediate points have to be introduced that split the concerned mortar segments into four smaller quadrilateral regions and thus reduces the linearization error.

The outlined first discretization step is done alike for the mortar and the non-mortar side of the coupling interface. The following second step affects again the parametric point couples that represent the boundary of the coupling interface. Though these points are not associated with the corners of the mortar segments, mortar nodes representing these points are created and stored with their associated data which is the physical location, the location in the two parametric domains, and the element IDs of the underlying surface elements. Boundary points that are aligned with their predecessors and successors in physical space are excluded, since they are superfluous for the description of the polygonal boundary.

In the next step, all line segments derived from the master and slave segment edges are evaluated for possible cross points in physical space. Since the intersections are numerically cheap to evaluate, a brute force search tests each line segment from the master patch side with all segments from the slave patch side. Again, mortar nodes are created for all evaluated intersections. In the same manner, intersections of the segment edges with the polygonal interface boundary are computed and corresponding nodes are created.

Master and slave segments may partially be located outside the polygonal bounded coupling interface, and so may the nodes associated with these segments. Since the mortar nodes form the basis for the triangulation, those nodes not directly part of the coupling interface must be identified. For this purpose, an even-odd rule is implemented: A line between the node of interest and a point known to be outside the interface bounding polygon is defined in order to count how often this line crosses the polygon edges. For an even number, the node is outside the polygon, otherwise it is located inside. The search is performed in the 2D parameter space of the master and the slave bounding surface. Details on the algorithm and implementation strategies are provided by Hormann and Agathos [80]. Furthermore, there may be nodes with identical location in physical space, e.g. as a result of coincident master and slave element corners. Also these nodes must be identified and the redundant definition resolved.

[9] Here, α represents one of the subdomains being coupled, i.e. $\alpha \in (1, ..., n_{sub})$

The set of mortar nodes that are inside or on the boundary of the coupling interface is identified as $\mathcal{S} = \{\hat{P}_m^{(a)} \mid \boldsymbol{x}(\hat{P}_m^{(a)}) \in \Gamma_c\}_{a=1}^{n_{mn}}$. These nodes uniquely define the corners of the bounding polygon and all existing cross points of the line segments that represent either the bounding polygon edges or the edges of the master and the slave patch surface elements. The set constitutes the scattered point input data used for the surface reconstruction algorithm of Oblonsek and Guid [123].

The algorithm starts with the evaluation of the k-neighborhood for all mortar nodes in $\mathcal{S}$. For a node $\hat{P}_m^{(a)}$ the k-neighborhood defines an ordered subset

$$\mathcal{S}_{\leq}^{(a)} \subset \mathcal{S} \quad \text{with} \quad \mathcal{S}_{\leq}^{(a)} = \left\{\hat{P}_m^{(g_{ab})}\right\}_{b=1}^{k}$$

where g_{ab} stands for the elements of the tuple $\left(g_{a1}, g_{a2}, \ldots, g_{ak}\right)$ which refer to the indices of the k mortar nodes closest to a, i.e. those nodes for which

$$\left\|\boldsymbol{x}\left(\hat{P}_m^{(a)}\right) - \boldsymbol{x}\left(\hat{P}_m^{(g_{ai})}\right)\right\|_2 \leq \left\|\boldsymbol{x}\left(\hat{P}_m^{(a)}\right) - \boldsymbol{x}\left(\hat{P}_m^{(g_{aj})}\right)\right\|_2 \text{ with } i < j$$

and

$$\left\|\boldsymbol{x}\left(\hat{P}_m^{(a)}\right) - \boldsymbol{x}\left(\hat{P}_m^{(g_{ak})}\right)\right\|_2 \leq \left\|\boldsymbol{x}\left(\hat{P}_m^{(a)}\right) - \boldsymbol{x}\left(\hat{P}_m^{(c)}\right)\right\|_2 \text{ with } c \neq a \text{ and } c \neq g_{ab}$$

holds. A space partitioning scheme divides the space covered by the nodes into uniform cells, each with a constant number of nodes that is independent of the total number of nodes. As the algorithm for the construction of the k-neighborhood for a given node only visits a limited number of cells, its time complexity for all nodes is linear. The k-neighborhood of all nodes is stored via the indices g_{ab} in a 2D array $K(n_{mn}, k)$.

In contrast to the original scattered point triangulation algorithm, it is known in advance, that the points, i.e. the mortar nodes, do not represent a closed surface[10] and thus, there is no inside or outside orientation. Also, the node normals can be determined in advance and directly from the bounding surfaces of the coupled NURBS patches.[11] Hence, there is no restriction for the selection the first node used to start the triangulation and so the nodes closest to the center of the coupling interface is selected and denoted $\hat{P}_m^{(st)}$. The second node $\hat{P}_m^{(nd)}$ is taken from the k-neighborhood $K(st, b)$ for $b = 1, 2, \ldots, k$ as the first second node candidate that fulfills the single element constraint. Together, these two nodes constitute the first triangle edge.

Triangles are added to the triangulation one at a time by creating a new triangle from an existing triangle edge and a new node on the right of that edge. Each node $\hat{P}_m^{(a)}$ is associated with a tuple $(prv_a, nxt_a, used_a, insd_a)$ where $prv - a - nxt$ denote the indices of the three nodes that represent the triangulation border, $used$ and $insd$ are Booleans where $used_a$ is true

[10] a surface without boundaries

[11] The normal of a node is computed as the normalized cross product of two linearly independent tangent vectors of that node. These tangent vectors are determined from the partial derivatives of the NURBS surface on which the node is located, for which the algorithm is given in [129, sec. 4.5]. They are oriented in the positive parametric direction. Since such a surface is the bounding surface of a NURBS solid, the calculated normal is oriented in the positive third parametric direction, when the correct order of the tangent vectors is preserved. From the location of the bounding surface with respect to the solid, the outward orientation is known to be either the positive or the negative parametric direction and the normal vector can be oriented accordingly. To ensure a consistent orientation of the node normals on the coupling interface, the normals of nodes originating on the slave surface have to be flipped.

when $\hat{P}_m^{(a)}$ is in the triangulation and $insd_a$ is true when $\hat{P}_m^{(a)}$ is not on the triangulation border anymore. For the first two nodes, the tuples take the values $(nd_{st}, nd_{st}, true_{st}, false_{st})$ and $(st_{nd}, st_{nd}, true_{nd}, false_{nd})$.

When $\hat{P}_m^{(cur)}$ is a node on the triangulation border and $\hat{P}_m^{(nxt)}$ is determined from nxt_{cur}, these two nodes define the edge $e_{cur,nxt}$ on the triangulation border. Candidate nodes to form a new triangle are determined from

$$S^{(cur,nxt)} = S_{\leq}^{(cur)} \cup S_{\leq}^{(nxt)} \tag{4.51}$$

which is the k-neighborhood of $e_{cur,nxt}$. The centroid of the neighborhood is given by

$$\boldsymbol{c} = \frac{1}{m} \sum_{a \in S^{(cur,nxt)}} \boldsymbol{x}\left(\hat{P}_m^{(a)}\right) \quad \text{with} \quad m = \left|S^{(cur,nxt)}\right| \tag{4.52}$$

and a plane through $\boldsymbol{c}$ that is tangent to the new triangle can similarly be determined by the approximated normal vector

$$\boldsymbol{n}^c = \frac{\sum_{a \in S^{(cur,nxt)}} \boldsymbol{n}\left(\hat{P}_m^{(a)}\right)}{\left\| \sum_{a \in S^{(cur,nxt)}} \boldsymbol{n}\left(\hat{P}_m^{(a)}\right) \right\|_2}. \tag{4.53}$$

In contrast to [123] there is no requirement to solve a minimization problem in order to determine $\boldsymbol{n}^c$ or to correct to orientation of the normal vector. The projection of the nodes in $S^{(cur,nxt)}$ onto the tangent plane is given by

$$\boldsymbol{x}'\left(\hat{P}_m^{(a)}\right) = \boldsymbol{x}\left(\hat{P}_m^{(a)}\right) - \left(\boldsymbol{x}\left(\hat{P}_m^{(a)}\right) - \boldsymbol{c}\right)^T \boldsymbol{n}^c \cdot \boldsymbol{n}^c \qquad \forall a \in S^{(cur,nxt)}. \tag{4.54}$$

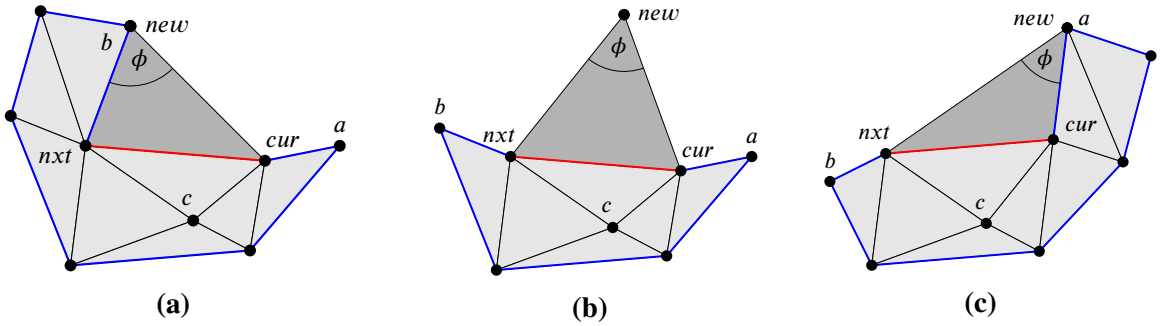


Figure 4.5: Adding a new triangle to the mesh. In the current iteration, the darker triangle is added to existing triangulation, which is represented by the lighter triangles. The triangulation border is depicted by the blue and red line segments with the red line marking the current edge $e_{cur,nxt}$. Node a is the predecessor of the current node cur and node b is the successor of the next node nxt, i.e. $prv_{cur} = a$ and $nxt_{nxt} = b$. Node c is not on the triangulation border, thus $insd_c = true$. The angle ϕ denotes the angle that is to be maximized by eq. (4.58). Each of the figures (a), (b), and (c) refer to one of the three cases in requirement (2). This figure is an adaptation based on the original figure in [123].

With these preparations at hand, the candidate nodes in $S^{(cur,nxt)}$ can be checked for their suitability. A suitable candidate node $\hat{P}_m^{(new)}$

(1) must be on the right side of the current edge, which is guaranteed when

$$\left\| \frac{\left(\boldsymbol{x}'\left(\hat{P}_m^{(new)}\right) - \boldsymbol{x}'\left(\hat{P}_m^{(cur)}\right)\right) \times \left(\boldsymbol{x}'\left(\hat{P}_m^{(nxt)}\right) - \boldsymbol{x}'\left(\hat{P}_m^{(cur)}\right)\right)}{\left\| \left(\boldsymbol{x}'\left(\hat{P}_m^{(new)}\right) - \boldsymbol{x}'\left(\hat{P}_m^{(cur)}\right)\right) \times \left(\boldsymbol{x}'\left(\hat{P}_m^{(nxt)}\right) - \boldsymbol{x}'\left(\hat{P}_m^{(cur)}\right)\right) \right\|_2} - \boldsymbol{n}^c \right\|_2 < \epsilon_{tol} \tag{4.55}$$

holds.

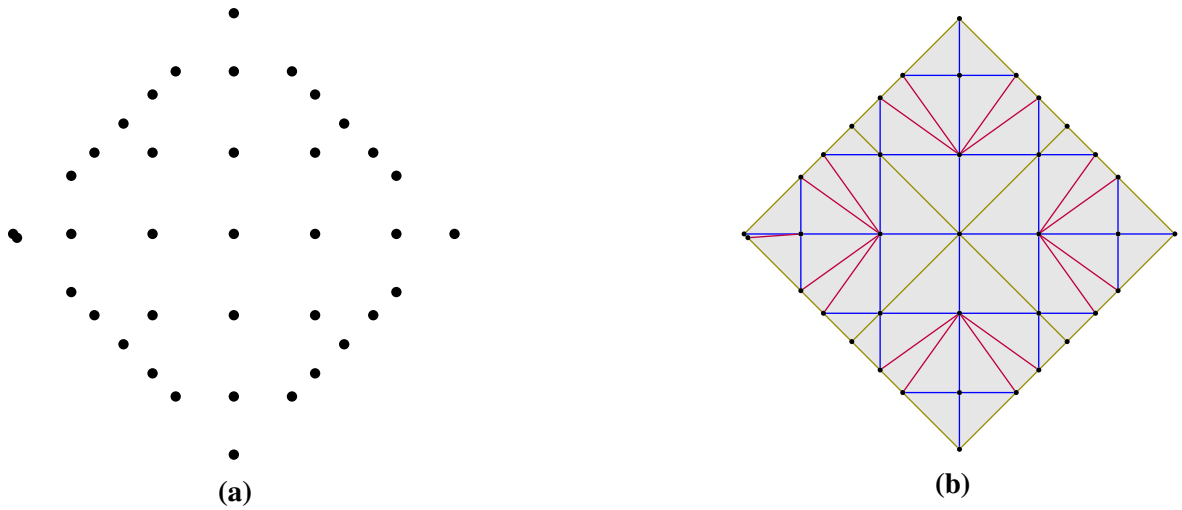
(2) must not yet be part of the triangulation, i.e. $used_{new} = false$ (cf. fig. 4.5(b)), or it is part of the triangulation but not inside the mesh, i.e. $used_{new} = true$ and $insd_{new} = false$, and it is either the predecessor of the current node $prv_{cur} = new$ (cf. fig. 4.5(c)) or the successor of the next node $nxt_{nxt} = new$ (cf. fig. 4.5(a)).

(3) must fulfill the single element constraint together with the current and next node, which can be efficiently guaranteed when the inequality

$$\left| \mathcal{E}^{(m)}\left(\hat{P}_m^{(new)}\right) \cap \mathcal{E}^{(m)}\left(\hat{P}_m^{(cur)}\right) \cap \mathcal{E}^{(m)}\left(\hat{P}_m^{(nxt)}\right) \right| > 0 \tag{4.56}$$

holds for the master side of the coupling interface and likewise for the slave side with

$$\left| \mathcal{E}^{(s)}\left(\hat{P}_m^{(new)}\right) \cap \mathcal{E}^{(s)}\left(\hat{P}_m^{(cur)}\right) \cap \mathcal{E}^{(s)}\left(\hat{P}_m^{(nxt)}\right) \right| > 0. \tag{4.57}$$



(a) **(b)**

Figure 4.6: For the coupled problem depicted in fig. 4.2 the outlined algorithm evaluates the set of mortar nodes S as shown in (a). Applying the proposed algorithm to triangulate the coupling surface represented by these nodes leads to the mesh shown in (b). The triangle edge colors are based on their origin as a NURBS element edge on either the master (blue) or slave (olive) patch boundary surface (red edges are newly introduced). The coloring visualizes that none of the triangles is attached to more than one master or one slave side patch boundary element.

From the nodes that fulfill these three requirements, the algorithm selects the candidate that creates a new triangle with the largest angle enclosed by the two new triangle edges. That node

is found via the expression

$$\hat{\boldsymbol{P}}_m^{(new)} = \underset{a \in S^{(cur,nxt)}}{\arg\min} \left(\frac{\left(\boldsymbol{x}' \left(\hat{\boldsymbol{P}}_m^{(cur)} \right) - \boldsymbol{x}' \left(\hat{\boldsymbol{P}}_m^{(a)} \right) \right)^T \left(\boldsymbol{x}' \left(\hat{\boldsymbol{P}}_m^{(nxt)} \right) - \boldsymbol{x}' \left(\hat{\boldsymbol{P}}_m^{(a)} \right) \right)}{\left\| \boldsymbol{x}' \left(\hat{\boldsymbol{P}}_m^{(cur)} \right) - \boldsymbol{x}' \left(\hat{\boldsymbol{P}}_m^{(a)} \right) \right\|_2 \left\| \boldsymbol{x}' \left(\hat{\boldsymbol{P}}_m^{(nxt)} \right) - \boldsymbol{x}' \left(\hat{\boldsymbol{P}}_m^{(a)} \right) \right\|_2} \right) \tag{4.58}$$

Further triangles are added to the interface mesh by iterating over the triangulation border and repeating the outlined procedure until all mortar nodes in S are part of the mesh and no further triangles can be found that fulfill requirements (1), (2), and (3). An interface mesh resulting from the algorithm is depicted in fig. 4.6 for the previously shown patch coupling example in fig. 4.2.

4.5 Examples

4.5.1 Cantilever beam

This chapter on the weak coupling of NURBS patches is primarily concerned with solid objects. Nonetheless, many aspects are better demonstrated on 2D examples – due to their simplicity and the availability of analytic solutions that can be used to compare the numerical results to.

An idealized cantilever beam subjected to a parabolic traction at its free end serves as the first example. The problem is pictured in fig. 4.7. Timoshenko and Goodier [158] solved it analytically and a number of numerical solutions are discussed in Augarde and Deeks [11].

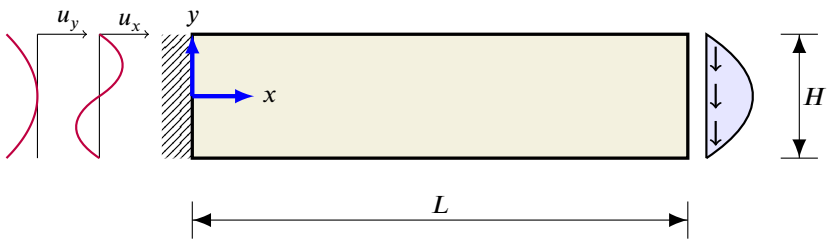


Figure 4.7: Idealized cantilever beam with tip load

For that beam of length L, height H and unit width, the displacement solution under plane stress conditions is, according to [158], given by eqs. (4.59) and (4.60).

$$u_x = \frac{Py}{6EI}\left[(6L - 3x)\,x + (2+\nu)\left(y^2 - \frac{H^2}{4}\right)\right] \tag{4.59}$$

$$u_y = -\frac{P}{6EI}\left[3\nu y^2\,(L-x) + (4+5\nu)\,\frac{H^2 x}{4} + (3L-x)\,x^2\right] \tag{4.60}$$

Therein, P is the integral value of the load applied at $x = L$, E is the Young's modulus, ν the Poisson ratio, and I the second moment of inertia which is calculated as $I = \frac{H^3}{12}$.

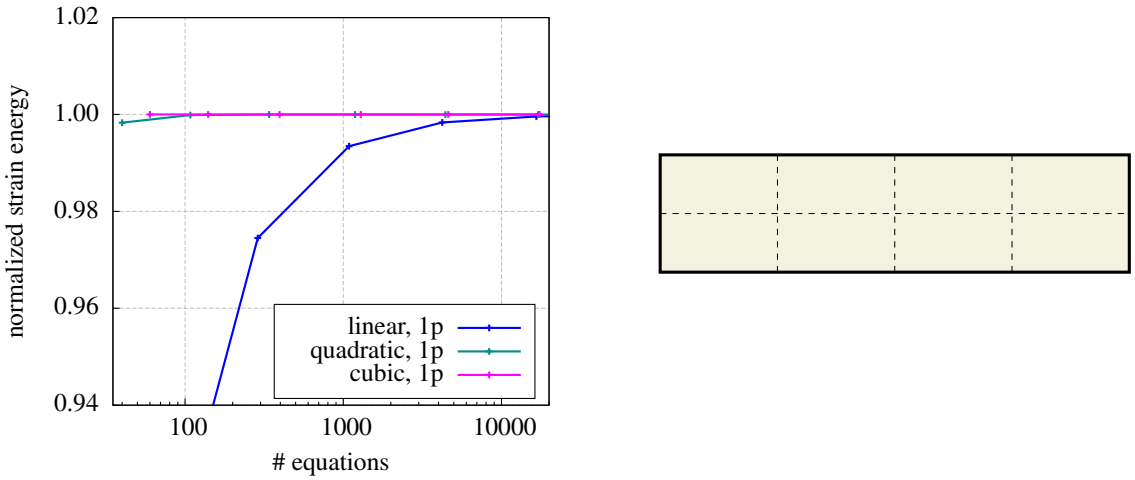
The corresponding stress field is found to be

$$\sigma_{xx} = \frac{P\,(L-x)\,y}{I} \qquad \sigma_{yy} = 0 \qquad \sigma_{xy} = -\frac{P}{2I}\left[\frac{H^2}{4} - y^2\right]. \tag{4.61}$$

When the described problem is to be solved with a 2D FEA while the solution in eqs. (4.59) to (4.61) is to remain valid, the cantilever must not be fully fixed at $x = 0$. Instead, eqs. (4.59) and (4.60) must be prescribed as Dirichlet boundary conditions for $u_x(0, y)$ and $u_y(0, y)$. Likewise, the tip load has to be applied such that it fulfills the solution for the shear stress in eq. (4.61).

For the numerical analysis carried out in this example, the related problem parameters are set to the following academic values: $L = 8$, $H = 2$, $P = 2$, $E = 1000$, and $\nu = 0.25$. With these, the exact strain energy U can be obtained from the analytic solution, it evaluates to $U = 0.5360$.

In order to apply the Dirichlet boundary conditions at $x = 0$, the polynomial functions representing these boundary conditions must be applied on the NURBS patch used to model the beam. Therefore, a NURBS curve whose degree and number of control points correspond to



(a) Strain energy plot

(b) Initial mesh

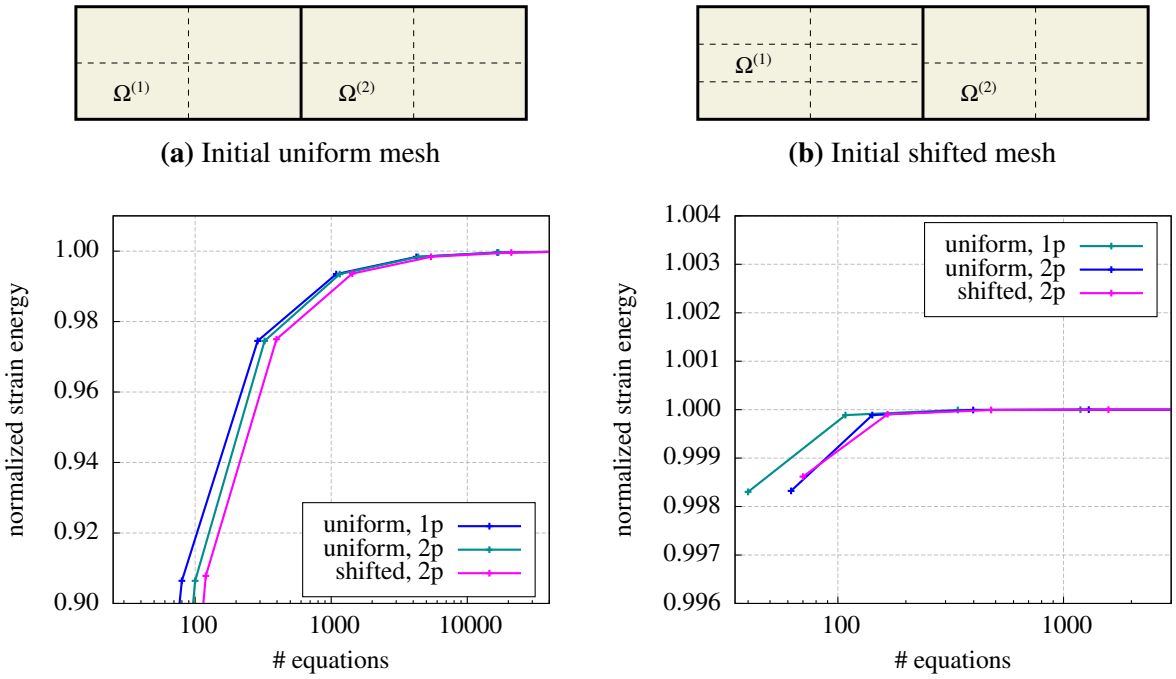
Figure 4.8: Normalized strain energy plot for the cantilever beam problem. A single NURBS patch is used to describe the beam geometry. Results are presented for linear, quadratic and cubic basis functions. Starting from the coarsest discretization shown in (b), the mesh is uniformly refined by subdividing each element of the current refinement step into four child elements of identical size for the next step.

the NURBS patch at the respective boundary is fitted to the polynomial functions in eqs. (4.59) and (4.60).[12] The control point values resulting from the fit are then applied as displacement constraints on the respective control points of the patch. It must be noted that the fitted curve does not comply very well with the true function describing the boundary conditions when the patch is discretized with linear basis functions and a low numbers of elements in y-direction. In this case, the analytically retrieved strain energy U deviates from the strain energy of a numerical solution that is, apart from the boundary conditions, hypothetically approximation error-free.

The problem is initially solved using a single domain which serves as a numerical reference for the subsequent mortar coupling variants. The strain energy results for basis functions with linear, quadratic, and cubic degrees are given in fig. 4.8(a). The results are normalized with the analytical solution. As anticipated, all result curves converge to the analytical solution when the element size decreases. In the cubic case, the results of the coarsest discretization agree completely with the analytical solution. This corresponds to the expectations, as the displacement solution involves at maximum terms of polynomial degree 3. Also for the quadratic case, the strain energy error is negligible for the initial mesh. A variant of the quadratic case is a patch discretization with an artificially introduced C^0 boundary at $x = L/2$. Such a model is most comparable to the two patch mortar coupling shown in fig. 4.9. The results obtained with that model are nearly identical with the standard quadratic case and thus are not plotted.

Results for the cantilever beam partitioned in two subdomains are presented in fig. 4.9. The analysis is performed for two mesh variants, a conforming mesh at the subdomain boundary and a non-conforming one. For the former case, the coupling could have also been realized as strong coupling with constraint equations, however, the mortar method is used in both cases.

[12] Strategies and algorithms for the approximation of point data with NURBS curves can be found in Piegl and Tiller [129, sec. 9.4]



(a) Initial uniform mesh

(b) Initial shifted mesh

(c) Strain energy plot, linear case

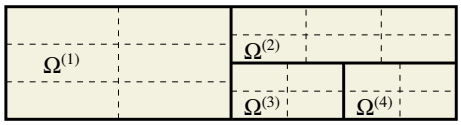
(d) Strain energy plot, quadratic case

Figure 4.9: Normalized strain energy plot for the cantilever beam problem discretized with two patches (2p). The results for linear basis functions are shown in (c), (d) contains results for the quadratic formulation – note the different scales in the two plots. The strain energy curves are evaluated for two different mesh types, a uniform and thus conforming mesh and a shifted, non-conforming mesh, displayed in (a) and (b), respectively. The mortar method is used for the patch coupling of both mesh variants. For an easier comparison, the respective curves for the single patch (1p) results are also shown in these plots.

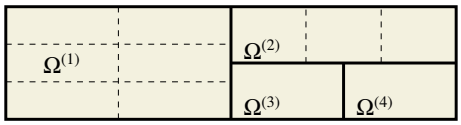
An inspection of the strain energy curves from the linear field interpolations reveals a slight difference between the conforming and the non-conforming meshes with an advantage for the former. In the quadratic case, both mesh types perform alike. They are, as expected, slightly inferior to the quadratic single patch solution, but in general they perform very well and deliver accurate results already for low DOF systems. The results shown in fig. 4.9 are retrieved with the Lagrange multiplier field being interpolated by linear Lagrangian functions. In the quadratic case, results nearly identical to the ones shown are obtained using the inherited NURBS basis functions for the multiplier field interpolation. In the linear case, both interpolation schemes coincide.

In a second mortar coupling investigation, the beam is partitioned in four subdomains which are depicted in figs. 4.10(a) and 4.10(b). The initial mesh for the linear variant given in fig. 4.10(a) is a little finer than that of the initial quadratic mesh, see fig. 4.10(b), in order to avoid single linear elements in any parametric direction.

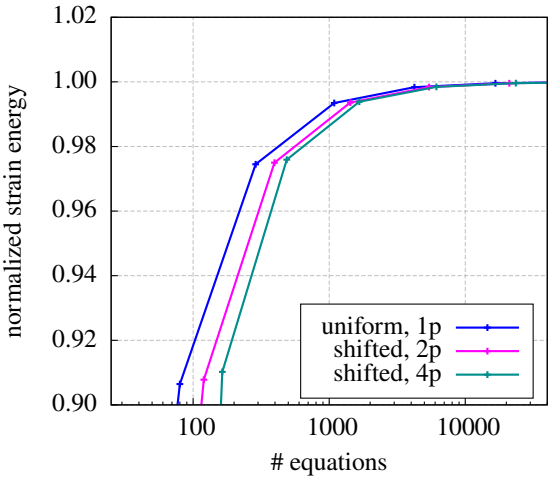
Regarding the results of the four patch partitioned beam, it is observed that the strain energy curve in the linear case is insignificantly inferior to the corresponding curve of the two patch partitioning, which itself is only slightly below the single patch curve. In the quadratic case, the strain energy results are very similar for all analyzed mesh variants. A noticeable difference exists only for the coarsest discretization. This difference results either from the subdomain partitioning or from the type of Lagrange multiplier field interpolation. For the latter, four



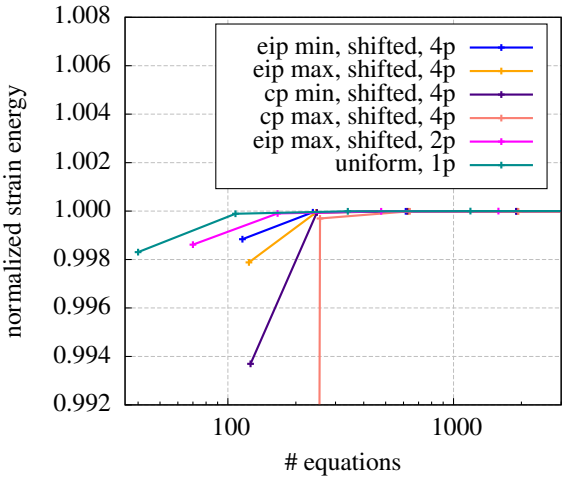
(a) Initial mesh, linear case



(b) Initial mesh, quadratic case



(c) Strain energy plot, linear case

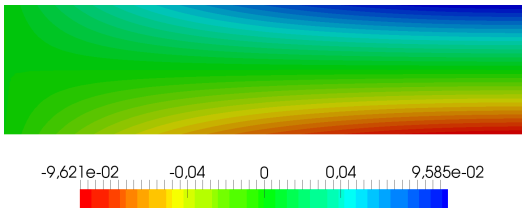


(d) Strain energy plot, quadratic case

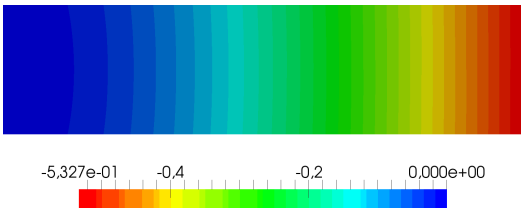
Figure 4.10: Normalized strain energy plot for the cantilever beam problem discretized with four patches (4p). The results are, as before, separated for the linear (c) and (d) quadratic formulations. Again, different scales are used for the two plots. Only a non-conforming mesh is analyzed, yet there are different initial meshes for the linear and the quadratic case. These are depicted in (a) and (b). The mortar method is used for the patch coupling in both mesh variants. The acronyms EIP and CP in the quadratic case strain energy plot refer to the linear Lagrangian (EIP) and the NURBS (CP) interpolation schemes of the Lagrangian multiplier field. The strain energy curves for the single patch (1p) and two patch (2p) discretizations are given to simplify comparisons.

variants are distinguished in fig. 4.10(d): The multiplier field is alternatively interpolated with NURBS and linear Lagrangian functions and for both, the non-mortar side is either selected such that the number of Lagrangian multipliers is minimized or that is maximized. For the NURBS interpolation with a maximized number of multipliers, i.e. *cp max* in the plot, the computation of the finite element solution fails for the coarsest mesh – the normalized strain energy is only at 7.0%. The mathematical problem is overconstrained, which results from too many Lagrange multipliers (38 LMs) compared to the number of unknown degrees of freedom (96 active DOFs). When computing the same example but selecting the non-mortar sides such that the number of LMs is minimized (*cp min*) there are also 96 DOFs but only 30 LMs. With this discretization the problem is solved accurately. The linear Lagrangian interpolation generally leads to fewer Lagrange multipliers. Thus, there is no comparable problem for that interpolation type.

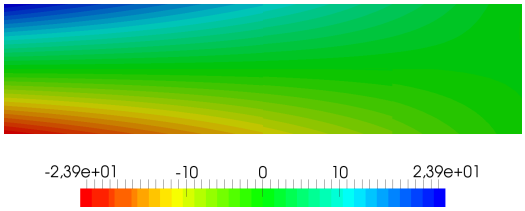
Looking at the plots of the displacement and stress results for the four subdomain quadratic beam problem discretized at the coarsest mesh level (see fig. 4.11) one can see that this discretization is sufficient to obtain a smooth displacement field over the entire domain. Also the σ_{xx} stress field in fig. 4.11(c) is smooth and the location of mortar interfaces cannot be anticipated from the result plot. Yet, this is not the case for the σ_{xy} stress field plotted in fig. 4.11(d).



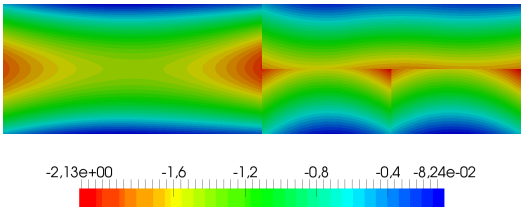
(a) Displacement u_x



(b) Displacement u_y



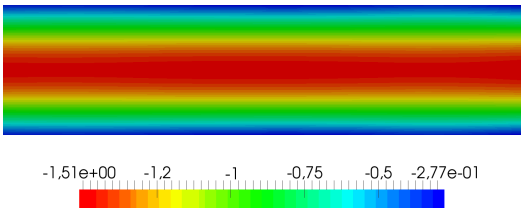
(c) Stress σ_{xx}



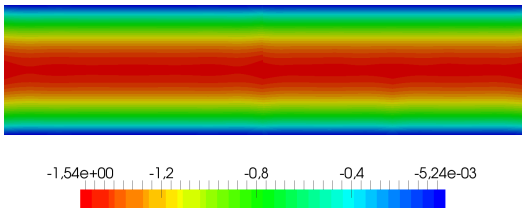
(d) Stress σ_{xy}

Figure 4.11: Displacement and stress results for the cantilever beam problem and the discretization shown in fig. 4.10(b), i.e. for four subdomains, each with quadratic basis functions, with in total 96 active DOFs and 20 LMs stemming from a linear Lagrangian interpolation of the Lagrange multiplier field.

According to eq. (4.61), this field is a quadratic function of y, whereas the σ_{xx} field is only bilinear. The computed range of σ_{xy} stress values with scalar values ranging between -2.13 and -0.1 is acceptable when compared to -1.5 and 0.0 for the analytic solution. Yet, the plotted σ_{xy} stress field looks unphysical and the mortar boundaries are clearly visible. For this situation, the results obtained from the mortar coupling discretization are clearly inferior to the coarsest single patch solution which is displayed in fig. 4.12(a). However, as can be seen from fig. 4.12(b), a similar result quality is also achievable with mortar couplings, it merely requires a higher numerical effort.



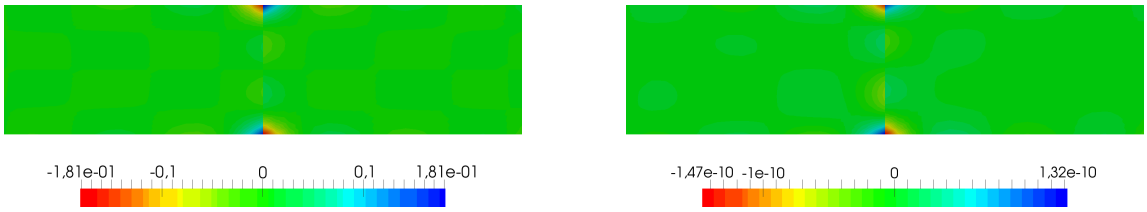
(a) Stress σ_{xy}, single patch



(b) Stress σ_{xy}, four patches

Figure 4.12: Shear stress results for the cantilever beam problem. (a) shows results for a single patch at coarsest refinement (cf. fig. 4.8(b)) with quadratic basis functions. Similar results (b) are obtained with the four patch partitioning when the coarse discretization of fig. 4.10(b) is uniformly refined twice. Then the model has 564 active DOFs and 50 LMs.

Another note is to be made on the influence of the Lagrange multiplier interpolation scheme. A significant advantage of the NURBS basis function interpolation regarding the general result quality is not found when comparing the respective strain energy curves with those obtained for the linear and thus potentially inferior Lagrangian interpolation of the multiplier field. In one case, the NURBS interpolation even proofed detrimental as the resulting system of equations was overconstrained. Yet, it can also be shown, that the higher interpolation quality of the NURBS functions has an advantageous effect in some situations. In fig. 4.13, the σ_{yy} stress



(a) Stress σ_{yy}, EIP

(b) Stress σ_{yy}, CP

Figure 4.13: Stress results for the cantilever beam problem modeled with two patches (cf. fig. 4.9(b)) and cubic basis functions. The difference between the plots is the type of Lagrange multiplier interpolation: Linear Lagrangian interpolation is used in (a) and cubic NURBS interpolation in (b).

fields obtained from the same cubic two patch partitioned problem are compared. That field should be constantly zero over the entire domain and both interpolations deliver acceptable results. When yet the ranges of σ_{xy} stress values at the coupling interface are compared, it is observed that the NURBS interpolation performs several orders of magnitude better than the linear interpolation.

4.5.2 Infinite plate with hole

The second example to be dealt with is the well-known problem of stress concentration around the circular hole in an infinite elastic plate. In this problem, a thin plate with infinite extension and a circular hole of radius R at its center is subjected to an unidirectional tensile stress applied at infinity. The load is aligned with the x-axis, thus it is denoted σ_x^∞. The problem setting is depicted in fig. 4.14.

An analytic solution to the problem is known. In closed form it was first published by Kirsch [94] in 1898. The stress field in terms of polar coordinates is given by eqs. (4.62) to (4.64).

$$\sigma_{rr}(r,\varphi) = \frac{\sigma_x^\infty}{2}\left[\left(1-\frac{R^2}{r^2}\right)+\left(1-4\frac{R^2}{r^2}+3\frac{R^4}{r^4}\right)\cos 2\varphi\right] \tag{4.62}$$

$$\sigma_{\varphi\varphi}(r,\varphi) = \frac{\sigma_x^\infty}{2}\left[\left(1+\frac{R^2}{r^2}\right)-\left(1+3\frac{R^4}{r^4}\right)\cos 2\varphi\right] \tag{4.63}$$

$$\sigma_{r\varphi}(r,\varphi) = -\frac{\sigma_x^\infty}{2}\left[\left(1+2\frac{R^2}{r^2}-3\frac{R^4}{r^4}\right)\sin 2\varphi\right] \tag{4.64}$$

The corresponding displacement field in Cartesian coordinates is then

$$u_x(r,\varphi) = \frac{\sigma_x^\infty R}{8\mu}\left[\frac{r}{R}(\kappa+1)\cos\varphi + 2\frac{R}{r}((1+\kappa)\cos\varphi+\cos 3\varphi) - 2\frac{R^3}{r^3}\cos 3\varphi\right] \tag{4.65}$$

$$u_y(r,\varphi) = \frac{\sigma_x^\infty R}{8\mu}\left[\frac{r}{R}(\kappa-3)\sin\varphi + 2\frac{R}{r}((1-\kappa)\sin\varphi+\sin 3\varphi) - 2\frac{R^3}{r^3}\sin 3\varphi\right], \tag{4.66}$$

with μ being the second Lamé constant (cf. eq. (3.14)) and $\kappa = (3-\nu)/(1+\nu)$.

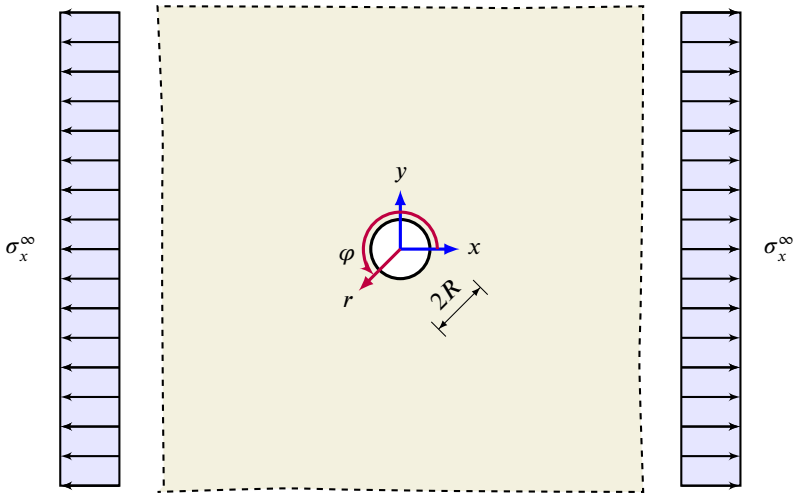


Figure 4.14: Infinite plate with circular hole problem

Only a finite plate can be considered in a numerical model. The tensile stress loading at infinity is incorporated by imposing Dirichlet boundary conditions obtained from the analytical solution instead of applying the actual traction load. Since the problem is symmetric with respect to the x- and y-axis, only a quarter of the plate needs to be modeled. This situation is pictured in fig. 4.15. Using a Young's modulus of $E = 1000$ and a Poisson ratio of $\nu = 0.3$ as the plate's material parameters, $L = H = 4$ and $R = 1$ as geometric dimensions and $\sigma_x^\infty = 10$ for the loading, an exact reference value for the strain energy stored in the plate can be determined from the closed form solution of the stress field. For the given parameters, it evaluates to $U = 0.8445$. The normalized strain energy is a measure for the result quality with global,

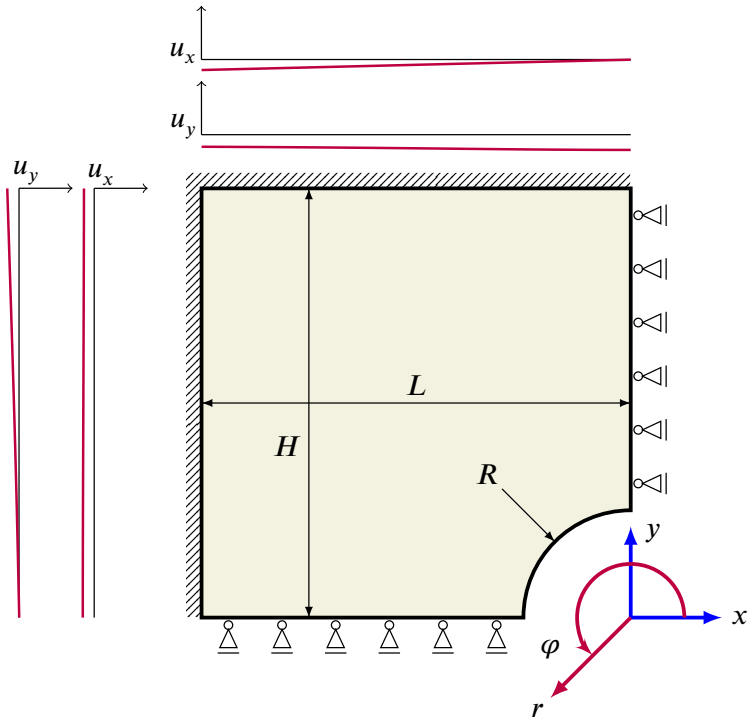
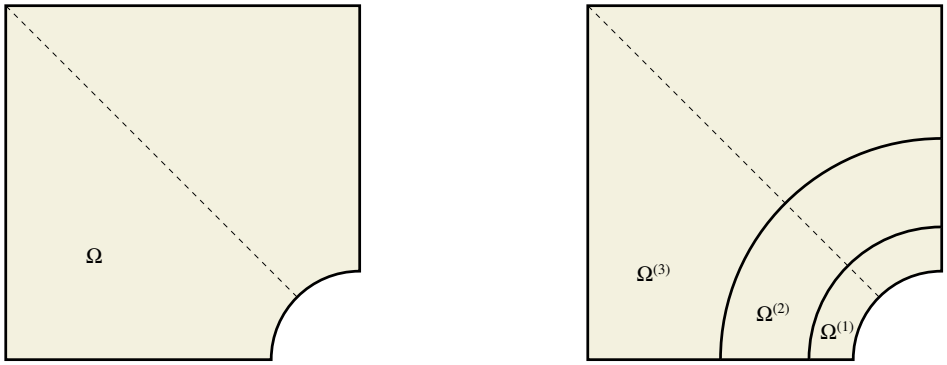


Figure 4.15: Finite quarter plate with circular hole and appropriate Dirichlet boundary conditions, used to model the problem depicted in fig. 4.14.

domain-wide character. In order to also have a measure for local result quality, the known analytical solution of the stress σ_{xx} at $r = 1$ and $\varphi = 1/2\pi$, which evaluates to $\sigma_{xx} = 3\,\sigma_{xx}^{\infty} = 30$, is used for the assessment of the numerical solution.

A single NURBS patch with two quadratic elements is sufficient for the description of the plate's geometry. Numerical results obtained at different mesh refinement levels of such an initial discretization serve as a reference solution for a second model that uses three NURBS patches for the representation of the quarter plate. The three patches are coupled with mortar constraints. Initial meshes of the two variants are pictured in fig. 4.16.



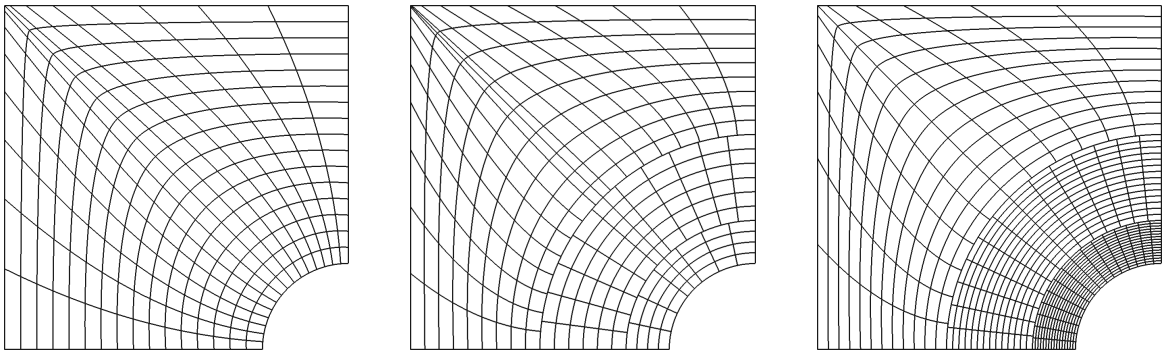
(a) Initial mesh, 1 patch **(b)** Initial mesh, 3 patches

Figure 4.16: Subdomain partitioning and initial element discretization of the infinite plate with circular hole example.

For this example, the solution field gradients are expected to be larger the smaller the radial coordinate is, which would indicate the creation of smaller elements in the vicinity of the hole. For the single patch reference, this would only be possible for the element dimension in radial direction. A refinement in circumferential direction propagates – owing to the tensor product structure of NURBS – through the entire patch. Therefore and for the reason of a better comparison with the three subdomain variant, only the uniform refinement in both directions is considered for the single patch. Owing to the radial patch geometry however, also the uniform refinement tends to create smaller elements near the hole. For the three patch discretization with mortar couplings, two refinement strategies are pursued. One, that approximately matches the uniform refinement of the single patch reference and a second strategy that creates finer meshes for the subdomains nearer to the center of the plate and thus matches the a priory considerations regarding the element size distribution. The three refinement variants are pictured in fig. 4.17.

In addition to the influence of these refinement strategies, the effect of selecting either the coarse or the fine boundary discretization as non-mortar side is investigated for the adapted mesh variant shown in fig. 4.17(c). Selecting the interface side with the fine mesh as non-mortar side, increases the number of Lagrange multipliers and thus the number of unknowns but should also improve the quality of the Lagrange multiplier field interpolation compared with the contrary selection.

Figures 4.18 to 4.21 show the results for this example. The first three basically contain the same data, just differently processed. Strain energy and stress results for basis functions with quadratic, cubic and quartic degrees grouped by their mesh refinement variant are visualized in fig. 4.18. Figures 4.18(a) and 4.18(b) picture the numerical reference solution obtained with



(a) Uniform refinement, 1 patch

(b) Uniform refinement, 3 patches

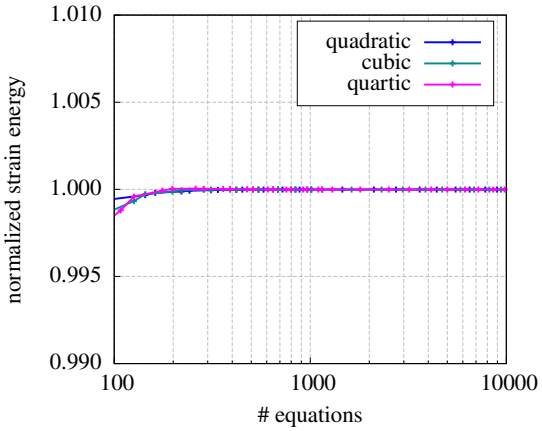
(c) Adapted refinement, 3 patches

Figure 4.17: Mesh refinement variants for the infinite plate with circular hole example. The uniform refinement of the single patch is shown in (a), the corresponding (approximate) uniform refinement of the discretization with three subdomains is given in (b), and (c) pictures the three patch variant with smaller elements near to the hole, later referred to as adapted mesh.

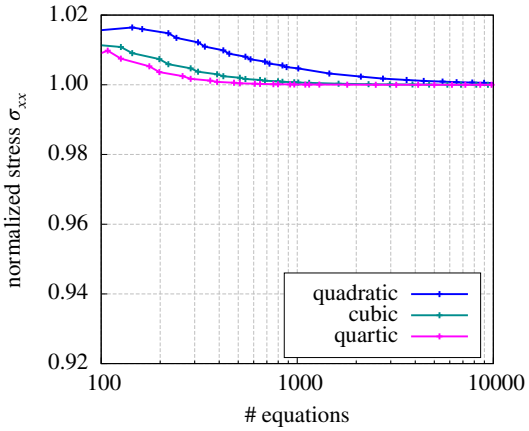
the single patch discretization, and at least the stress results correspond to the expectations: Higher degree basis function solutions converge faster than lower ones but all results converge gradually to the analytical solution. Regarding the single patch strain energy plot, there is no relevant difference to observe between the three curves with their respective degree of the basis functions. All three converge already at a very coarse discretization to the true solution.

The reference solution is to be compared with the results pictured in figs. 4.18(c) and 4.18(d), which are obtained from the uniformly refined three subdomain discretization. It is to note that for coarse meshes the local stress results from cubic and quartic basis functions are markedly inferior to the respective single patch results. A comparable behavior can be observed for the strain energy curves of all basis function degrees, yet the difference on a percentage basis is not as significant. In general, it is to note that quadratic basis functions perform better in coarse discretizations than the cubic and quartic counterparts, which only turns into the opposite after attaining a distinct, degree dependent refinement level. Very similar observations can be made for the results of the three patch discretization with adapted mesh sizes (figs. 4.18(e) and 4.18(f)). It is assumed that for too coarse discretizations, the coupling tractions at the mortar interfaces are not sufficiently well interpolated, which then accounts for the observed behavior. Also, it must be born in mind that the Dirichlet boundary conditions (cf. eqs. (4.65) and (4.66)) contain trigonometric terms as well as terms of order 3. Their "exact" interpolation with NURBS requires a certain refinement level at the boundary. This may also be responsible for some of the result fluctuations seen at coarse refinement levels.

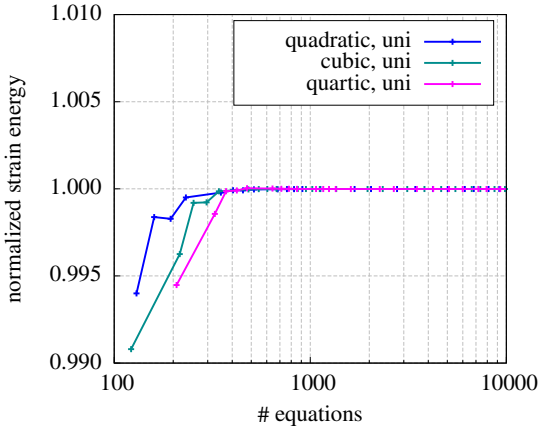
The same data as shown in fig. 4.18 is also presented in fig. 4.19, but now the curves are grouped by their associated degree of the basis functions. This allows to better investigate the influences of the refinement strategy and the mortar couplings on the results. Regarding the strain energy, the uniformly refined single patch variant performs better than any of the discretizations with mortar couplings. Especially in the cubic and quartic cases, which are depicted in figs. 4.19(c) to 4.19(f), the discretizations with mortar couplings behave similar and jointly distinct to the single patch variant. After reaching a specific refinement level, a relative sharp bend in the mortar result curves can be observed. For the stress result, the superiority of the single patch



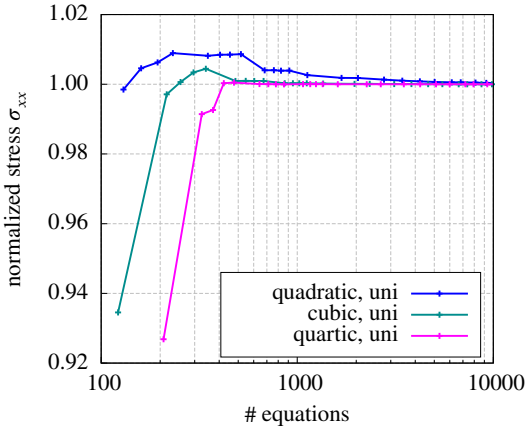
(a) 1 patch – strain energy



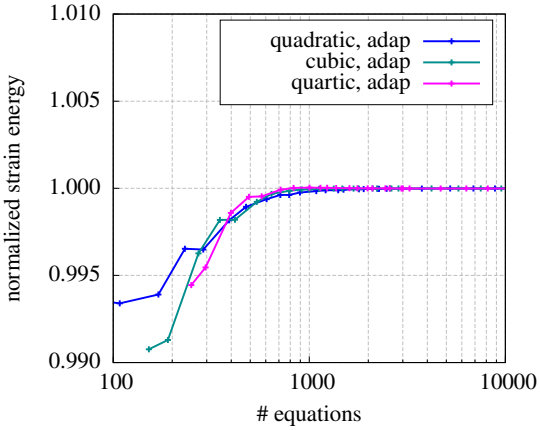
(b) 1 patch – stress



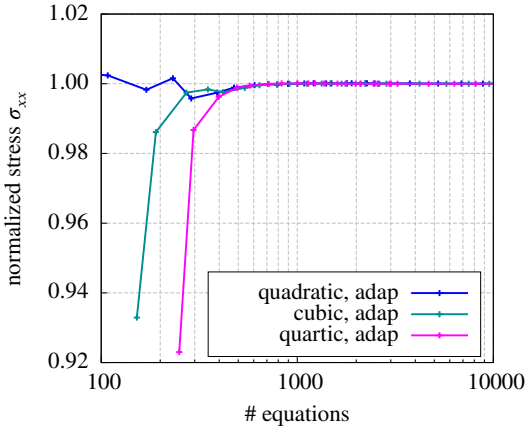
(c) 3 patches, uniform – strain energy



(d) 3 patches, uniform – stress

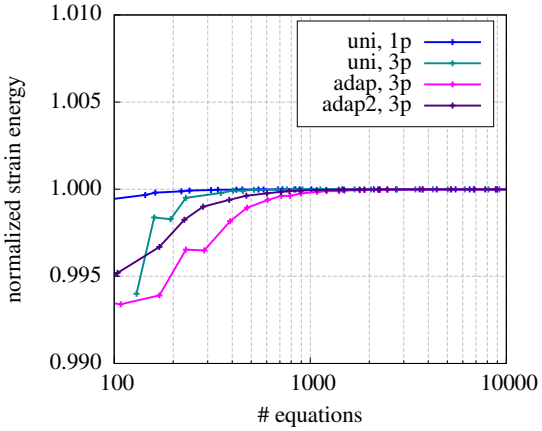


(e) 3 patches, adapted – strain energy

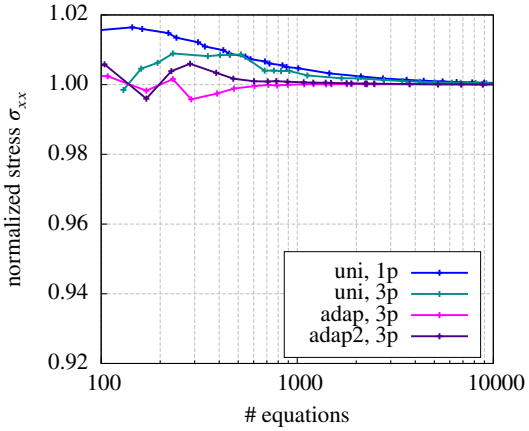


(f) 3 patches, adapted – stress

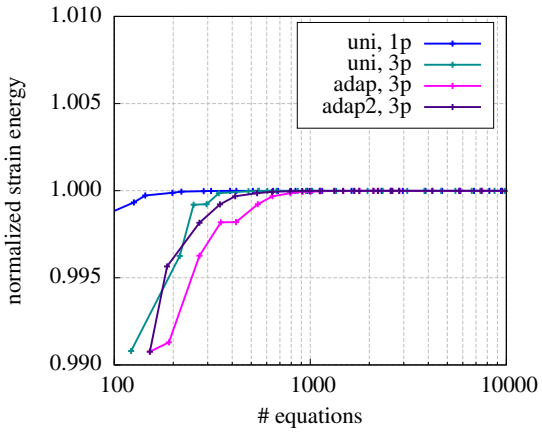
Figure 4.18: Result plots for the infinite plate with circular hole example visualizing the normalized strain energy and the normalized stress $\sigma_{xx}(r = 1, \varphi = 1/2\pi)$. Each plot in this figure contains the results obtained for quadratic, cubic and quartic basis functions at different mesh refinement levels for one of the three discretization variants shown in fig. 4.17.



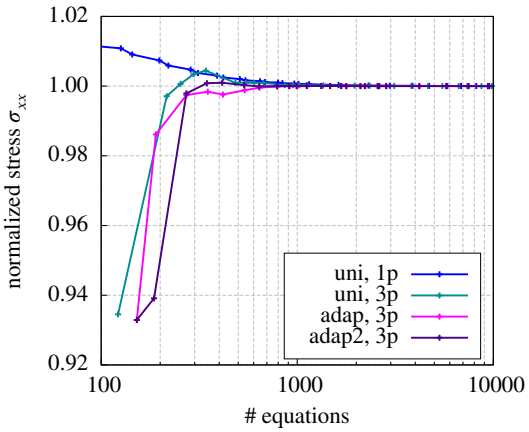
(a) Quadratic – strain energy



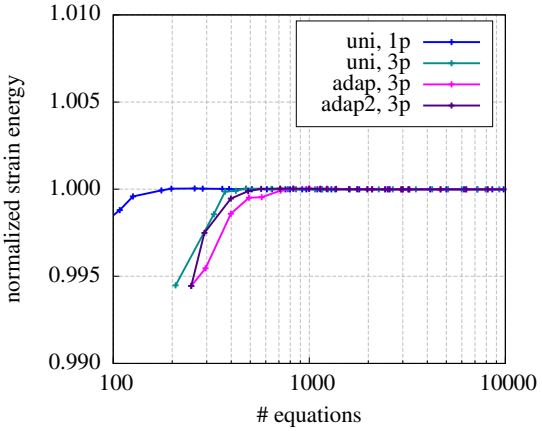
(b) Quadratic – stress



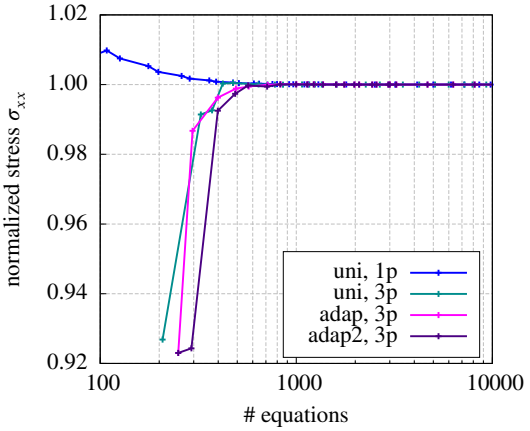
(c) Cubic – strain energy



(d) Cubic – stress

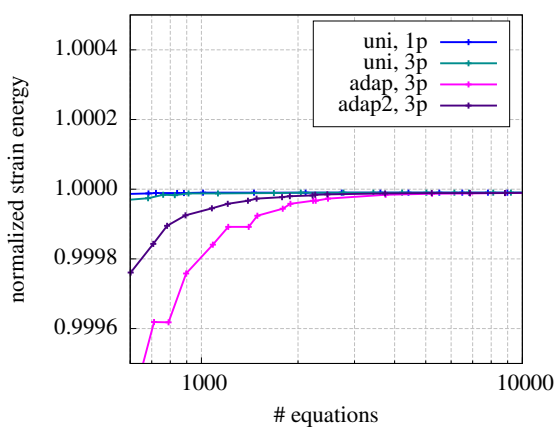


(e) Quartic – strain energy

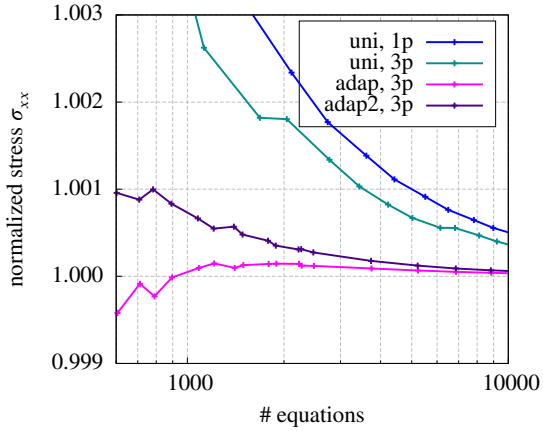


(f) Quartic – stress

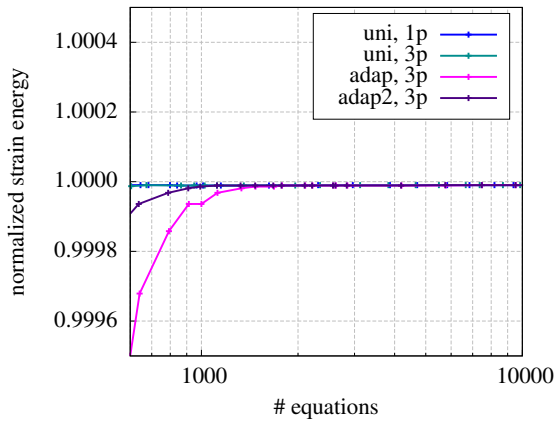
Figure 4.19: Result plots for the infinite plate with circular hole example visualizing the normalized strain energy and the normalized stress $\sigma_{xx}(r = 1, \varphi = 1/2\pi)$. Each plot in this figure contains the results obtained at different refinement levels for each of the three mesh variants shown in fig. 4.17 at a fixed degree of the NURBS basis functions. The results contain the additional mesh variant *adap2* that represents an intermediate version of figs. 4.17(b) and 4.17(c).



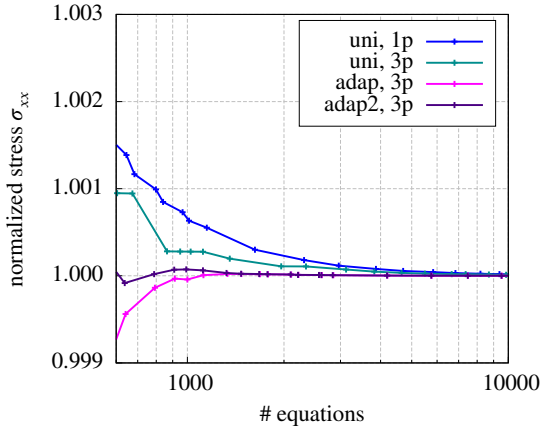
(a) Quadratic – strain energy



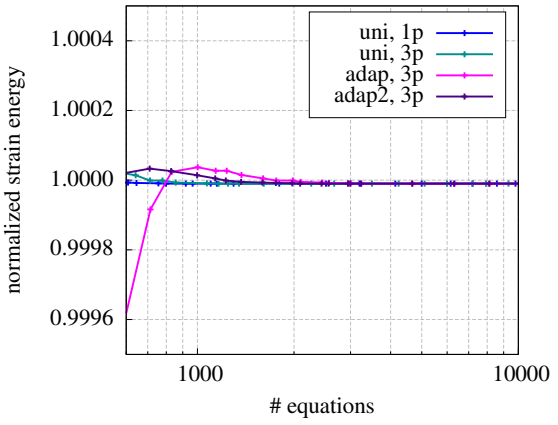
(b) Quadratic – stress



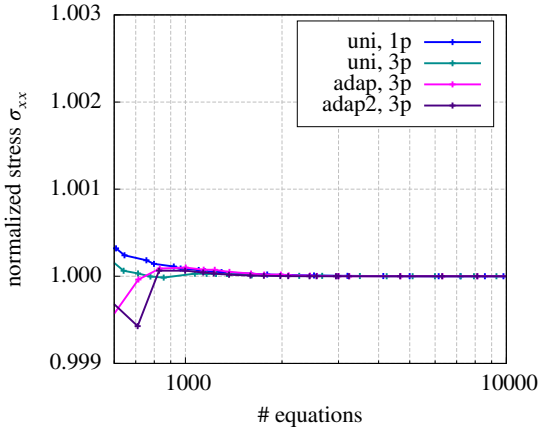
(c) Cubic – strain energy



(d) Cubic – stress

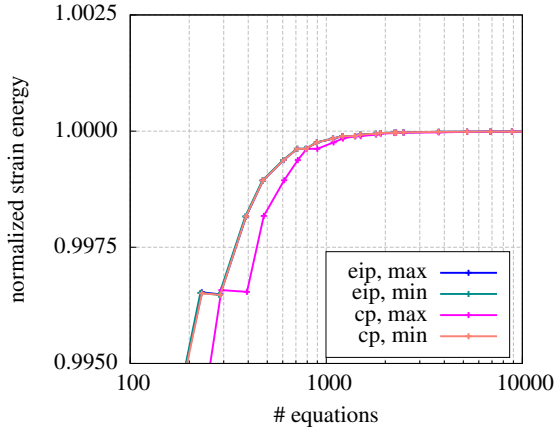


(e) Quartic – strain energy

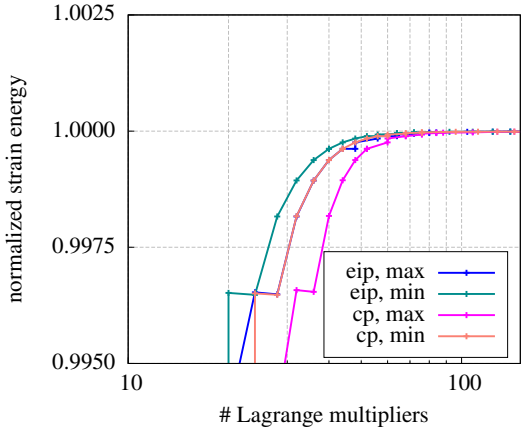


(f) Quartic – stress

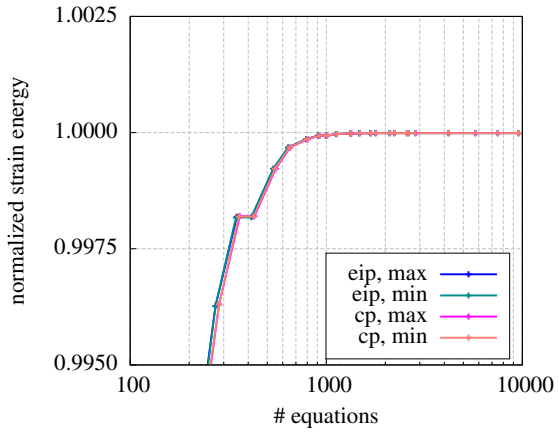
Figure 4.20: Result plots for the infinite plate with circular hole example visualizing the normalized strain energy and the normalized stress $\sigma_{xx}(r = 1, \varphi = 1/2\pi)$. Each plot in this figure contains the results obtained for different refinement levels of each of the three mesh variants shown in fig. 4.17 at a fixed degree of the NURBS basis functions. These plots have the same content as those in fig. 4.19 but the scale is different to enlarge the relevant sections of the previous plots.



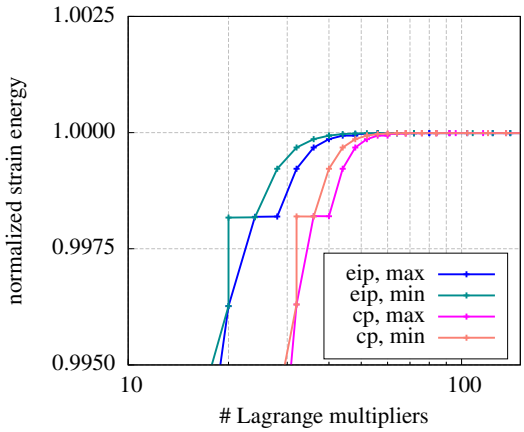
(a) Quadratic – strain energy



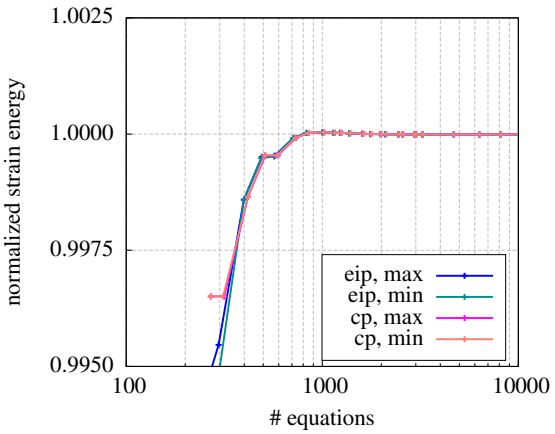
(b) Quadratic – stress



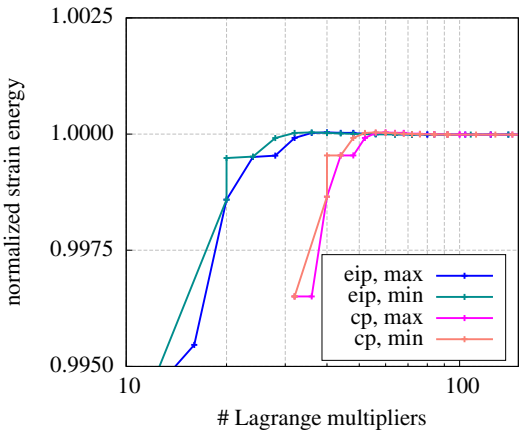
(c) Cubic – strain energy



(d) Cubic – stress



(e) Quartic – strain energy



(f) Quartic – stress

Figure 4.21: Result plots for the infinite plate with circular hole example visualizing the normalized strain energy. Each plot in this figure contains the results obtained for different interpolation schemes of the Lagrange multiplier field at a fixed degree of the NURBS basis functions and the discretization shown in fig. 4.17(c). The left column plots these results over the number of equations, whereas in the right column, they are plotted over the number of multipliers.

solution is not as obvious. After overcoming the initial shortcoming of a too coarse mortar interface discretization, the three patch variants seem to perform equally well or better.

Note that all plots in fig. 4.19 contain a fourth mesh variant that represents an intermediate state between the uniform (fig. 4.17(b)) and the adapted refinement (fig. 4.17(c)). It is investigated to address a situation where the mesh produced by the adapted refinement might be to coarse at the Dirichlet boundary to correctly apply the boundary conditions. And indeed, the global behavior (strain energy) improves while the local behavior (stress at the hole) deteriorates when comparing the *adap2* variant with the general *adap* mesh refinement strategy for coarse discretizations.

In order to investigate the behavior after the bend more thoroughly, this part of the curves is plotted again at an increased resolution in figs. 4.20(c) to 4.20(f). It is assumed that these curve sections represent a state at which the mortar interfaces are sufficiently fine discretized. For the strain energy curves it can be observed that the uniformly refined single patch solution nearly coincides with the solution of the also uniformly refined mortar coupled three subdomain discretization. Both perform slightly better than the adapted mesh variants, which have their DOFs concentrated around the hole. The opposite behavior is observed for the local stress results: The adapted mesh variants show a better performance than the uniform refinement cases – with and without mortar couplings.

Prior to reaching the refinement level associated with the bend, the result behavior is clearly dominated by the mortar couplings. The effect seems to increase with the degree of the basis functions. With the displacement solution being of quadratic order, the associated strain and stress fields are linear over each subdomain. The Lagrange multiplier field – denoting the coupling tractions on the subdomain interfaces – is interpolated by linear Lagrangian functions for all result plots discussed so far. This implies linear strain and stress fields across subdomain interfaces. As the same level of gradient field interpolation is achieved with the single patch discretization and quadratic basis functions, the less pronounced effect in the quadratic case is not particularly surprising.

The field of Lagrange multipliers is interpolated with linear Lagrangian shape functions for all plots in figs. 4.18 to 4.20. Also, in all these cases, the non-mortar side is selected such, that a maximum number of multipliers is created. The two different Lagrange multiplier interpolation schemes are compared in fig. 4.21. For the adapted mesh variant (*adap*), the effect of selecting either the NURBS or the Lagrangian interpolation is investigated. Furthermore, the two cases leading to either a minimum or a maximum number of Lagrange multipliers is analyzed for both schemes. For these, in sum, four cases, the strain energy results are plotted over the total number of equations in figs. 4.21(a), 4.21(c), and 4.21(e) and over the number of Lagrange multipliers in figs. 4.21(b), 4.21(d), and 4.21(f) for the quadratic, cubic and quartic basis functions respectively. Unexpectedly, all variants coincide when plotted over the number of equations.

Previously, the mortar couplings were held responsible for the global convergence behavior. Improving the interpolation quality, as is done by using the NURBS interpolation, is therefore expected to have an effect on the results. Yet, with the exception of a slight difference in the quadratic case, no relevant influence is observed when analyzing the *strain energy vs. the number of equations* plots in fig. 4.21. Hence, the improved order of interface traction interpolation

does not increase the global result quality. On the contrary, more Lagrange multipliers are required to obtain the same level of result accuracy, as can be seen from the *strain energy vs. the number of Lagrange multipliers* plots in fig. 4.21.

It is to note, the regardless of the order of Lagrange multiplier field interpolation, the numerical integration of the interface is, for this work, always conducted on a linear discretization of the interface. As the interfaces are curved in this example, the integration is associated with an error, which is the larger the coarser the discretization of the adjacent patches. It is reasonably possible that the integration error dominates the effect of the mortar interpolation order and thus, it may be possible to observe a different effect from the interpolation order in other examples.

Summarizing the findings from this example, it is to conclude that mortar couplings work remarkably well for quadratic basis functions at any underlying discretization of the subdomains. Ensuring a minimum discretization of the mortar interfaces, this also applies to the cubic or higher basis function degree variants. When, however, the total number of equations is to be kept low and the result precision is not of the utmost concern, a discretization with quadratic NURBS basis functions and a relatively coarse mesh, appears to be the most adequate choice. For curved patch boundaries and a linearized integration of the coupling interfaces, the type of Lagrange multiplier field interpolation does not seem to influence the solution. Therefore, the variant leading to the least number of multipliers should be preferred, which is the linear Lagrangian interpolation with the coarser patch boundary being selected as the non-mortar side.

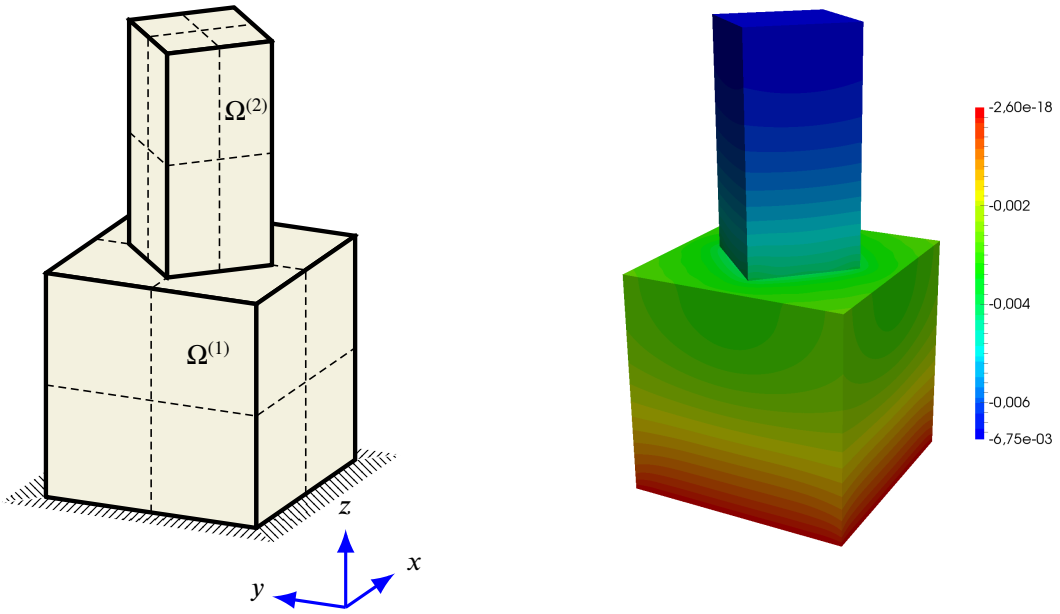
4.5.3 Coupled solid cubes

The coupling of two elastic solid cubes, which already served as an example in sect. 4.4.1, is discussed next. Each cube's geometry is exactly modeled by a single NURBS patch with linear basis functions and only one element. Mortar constraints are applied to couple the two patches in a numerical analysis. The general problem and the original discretization are depicted in fig. 4.22(a). The lower solid is a unit cube, the upper one has plan dimensions of 0.48×0.48 and unit height. With respect to each other, they are rotated by 45° around the z-axis. The two cubes are vertically aligned at the centroid of their base area. Both cubes are subject to a body load of 50 acting in the negative z-axis direction. The lower cube is fully fixed at its base. The cubes' material is defined by the Young's modulus and the Poisson ratio, assumed to be $E = 10^4$ and $\nu = 0.3$.

An analytical solution for this problem is not known and neither is a single patch IGA discretization available for the given geometry. The reference analysis is hence conducted with standard FEM, in particular the ANSYS[13] software package is used. For the related model, the geometry is discretized with elements showing a quadratic displacement behavior.[14] At the highest refinement level, the elements have an edge length of 0.02 and the model contains approximately 5.1 million DOFs. Results obtained with this model are assumed to represent the true solution and thus, all IGA results are normalized to this reference.

[13] Ansys Mechanical APDL, v16.0, ANSYS Inc., www.ansys.com

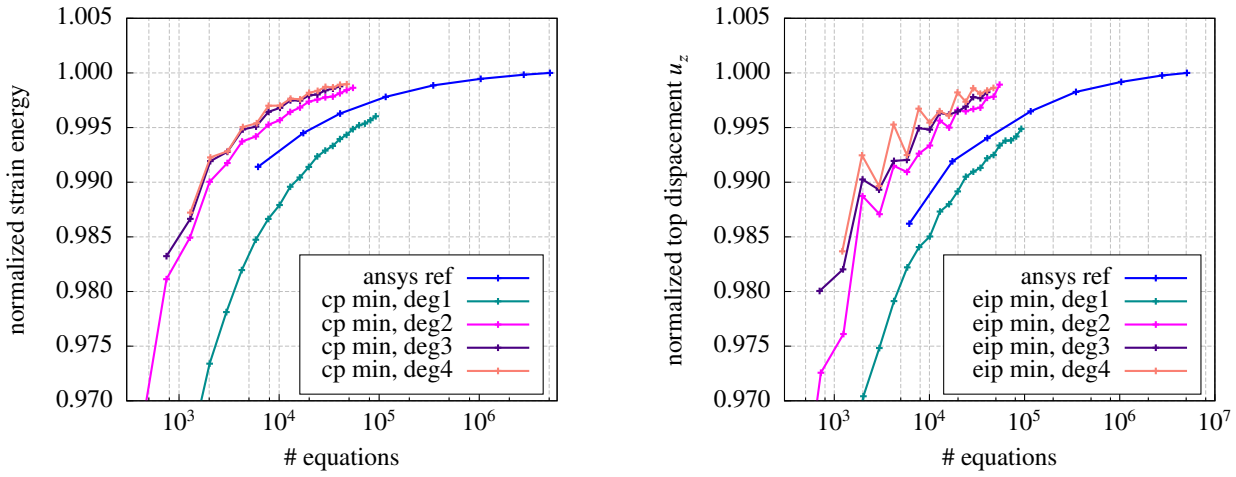
[14] SOLID186 – 20-node structural solid element with quadratic displacement behavior. The keyoptions are set to use a pure displacement formulation. The brick type element collapses to a tetrahedron, pyramid, or prism whenever necessary. In that case, the node count is reduced accordingly.



(a) Coupling problem **(b)** Displacement field u_z

Figure 4.22: The geometry and the initial discretization of the cube coupling problem are depicted in (a). The exemplary result plot in (b) visualizes the displacements u_z obtained from an analysis with 7623 DOFs and cubic basis functions.

For the isogeometric analysis, the original discretization required to represent the exact geometry is refined by knot insertion, resulting in four equally sized elements per patch. The corresponding initial mesh is depicted in fig. 4.22(a). With each subsequent refinement iteration, the function space of the preceding iteration's mesh is enriched by increasing the element count in every coordinate dimension by one. Thereby, the patch uniform element size and shape is preserved throughout the analysis of this example. Owing to the different geometric



(a) Strain energy **(b)** Top displacement u_z

Figure 4.23: Comparison of the results from IGA and FEA. The IGA model is analyzed in four variants with linear to quartic degree basis functions. These results are compared with the reference solution obtained with classical finite elements. The key entries 'cp min' and 'eip min' refer to the type of Lagrange multiplier interpolation for the mortar couplings.

extensions of the two patches, the elements in $\Omega^{(2)}$ are smaller than those in $\Omega^{(1)}$ at any time of the analysis.

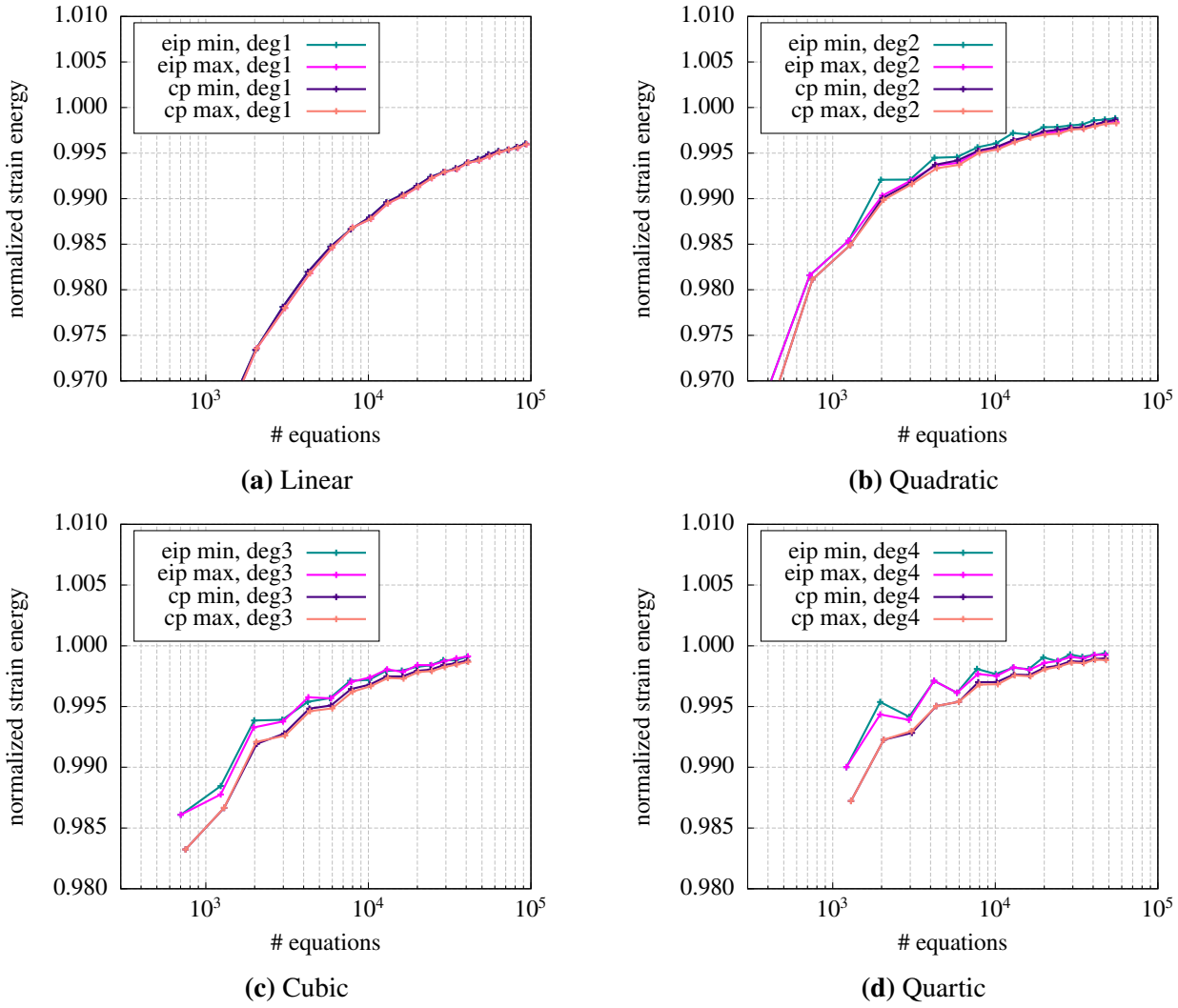
Discretization variants are created by application of the degree elevation technique, leading to four models with linear to quartic basis functions. Normalized strain energy and displacement results obtained from the respective models at different refinement levels are plotted in fig. 4.23. They are compared with corresponding results from the reference model.

Prior to assessing the results in detail, it is to note that in contrast to the previous example, there are no implications expected from the integration of the mortar integrals as the coupling interface is flat and thus, the linear interpolation of the integration cells is sufficient to obtain exact integration results. Also it is pointed out that the initial discretization used for the numerical analysis is very coarse. The results retrieved with this mesh are accordingly somewhat inaccurate, as can be seen from the left-hand side beginning of the plotted result curves. However, in contrast to classical finite elements it is possible to obtain approximate results even with such a coarse discretization. Furthermore, it is emphasized that in this example, like in any other, the complete coupling process is fully automatic. The analysis input is merely the geometry of the two patches, the boundary conditions and the material parameters.

Considering the weak coupling, the strain energy results plotted in fig. 4.23(a) are surprisingly good. Only the linear NURBS interpolation variant results are inferior to those of the reference solution – which, however, corresponds to the expectations: There is no difference between the linear NURBS basis and linear finite elements and hence, quadratic finite elements must perform better than the linear IGA variant that additionally suffers from the weak coupling. Analyses with higher degree NURBS basis functions, however, outperform the reference solution. Furthermore, it is noted that by application of the recovery procedure outlined in sect. 5.5, the quality of the strain energy results from the linear variant can be enhanced to approximately correspond with the reference solution. Summarizing fig. 4.23, it is found that the weak coupling is not detrimental to the quality of global result values and the superiority of the NURBS interpolation over classic finite elements appears to persist also when patches are weakly coupled.

The effect of different Lagrange multiplier interpolation schemes on the results was already investigated for the 2D examples, but no fundamental advantage of either scheme over the other was found. For the cube coupling example, analyses with varying interpolation schemes but otherwise completely identical parameters are conducted to evaluate the impact of the scheme on the results in a 3D situation. For the linear to quartic basis function degree variants, the alternatives of NURBS (CP) and Lagrangian (EIP) interpolations are investigated considering either side of the coupling interface as the non-mortar side and thus as the basis of the multiplier field discretization. The strain energy results are visualized in fig. 4.24, the top displacements in fig. 4.25. The terms ‘min’ and ‘max’ in the key of these plots denote the non-mortar side being selected as to achieve a minimum or maximum number of Lagrange multipliers.

Since the two schemes are identical for the linear variant, it does not surprise to see also completely coinciding result curves in that case. Also the choice of the non-mortar side does not have a relevant effect on the results in the linear case – only a minimal difference between the curves is visible. With an increasing degree of the basis functions, however, a separation of the curves basing on one scheme from those basing on the other can be observed. This distance between the curves, which is especially notable for coarse meshes, shrinks with progressing



(a) Linear **(b)** Quadratic

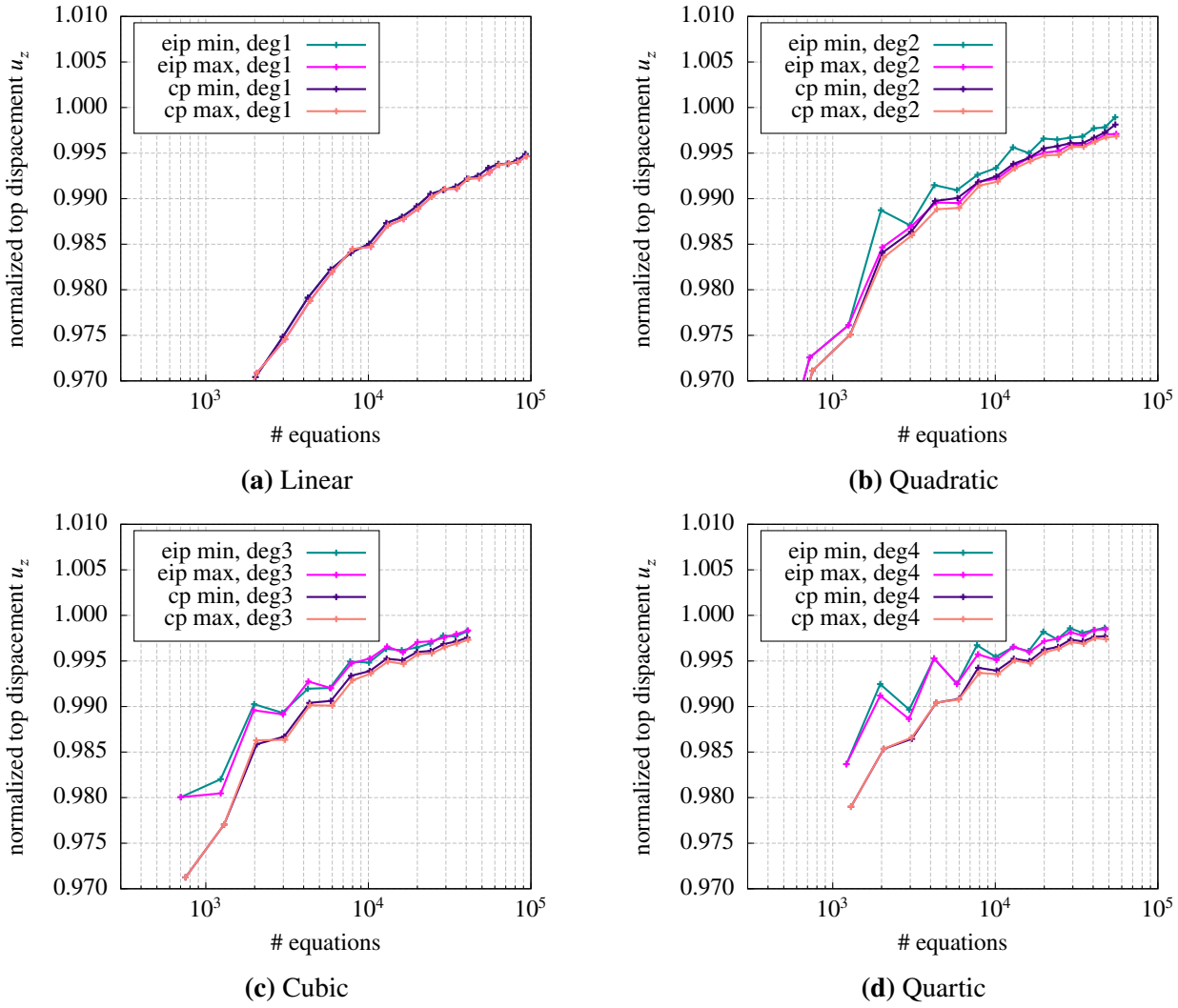
(c) Cubic **(d)** Quartic

Figure 4.24: Strain energy results. The effect of the different Lagrange multiplier interpolation schemes is evaluated, separated by the degree of the NURBS basis functions.

mesh refinement. Surprisingly, the bilinear Lagrangian interpolation performs slightly better than the NURBS interpolation with its supposedly superior field interpolation quality.

Another peculiarity observed from the comparison of the various result curves in figs. 4.24 and 4.25 is an increasing unsteadiness of the curves that accompanies a growing degree of the basis functions. This unsteadiness effect fades with mesh refinement. It is suspected that this behavior is caused by the varying discretization of the underlying patches. With a growing degree of the basis functions, the range of the interface's influence on control points outside the direct coupling boundary extends. Depending on the location of the coupling interface with respect to the elements of the underlying patch, the mortar coupling interferes with more or less control points of the associated patches. When the location of the elements varies as a result of the ongoing refinement, the difference in the number of influenced control points creates the result variations that are visible as small kinks in the plotted curves. However, as the amplitude of the unsteadiness remains small and as it does not influence the overall convergence behavior it is not of practical concern.

A second observation in the context of kinks in the result curves is made during the evaluation



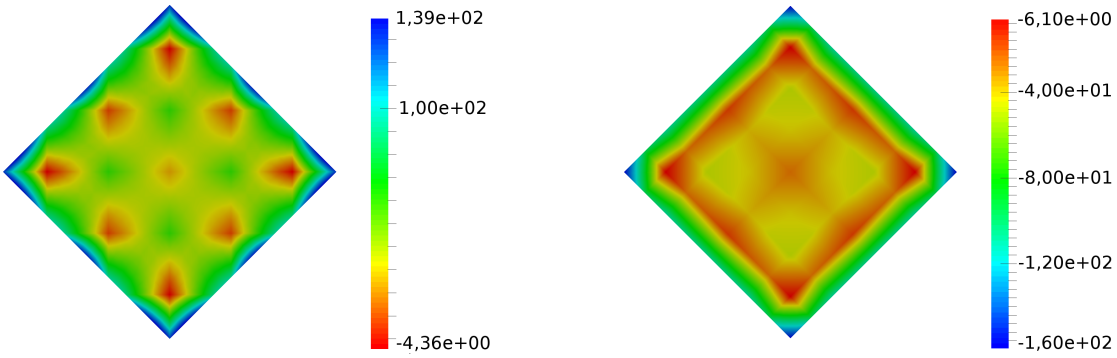
(a) Linear **(b)** Quadratic

(c) Cubic **(d)** Quartic

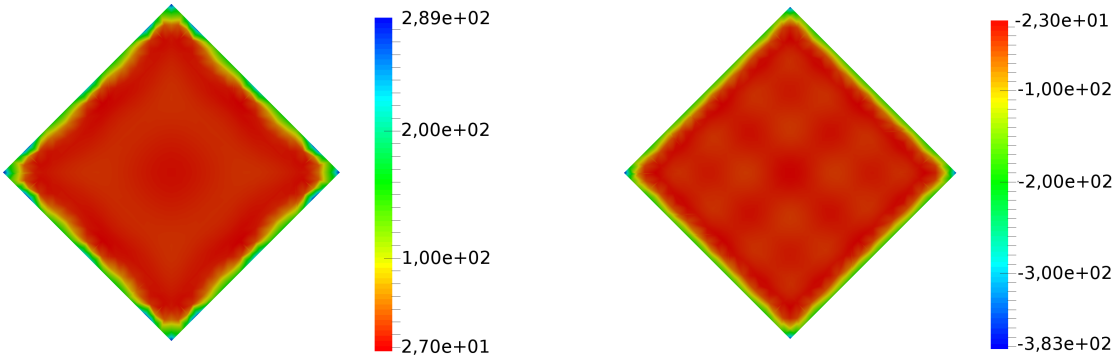
Figure 4.25: Vertical displacement results evaluated at the top of the patch representing domain $\Omega^{(2)}$. The influence of the different Lagrange multiplier interpolation schemes is investigated separately for the different degrees of the NURBS basis functions.

of this example: The accuracy in the determination of corresponding points in the parameter spaces of coupled patches has a direct influence on the results. These pairs of parametric points are required during the evaluation of the mortar matrices. They are found by evaluating the physical point from a parametric coordinate of one patch and then projecting that point into the parameter space of the other patch (cf. sect. 4.4.3.3). Varying the accuracy of the projection between refinement iterations can lead to similar kinks as those visible in e.g. figs. 4.25(c) and 4.25(d).

Next, the discussion returns to the topic of the Lagrange multiplier field interpolations. The influence of the different schemes on local results, i.e. on results in the vicinity of the coupling interface is analyzed. For that purpose, the multiplier fields obtained with the various schemes are visualized in fig. 4.26. An identical discretization of the given problem with 7623 DOFs and basis functions of degree 3 was used in all cases shown in the figure. The number of DOFs results from a very reasonable mesh of 8×8 elements on the two patch faces being in contact. As the Lagrange multiplier field represents the coupling tractions, the comparison of



(a) Bilinear minimum, LM_z **(b)** Bilinear maximum, LM_z



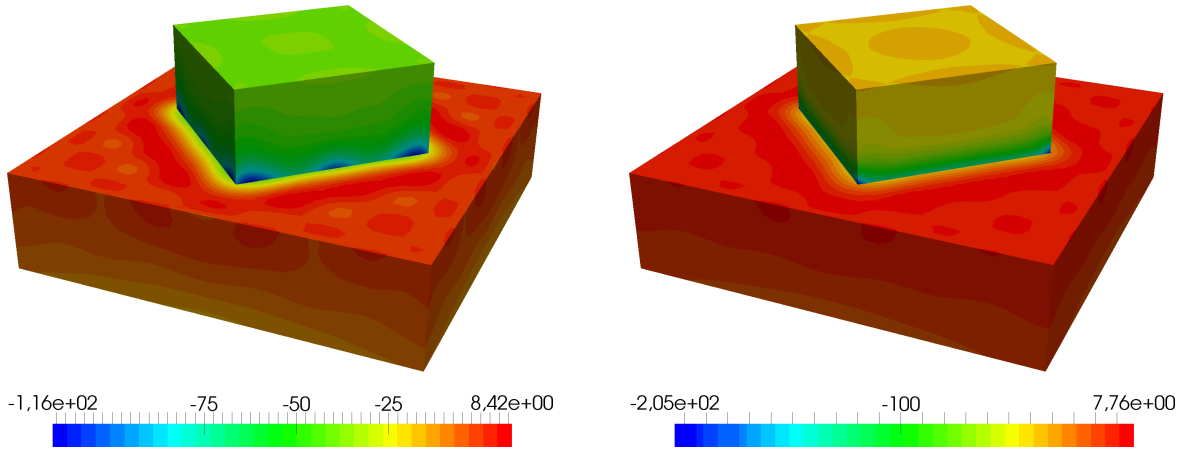
(c) NURBS minimum, LM_z **(d)** NURBS maximum, LM_z

Figure 4.26: Visualization of the Lagrange multiplier field interpolated with bilinear Lagrangian and NURBS functions and both sides of the coupling interface being alternately used as the non-mortar side. The underlying discretization of the given example is fixed at basis function degree 3 and a refinement level corresponding to 7623 DOFs in total. Note that color scales are reversed for the alternating non-mortar sides to produce comparable plots.

the figures in fig. 4.26, allows to conclude that the NURBS interpolation is by a far margin delivering physically more reasonable results.

For the two 'extreme' cases of linear Lagrangian multiplier interpolation with a minimum number of multipliers and the NURBS interpolation with a maximum number of multipliers, the stresses σ_{zz} in the vicinity of the coupling interface are plotted in fig. 4.27. The advanced interpolation with NURBS functions corresponds very well with the results obtained from the classic finite element analysis (not shown). The linear interpolation shown in fig. 4.27(a) does not reproduce the stress singularities at the corners of the upper cube in a comparable quality. As a consequence, the maximum stress values only have about half the magnitude of the other scheme. However, looking inside the cutout and matching the color scale of both plots, as is done in fig. 4.28, reveals that there is no correspondingly large difference in the stress fields of the continuum adjacent to the coupling interface. Nonetheless, the NURBS interpolation remains the superior option with regard to the stresses in the neighborhood of the mortar coupling.

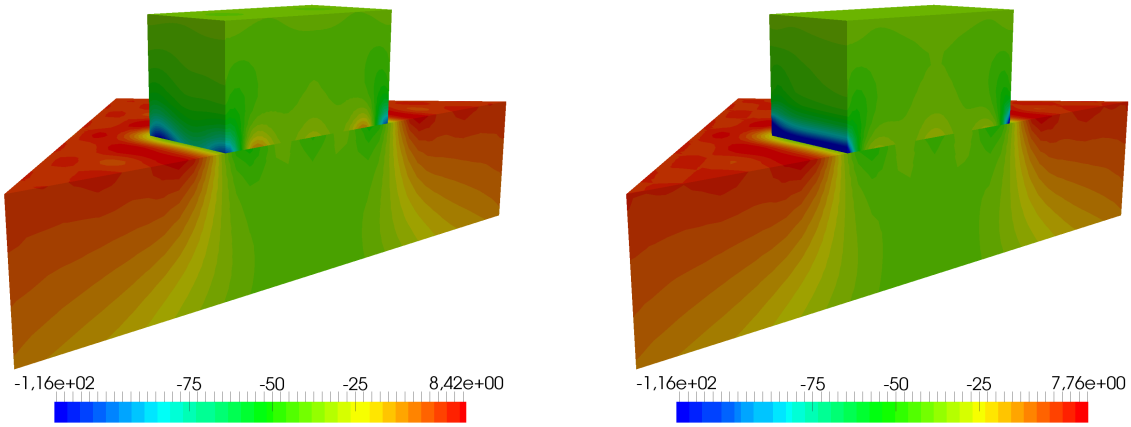
Recalling that a mortar coupling enforces the displacement continuity only in a weak sense requires a comparison also of the displacement results. For that purpose, the difference in the displacements u_z at the coupling interface is pictured in fig. 4.29 for the specific discretization under investigation. Examining the figures therein reveals that a mutual penetration of the two



(a) Stress σ_{zz}, bilinear min interpolation **(b)** Stress σ_{zz}, NURBS max interpolation

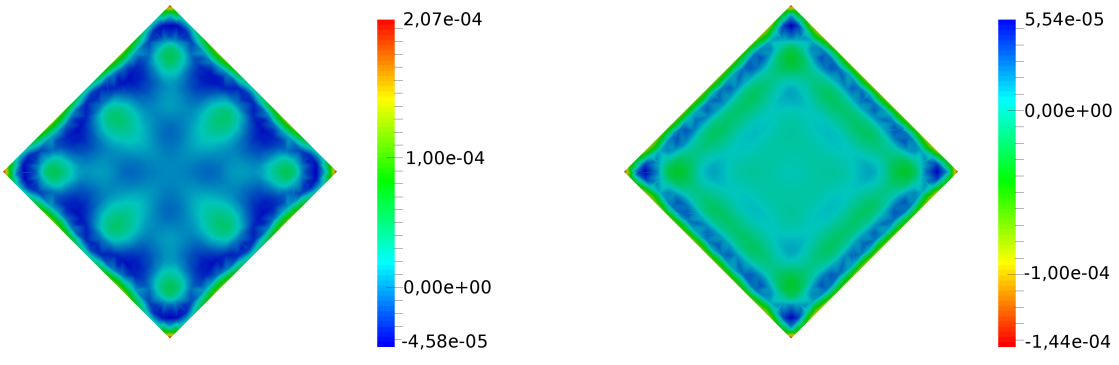
Figure 4.27: Stresses σ_{zz} in the vicinity of the coupling interface for a clipped section of entire geometry. (a) corresponds to the Lagrange multiplier field in fig. 4.26(a) and (b) to fig. 4.26(d).

cubes does indeed take place. The absolute displacement in this region of the cube coupling problem has a magnitude of approximately $4 \cdot 10^{-3}$. The mutual penetration again depends on the Lagrange multiplier interpolation scheme. Its magnitude ranges between $1 \cdot 10^{-4}$ for the linear Lagrangian interpolation with a minimum number of multipliers and $5 \cdot 10^{-6}$ for the NURBS interpolation with a maximum number of multipliers. Again, the NURBS interpolation clearly delivers better results, but also those of the Lagrangian interpolation are acceptable. Interestingly, the mutual penetrations do, just like the Lagrange multiplier fields itself, preserve the symmetry of the original problem. In consequence, the upper solid does not tend to incline in the deformed configuration, what could occur with an asymmetric penetration field.



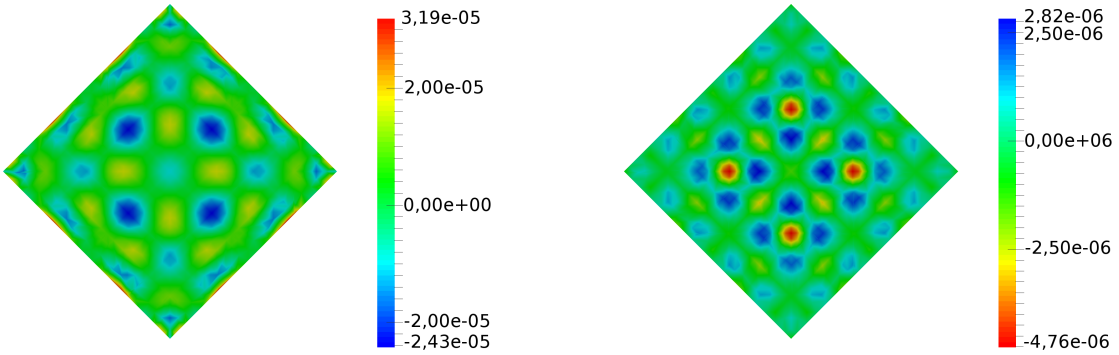
(a) Stress σ_{zz}, bilinear min interpolation **(b)** Stress σ_{zz}, NURBS max interpolation

Figure 4.28: Stresses σ_{zz} inside the clipped sections shown on fig. 4.27. In order to make the plots comparable, the color scale of (b) is limited to the stresses of the example in (a). (a) corresponds to the Lagrange multiplier field in fig. 4.26(a) and (b) to the one in fig. 4.26(d).



(a) Bilinear minimum

(b) Bilinear maximum

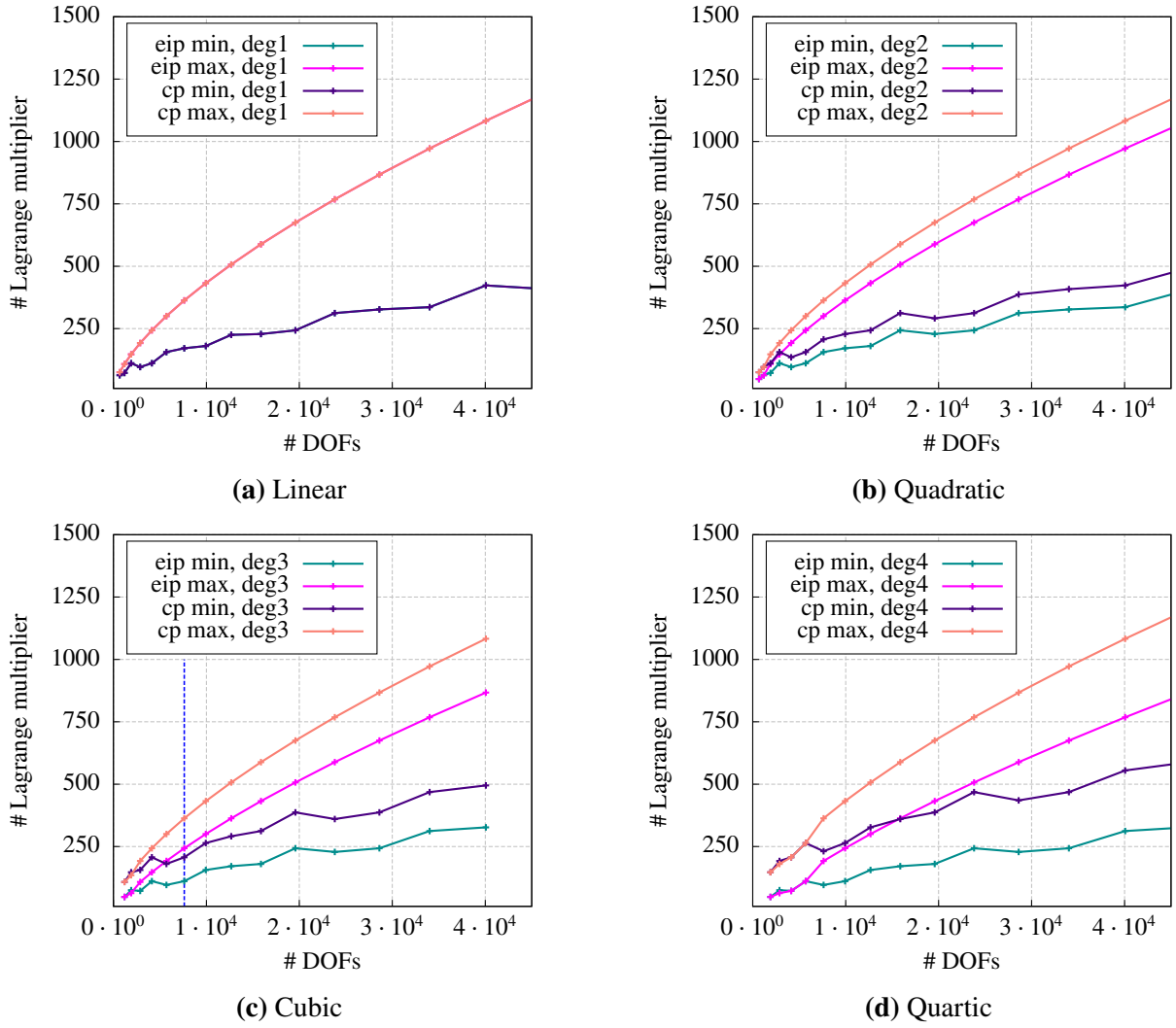


(c) NURBS minimum

(d) NURBS maximum

Figure 4.29: Visualization of the mutual penetration $u_z^{(m)} - u_z^{(s)}$ of the weakly coupled solid cubes. The results are given in dependency of the Lagrange multiplier field interpolation, which is realized with bilinear Lagrangian or NURBS functions and both sides of the coupling interface being alternately used as the non-mortar side. The underlying discretization of the given example is fixed at basis function degree 3 and a refinement level corresponding to 7623 DOFs in total. Note that color scales are reversed for the alternating non-mortar sides to produce comparable plots.

After discussing the advantages and disadvantages of the different interpolation schemes extensively, a final decision on which one to choose remains difficult. The linear Lagrangian interpolation performs better for global result values, whereas the NURBS interpolation shows advantages for the results in the vicinity of the mortar coupling. To support a decision, the development of the number of Lagrange multipliers with progressing mesh refinement is plotted in fig. 4.30. The specific discretization example discussed before is marked in fig. 4.30(c) by the dashed blue line. The effort required for solving the global system grows with the number of Lagrange multipliers, the smallest number preserving the desired result quality is thus desirable. From the figures it can be learned that for coarse meshes the number of Lagrange multipliers is not relevantly influenced be the selection of the non-mortar side. For an increasingly refined mesh with only a moderate difference in element sizes on either side of the coupling interface however, the decision can make a big difference. Furthermore, it is apparent that the difference between the Lagrangian and the NURBS interpolation scheme grows with the degree of the NURBS functions. And finally it can be seen, that the ratio of multipliers to DOFs decreases with the problem size – which, of course, could have also been concluded by geometric considerations.



(a) Linear

(b) Quadratic

(c) Cubic

(d) Quartic

Figure 4.30: Relation between the mesh size and the number of Lagrange multipliers in dependency of the Lagrange multiplier interpolation scheme and the degree of the NURBS basis functions. The dashed blue line in (c) marks the previously discussed discretization variant presented in figs. 4.26 to 4.29.

Based on the discussion of this and previous examples, the suggestion for the use of a specific interpolation scheme is to select the non-mortar side such that the minimum number of Lagrange multipliers is created. The decision for either of the NURBS or Lagrangian interpolation depends on the scope of the analysis. Whenever local result values have a special importance, the NURBS scheme should be used. When, however, the focus is directed towards the global model behavior, like it is in this work, the Lagrangian interpolation scheme is the best option.

Chapter 5

Solution methods for the linear system of equations

5.1 Saddle point problems

The standard procedure of the displacement based finite element method leads to the linear system of equations denoted by

$$\boldsymbol{K}\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{f}} \tag{5.1}$$

where $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ is a symmetric, positive-definite matrix, generally referred to as the global stiffness matrix. The symbols $\tilde{\boldsymbol{u}} \in \mathbb{R}^n$ and $\tilde{\boldsymbol{f}} \in \mathbb{R}^n$ denote the global displacement vector and the global load vector, respectively. The procedure of establishing eq. (5.1) was previously outlined; see sect. 3.3.2 for polynomial shape functions and sect. 3.5.2 for the extension to NURBS basis functions. Since both function types lead to comparable systems of equations, it is not further distinguished between nodal and control point vectors in this chapter and accordingly, eq. (5.1) refers equally to either eq. (3.47) or eq. (3.100).

The previous chapter introduced the coupling of non-conforming subdomain discretizations by the mortar method. The linear system of equations arising from the application of that method on a problem with two subdomains was given in eq. (4.30). In order to formulate a representation for an arbitrary number of subdomains, the stiffness matrices of all subdomains are stored in the block diagonal matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$ and the mortar matrices in the block matrix $\mathbf{B} \in \mathbb{R}^{m\times n}$

$$\mathbf{A} = \begin{bmatrix} \boldsymbol{K}^{(1)} & 0 & & 0 \\ 0 & \boldsymbol{K}^{(2)} & & 0 \\ & & \ddots & \\ 0 & 0 & & \boldsymbol{K}^{(n_{sub})} \end{bmatrix}, \quad \mathbf{B}^T = \begin{bmatrix} \boldsymbol{m}^{(1)^T} \\ \boldsymbol{m}^{(2)^T} \\ \vdots \\ \boldsymbol{m}^{(n_{sub})^T} \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} \tilde{\boldsymbol{u}}^{(1)} \\ \tilde{\boldsymbol{u}}^{(2)} \\ \vdots \\ \tilde{\boldsymbol{u}}^{(n_{sub})} \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} \tilde{\boldsymbol{f}}^{(1)} \\ \tilde{\boldsymbol{f}}^{(2)} \\ \vdots \\ \tilde{\boldsymbol{f}}^{(n_{sub})} \end{bmatrix}.$$

It must be observed that each mortar matrix $\boldsymbol{m}^{(i)}$ contains the terms of all mortar couplings associated with the respective subdomain $\Omega^{(i)}$. The subdomain specific vectors of unknown nodal or control point displacement are stored in $\mathbf{u} \in \mathbb{R}^n$ and the respective force values in $\mathbf{f} \in \mathbb{R}^n$. The Lagrange multipliers of all coupling interfaces are represented by the vector $\boldsymbol{\lambda} \in \mathbb{R}^m$. With these definitions, the generalization of eq. (4.30) for n_{sub} subdomains can be given by

$$\begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix} \qquad \text{with} \qquad \boldsymbol{K} = \begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} \quad \boldsymbol{f} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix} \tag{5.2}$$

Linear systems of equations of the form (5.2) are known as saddle point problems. They arise from various physical or engineering applications, amongst them non-conforming NURBS based finite element discretizations that are weakly coupled by the mortar method with Lagrange multipliers. The matrix $\boldsymbol{K}$, which is further referred to as the stiffness matrix of the problem, remains symmetric, but in contrast to $\boldsymbol{K}$, it is not a positive-definite but an indefinite matrix. As a consequence, the solution techniques commonly applied in structural mechanics cannot be used to solve this type of problem. However, solving these systems is an essential part of the developed framework and therefore, the matter is discussed in this chapter.

It is to be recalled from chapter 2 that the intention of this framework is to base the analysis on the volumetric representations of the individual structural components. Associating each of these structural parts with their own subdomain is a direct consequence of the mortar approach which is used to couple the NURBS patches as the geometric representations of the structural components. Thus, the number of subdomains and accordingly the number of stiffness matrices $\boldsymbol{K}^{(i)}$ in $\mathbf{A}$ is going to be large for real life problems. Though the results presented in chapter 4 show that moderately coarse subdomain discretizations are sufficient to obtain results of acceptable quality and despite the superior approximation qualities of NURBS compared to polynomial element formulations, it is undeniable that solid model analysis results in large systems of equations. Efficient solution techniques for the saddle point problem (5.2) are therefore vital.

A survey on the numerical solution of saddle point problems was compiled by Benzi et al. [29]. The extent of that survey reveals the large number of existing solution techniques. One finding of Benzi et al. is that there is no single best approach, but that individual characteristics of the underlying problem and the matrix properties resulting therefrom must be considered. As none of the approaches presented in [29] is directly suited for the problem discussed here, a strategy specific to the given problem had to be developed. This strategy is presented in the following sections of this chapter. Along with the techniques for the solution of the linear system, this chapter addresses the implemented algorithm for the efficient assembly of the global stiffness matrix and the procedure for the postprocessing of the solution.

Independent of the different approaches that are yet to be discussed, it is unquestionable that solving the linear system (5.2) is more involved than solving the standard case in eq. (5.1). Furthermore, it is acknowledged that there exist weak coupling techniques other than the mortar method that do not lead to a saddle point problem but that will result in a positive-definite global stiffness matrix. Yet, since the numerical analysis of solid models results in large linear systems, they need to be solved in parallel in order to be time efficient. This is especially true in the light of the computer processor development over the past decade. The growth of single thread CPU performance has nearly come to an end. Instead, a given problem is worked on concurrently by multiple cores on a single chip in order to achieve a better performance [156]. Having eight parallel threads on a regular desktop computer is the standard today and up to 244 parallel threads are possible with specialized multicore cards[1] on a single computer. As the general structural mechanics problem (5.1) is not well suited for a concurrent solution process, it is one intention of this work, to exploit the specific block diagonal structure of the matrix $\mathbf{A}$ for that purpose.

[1] Intel Xeon Phi Coprocessor 7100

5.2 Parallel programming

Prior to discussing the different approaches for an efficient assembly and solution of the linear system of equations, a brief digression to the topic of parallel programming is to be made. Traditionally, software applications are executed by a single processor in a sequential manner and without a time overlap of individual tasks. Neglecting instruction-level parallelism, it can be said that the order of statements in the code determines the order of their execution and the speed of processing the individual tasks determines the execution time of the entire application. Here, the term parallel programming is associated with concepts that allow the execution of individual tasks simultaneously on multiple processing units. Quite a few programming techniques exist for that purpose; many of them are related to a specific hardware type.

Hardware

Current general purpose computers are usually symmetric multi-processor (SMP) systems. They have multiple general purpose processing units that are integrated on a single chip. Often, each processing unit is referred to as a core and the entire chip as the CPU. It is possible for a computer system to have several of these chips, which is then denoted a multi-socket system. All cores of an SMP system are identical, they can all access the same random access memory (RAM), which is the main memory of the system, and they are able to work independently on a sequential stream of instructions. The individual cores access the main memory through the system bus. The access time can thereby vary from core to core, depending on the location of the main memory in relation to that specific core (non-uniform memory access (NUMA) vs. uniform memory access (UMA) systems). In multi-socket systems, there is usually a block of the main memory associated with each socket. When a core from one socket accesses main memory that is associated with another socket, it is slower than accessing the memory associated with the core's own socket. In general, however, any direct memory access through the system bus is relatively slow. For this reason, there are several hierarchic stages of additional RAM that are local and private to a single or a group of cores. This private memory, which the core does not access through the system bus, caches the data from the main memory currently worked on by the specific core. When several cores work simultaneously on the same data, the processor design has to ensure, that the data in the core-local cache RAM is consistent. This is referred to as cache-coherent non-uniform memory access (ccNUMA). The overhead from the necessary consistency checks can seriously reduce, eliminate or even reverse any time advantages that result from the parallel processing of computational tasks that rely on the same data. When the involved cores are from different sockets, the performance will even be worse, as the data exchange uses the slow, non-uniformly connected main memory.

Contrasting the SMP architecture, there are also distributed memory processing (DMP) systems. Each building block of such a DMP system can only access its local main memory and all blocks run independently on their own operating system. DMP systems are often created from a number of SMP systems linked by some kind of interconnect that allows the exchange of data between the individual units. Though there are advanced network technologies, like e.g. 12x EDR InfiniBand, whose theoretical data exchange rates are higher than those of a regular system bus, there is no hardware that organizes the data exchange or that performs consistency checks. Instead, the data must be actively distributed by software running locally on all cooperating building blocks of the DMP system.

Another hardware type that is relevant for parallel computing are vector processing units. Their main purpose is to simultaneously process one instruction on different data (SIMD – single instruction multiple data). For example, they can add two vectors at once instead of adding the elements of the vector sequentially. Though general purpose CPUs in SMP machines have a limited vector processing capability, they are no true vector processors. Those are often found in systems that are designed for a specific task. When simplifying matters, one can also regard graphics processing units (GPUs) as vector processors.

Software

The message-passing interface (MPI) [74] is an application programming interface (API) that is closely related to parallel programming on DMP systems. It provides means for the communication and synchronization of independent processes running on different processors on shared memory or distributed systems. Due to its portability, the high-performance design and its scalability, it has effectively become the standard for message-passing in parallel programming. Its scope of application are mainly large-scale computations as can be found in science and industry research and development.

Message-passing is not necessary on shared memory systems, when different processes were allowed to access the same data in the main memory. Operating systems, however, by default preclude independent processes to do so for security reasons. The open multi-processing (OpenMP) API [46] was designed to yet enable and facilitate parallel computing on shared memory platforms. OpenMP parallelism does not base on independent processes; instead, there is a single sequential process denoted the master thread that spawns a team of (sub-) threads whenever a parallel execution of computational tasks is requested in the code. When the slave threads finish their assigned work, they re-unite with the master thread which then continue its sequential execution. The slave threads live within the runtime-environment of the master thread. They can access and modify the data of the master thread – if requested – but can also have private data and share only computed results when finishing the parallel execution. This behavior is achieved by calling runtime functions and by placing compiler directives in the code. The OpenMP API has been implemented by many C/C++ and FORTRAN compilers, rendering OpenMP portable and nearly platform-independent. Recently[2], the OpenMP specifications were extended to also support heterogeneous parallelization with accelerators like GPUs or Intel's many core cards[1] that are directly attached to an SMP system.

Considering the scope of application for the framework implemented in this work, the OpenMP parallelization on SMP systems was selected as the parallelization model of choice. The hardware used for the execution of the implemented software is a two-socket Intel system with Intel Xeon e5-2650 v2 processors and 64 GB 1866 MT/s main memory. The topology of that system is described by fig. 5.1. Tests were also conducted with GPU accelerators, however, these could not satisfy performance expectations and furthermore, they have the big disadvantage of being only available on selected systems, thus rendering the parallel code not as portable. The practical implementation of OpenMP parallel applications is significantly simpler than using MPI. However, in order to achieve a good parallel scaling, the layout of parallel algorithms and the distribution of data to individual threads or processes has to be considered carefully with both models.

[2] At the time of writing, the latest specifications are OpenMP 4.0, released in July 2013, cf. www.openmp.org. Compiler implementations with accelerator support are, however, yet limited.

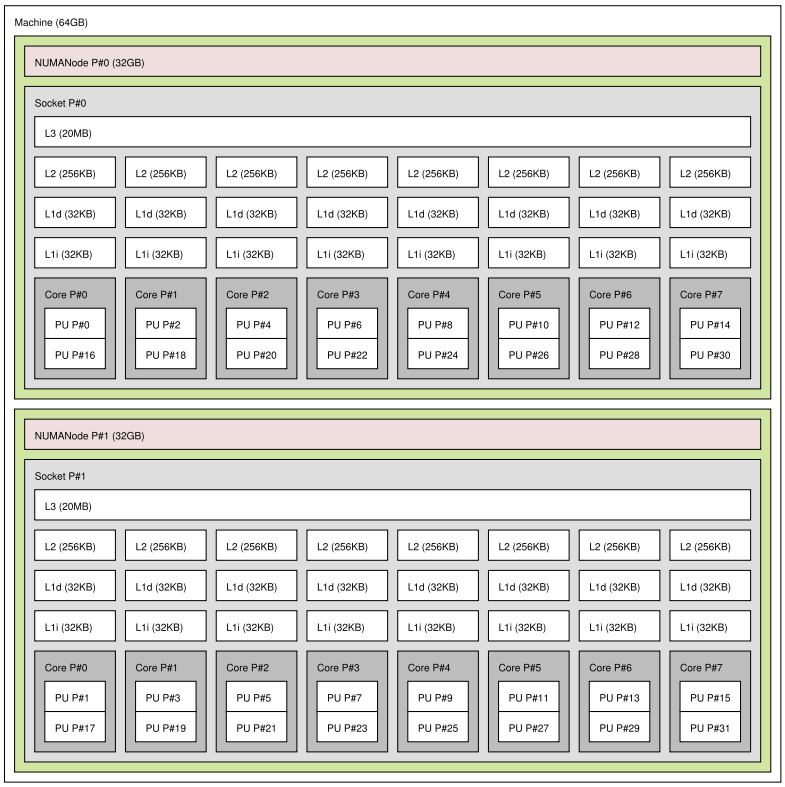


Figure 5.1: Topology of the SMP system used for computing the examples that are presented in this chapter. The system has two sockets, each equipped with one Intel Xeon e5-2650 v2 CPU and 32 GB main memory. A CPU has eight cores with hyperthreading enabled. The different cache levels and their association with the cores are also visible.

5.3 Matrix assembly

The necessity for an efficient and parallel evaluation of the global stiffness matrix is illustrated by fig. 5.2. The chart visualizes the duration of the sequential global stiffness matrix evaluation for the coupled cube example in sect. 4.5.3 on a current Intel CPU machine. While the number of elements is kept constant, the degree of the basis functions is raised from linear to quartic. As a result, the number of active DOF increases from $28,611$ to $46,800$.

While the matrix evaluation time is low for linear and moderate for quadratic basis functions, it increases dramatically for higher degree basis functions. There are several influence factors leading to this result. First of all, the recursive evaluation of the basis functions (cf. eq. (3.76)) becomes more costly for higher degree functions. Secondly, the number of integration points per element grows with the degree of the basis functions when full Gauss quadrature is applied.[3] And finally, also the number of support points per element increases.

A parallel implementation of the element stiffness matrix evaluation is straightforward, as the computation of one element's stiffness matrix is totally independent from the computation of any other element's stiffness matrix and therefore, it can be done concurrently for all elements.

[3] There exist techniques that improve the efficiency of the numerical integration. In the context of isogeometric analysis, patch wide integration schemes briefly mentioned in sect. 3.5.3 may be applied. Furthermore it is possible to underintegrate the element matrices when appropriate stabilization techniques are used. However, both topics are beyond the scope of this work.

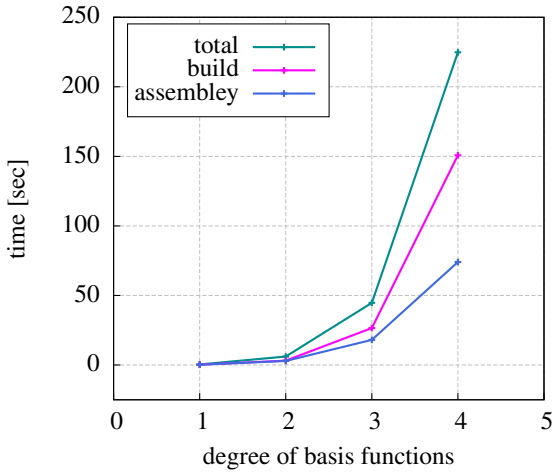


Figure 5.2: Duration of the global stiffness matrix evaluation for the coupled cube problem depicted in fig. 4.22. The chart distinguishes the total evaluation time of element stiffness matrices and the time required for their assembly in the sparse global matrix.

This is not the case for the assembly of the global matrix. As individual element stiffness matrices overlap in the global matrix, concurrent writes to the same matrix entry could occur during the assembly. Since that would lead to an undefined result, concurrent writes must be precluded. Furthermore, the global stiffness matrix has to be stored in a sparse matrix format to meet the limited memory resources of any computer. As sparse matrix formats align the nonzero data in linear vectors, it is not possible to efficiently add matrix entries in an arbitrary order.

The sparsity of the coefficient matrix $\boldsymbol{K}$ of the saddle point problem follows directly from the nature of its block diagonal structure, additional sparsity is a results of the basis function's local support on each subdomain, causing the subdomain stiffness matrices to be sparse themselves. While the element stiffness matrices are evaluated, they are successively added to the global sparse matrix stored in a modified coordinate list (COO) format. When all element and mortar matrices are evaluated, the COO matrix is transformed into compressed sparse column (CSC) format.

A matrix in regular COO format is a simple unsorted list of triplets where each triplet specifies the row index, the column index and the value of a matrix entry. That format has been modified to store duplets that only specify the row index and the value of the matrix entry. These duplets are stored in a multi-dimensional data structure, which has fixed sizes in two dimensions. The first dimension represents the number of computing threads and the second the number of matrix columns. In the third dimension, the structure provides again a list of variable size that contains the unsorted duplets. Using that type of data structure, any thread can add an arbitrary matrix entry at any time without the risk of a concurrent write access to an element.

As random access times to elements in a COO matrix are high, sparse matrices of that type should be converted to CSC format prior to any algebraic computations, as these are better suited and thus more efficient for that purpose. Accordingly, most solver libraries provide interfaces to CSC matrices. A matrix in that format consists basically of three linear vectors, where the first stores the value of a given nonzero entry and the second vector the associated row index. The linear vector index at which the data of a given column in the first two vectors

begins is then stored in the third vector. Converting the matrix from one format to the other, which makes up for the assembly time in fig. 5.2 can also be carried out completely in parallel, with the exception of allocating memory for the matrix and specifying the column indices in the third vector. The algorithm for the parallelized matrix assembly is outlined in algs. 5.1 and 5.2.

Algorithm 5.1 Compute global stiffness matrix in COO format in parallel

Inputs:
numThreads ← number of parallel working threads
numCols ← number of columns in the global stiffness matrix
Initialize:
elemList ← elements of all subdomains (NurbsPatches)
gloStiffCOO ← list[*numThreads*][*numCols*]

for *elem* ← *first* . . *last* **in** *elemList* **do in parallel**
Matrix *elemStiff* ← *elem*.computeStiffnessMatrix()
for *col* ← 1. . *elemStiff*.cols() **do**
for *row* ← 1. . *elemStiff*.rows() **do**
gloStiffCOO[*thread*][*col*].add(duplet(*row*, *elemStiff*(*row*, *col*)))
end for
end for
end for in parallel

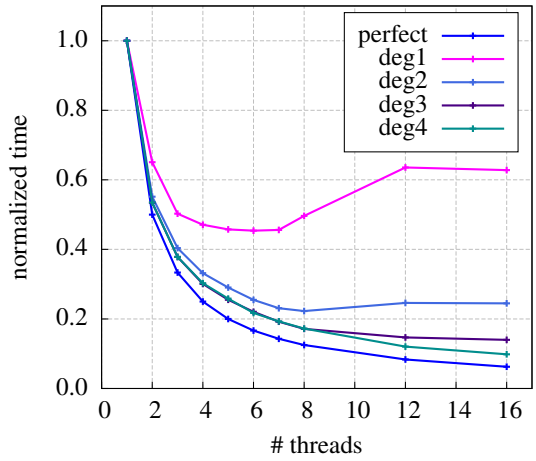


Figure 5.3: Time required for the parallel evaluation of the global stiffness matrix for the example of fig. 4.22 with varying degrees of the basis functions but a constant number of elements. All times are normalized with the duration of the respective sequential matrix evaluation. Absolute time durations for the sequential case of the respective curves with their associated degree of the basis functions can be found in fig. 5.2. The blue curve denotes perfect parallel scaling.

The parallel performance of algs. 5.1 and 5.2 is illustrated by fig. 5.3. When evaluating this plot, it must be kept in mind, that the absolute sequential time required for the evaluation of the global stiffness matrix grows with an increasing degree of the basis functions (cf. fig. 5.2). This explains, why the performance cannot be improved beyond a limit that is specific for a given workload, i.e. basis function degree. A further increase in the number of parallel threads after reaching that limit causes the parallelization overhead to become larger than the gain from

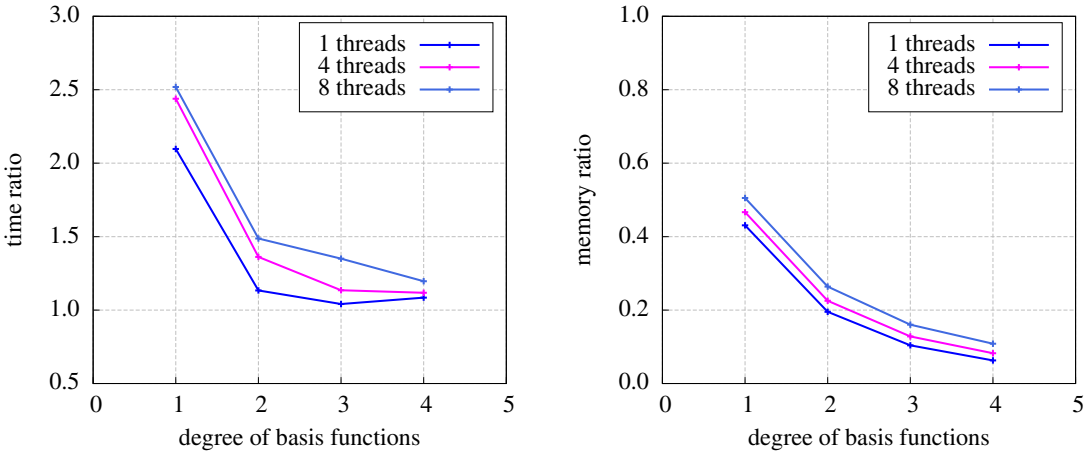
Algorithm 5.2 Convert global stiffness matrix from COO to CSC format in parallel

```
Inputs:
    gloStiffCOO ← global stiffness matrix in COO format
    numThreads ← number of parallel working threads
    numCols ← number of columns in the global stiffness matrix
Initialize:
    nnz ← array[numCols]

for col ← 1.. numCols do in parallel
    for thread ← 2.. numThreads do
        gloStiffCOO[1][col].add( move( gloStiffCOO[thread][col] ))
    end for
    sort( gloStiffCOO[1][col].first() .. gloStiffCOO[1][col].last() by duplet.row() )
    concentrate( gloStiffCOO[1][col] )  ▷ sum duplet values with equal row index at first occurrence
    unique( gloStiffCOO[1][col] )  ▷ delete all duplets with equal row index but the first occurrence
    nnz[col] ← gloStiffCOO[1][col].size()
end for in parallel

allocate( gloStiffCSC( nnz ))
gloStiffCSC.colInd[1]← 1
for  col ← 1.. numCols do
    gloStiffCSC.colInd[col + 1]← nnz[col] + gloStiffCSC.colInd[col]
end for

for col ← 1.. numCols do in parallel
    for rowInd ← 1.. nnz[col]  do
        ind← rowInd + gloStiffCSC.colInd[col] −1
        gloStiffCSC.row[ind] ← gloStiffCOO[1][col].at(rowInd).row()
        gloStiffCSC.value[ind] ← gloStiffCOO[1][col].at(rowInd).value()
    end for
end for in parallel
```

the parallel execution. When however, the workload is large enough, the parallel scaling of the outlined algorithms is very close to perfect scaling. This is a result of the algorithm enforcing the individual threads to work exclusively on their own data and thus eliminating the need for communication or interdependent cache updates. Especially for the high workloads that result from the global stiffness matrix evaluation for NURBS with high degree basis functions, which should generally be used because of their better approximation properties, the algorithms reduces the duration of the matrix evaluation in a relevant order.

However, there is a drawback to alg. 5.1 that is particular to IGA. The data structure used to store the stiffness matrix in COO format causes a significant growth of required memory when the degree of the basis functions increases. This is caused by regular control points being the support points of multiple elements. An interior control point in a quartic solid patch, for example, provides support in 125 elements. Hence, there are 125 duplets to be stored in a single column of the modified COO matrix for just one DOF associated with that support point.

Whenever the main memory of the computer system is an issue, an alternative implementation must be used. In that case, the third dimension of the COO matrix data structure is modified



(a) Time **(b)** Memory

Figure 5.4: Comparison of time and memory requirements of the matrix assembly when different data structures are used for the COO matrix. Results for the map container variant are normalized with corresponding results of the original variant with the unsorted list container.

from a list to an associative map container. The original list stores all duplets in the successive order of their appendance with many duplets referring to the same matrix row index. For the map however, the duplets are ordered by their row index. Whenever a new duplet with an already existing row index is to be inserted, the matrix element's value of that duplet is summed with the value of the existing entry. Thus, each row index will only exist once in a given column's map. The matrix element values originating in different finite elements are successively added to a single duplet. This reduces the memory requirement significantly, but comes at the price of searching the map for the row index each time a new duplet is to be added. The search increases the duration of the COO matrix building but at the same time, the map container causes a reduction of the time required for the conversion to CSC format – as the column entries are already sorted in correct order. A comparison of time and memory requirements of the original algorithms (5.1 and 5.2) to the modified variant is given in fig. 5.4. As can be seen quite clearly, the higher the degree of the basis function, the more reasonable is the use of the second variant with the COO matrix columns being stored in associative map containers. The increased time requirement reduces with higher degree basis functions and simultaneously, the effect of the reduced memory requirement amplifies.

5.4 Solution strategies

5.4.1 Preliminary note

Algorithms for the solution of linear systems are generally categorized as direct or iterative methods. Precluding round-off errors, direct methods result in the exact solution after a predictable finite number of algebraic operations [113]. Unfortunately, this type of solver is also associated with large memory footprints and long computing times, rendering them often unsuitable for large systems. Iterative methods, in contrast, are designed to yield an approximate

solution. Starting from an initial guess, the solution is incrementally improved up to the required accuracy. They usually need considerably less memory and, depending on the convergence rate, they also require less time to lead to a solution of sufficient accuracy. In practical work, however, the same linear system often has to be solved for different load vectors. In that case, most of the computational work of the direct approach can be reused for any load vector following the first, whereas the comparable effect for an iterative solver is limited. The iterative process must be completely repeated and only the constructed preconditioner may be reused. For that reason, both solver categories will be discussed.

The superior performance of IGA over standard C^0 finite elements was noted at several occasions in previous chapters. The performance advantage, i.e. comparable result quality obtained from fewer DOFs, is mainly attributed to the increased continuity of the NURBS basis functions. The downside to this is an also increased average bandwidth of the coefficient matrix leading to a higher cost for solving the system of equations. The number of nonzero entries per row for an interior DOF is $3(2p+1)^3$ in IGA with a displacement based solid element formulation. With classic C^0 elements the matter is more complicated: There, the number of nonzeros is related to the position of the nodal point in an element, it varies from $3(p+1)^3$ for an interior DOF to $3(2p+1)^3$ for a vertex DOF. An extensive study on this issue was done by Collier et al. [40]. They find the number of nonzero entries per row in the case of IGA to be eight times that number of C^0 elements in the limit of large polynomial degrees p. However, for C^0 elements, it is possible to perform a static condensation of the element interior DOFs during matrix assembly. Considering that, the ratio of nonzero row entries between IGA and C^0 elements is unbounded with growing p. For practically relevant degrees of $p = 2$ and $p = 3$, the ratio is reported to be 2.0 and 2.7, respectively when static condensation is not used, and 2.2 and 3.8 when it is used. The increased fill of the global stiffness matrix obviously affects the numerical expense of the solution algorithm. Though the impact on iterative and direct solvers is different, the general statement of an increased memory footprint and longer computing times for linear systems arising from IGA holds for both solution strategies when compared with linear systems from the standard FEM.

This work focusses on the special case of IGA where the NURBS patches are coupled with the mortar method which leads to the saddle point problem in eq. (5.2). As a result of the coupling terms in $\mathbf{B}^T$, the number of nonzero row entries is further increased. Yet, this only affects the rows associated with the DOFs on the coupling interface. And due to their location on the patch boundary, these DOFs originally have a reduced number of nonzero entries which is $3(p+1)(2p+1)^2$. However, the number of additional row entries from the coupling terms can be large for DOFs on the slave side of the coupling interface. For the master side, the number depends on the type of Lagrange multiplier interpolation. At maximum, it is $(2p+1)^2$ for NURBS and $(p+2)^2$ for bilinear Lagrangian interpolation and a solid element formulation. Hence, the total number of nonzero entries will always be smaller for the master side DOFs than for a general patch interior DOF. For a better visualization, the numbers are evaluated in table 5.1 for different degrees of the basis functions. For the slave side, however, the number of additional entries primarily depends on the discretization of the slave mortar side in relation to that of the master side. The coarser the element mesh on the slave interface and the higher the degree of the basis functions, the more Lagrange multipliers are associated with each DOF of a single slave surface element. In the limit, all Lagrange multipliers of a given interface may be associated with the slave DOFs of that interface and the number of their nonzero row entries grows accordingly. On the other hand, there is also a lower bound to the additional row entries

for affected slave DOFs when the discretization of the slave side is very fine. That bound equals $(p+1)^2$ for the NURBS interpolation of the Lagrange multiplier field and 4 in the bilinear case. Thus, it is advisable the keep the discretization of the slave patch face comparable or finer than the master side discretization in order to minimize the number of row entries for the associated slave DOFs. Presuming this is considered, the average number of nonzero row entries does not relevantly increase because of the mortar coupling.

	master side patch boundary DOF			patch interior DOF
	patch DOFs +	LMs =	total	
degree	$3(p+1)(2p+1)^2$	$(2p+1)^2$		$3(2p+1)^3$
1	54	9	63	81
2	225	25	250	375
3	588	49	637	1029
4	1215	81	1296	2187
5	2178	121	2299	3993

Table 5.1: Maximum number of nonzero row entries of the coefficient matrix $\mathbf{K}$ for DOFs on the master side of a mortar coupling interface and in comparison for interior DOFs of a regular NURBS patch.

5.4.2 Iterative methods and preconditioning

5.4.2.1 Iterative solvers

The basic idea of an iterative solver is to improve the solution $\boldsymbol{u}$ by the repeated evaluation of the general recursion

$$\boldsymbol{r}_0 = \boldsymbol{f} - \boldsymbol{K}\boldsymbol{u}_0 \qquad \boldsymbol{z}_0 = \boldsymbol{M}^{-1}\boldsymbol{r}_0 \qquad \boldsymbol{u}_{i+1} = \boldsymbol{u}_i + f\left(\boldsymbol{z}_i\right) \tag{5.3}$$

where $\boldsymbol{r}$ is the residual vector, $\boldsymbol{z}$ its preconditioned counterpart, and $\boldsymbol{M}$ a yet to define precondition operator, which in its most simple form could equal the identity matrix. Iterative methods of Krylov subspace type are considered the most important iterative solution technique for large and sparse linear systems [141]. They follow the general recursion of eq. (5.3), where the definition of the function $f(\boldsymbol{z}_i)$ and the update instruction for $\boldsymbol{z}_i$ determine the specific solver type. In each iteration, Krylov methods usually require the computation of one matrix-vector product and a number of vector scalings and dot products for a system that corresponds in size to the number of unknowns. The application of the precondition operator often requires an additional matrix-vector product per iteration. In consequence, the overall computational cost depends strongly on the sparsity of the coefficient matrix, the precondition operator and on the number of iterations that are required for the method to converge to the solution.

Prominent examples are the conjugate gradient method (CG), the generalized minimal residual method (GMRES), and the bi-conjugate gradient stabilized method (BiCGstab). Well known and primarily used in the context of structural analysis is the CG method because of its efficiency for solving the linear system $\boldsymbol{K}\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{f}}$ where $\boldsymbol{K}$ is a symmetric positive definite matrix

(cf. eq. (5.1)). Unfortunately, the CG method is not robust when the coefficient matrix is indefinite. Thus, the particular properties of the coefficient matrix representing a linear system have to be considered to determine the appropriate Krylov subspace method.

For the saddle point problem $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$ defined in eq. (5.2), the coefficient matrix $\boldsymbol{K}$ is symmetric and indefinite. Iterative methods that focus on that type of coefficient matrices are the symmetric LQ method (SYMMLQ) and the minimum residual method (MINRES) by Paige and Saunders [125]. A third method for indefinite systems is the symmetric quasi-minimal residual method (SQMR) which was developed by Freund and Nachtigal [66]. It is beyond the scope of this work to discuss the mathematical details or the theoretical differences of the three methods. For that purpose, the interested reader is referred to the original publications or, for a general overview on iterative methods, to the monographs by Saad [141] and van der Vorst [159]. The named methods do not differ significantly in regard to their computational cost per iteration and their memory requirements. Thus, their assessment can be reduced to their performance when solving the specific linear systems arising from the discussed IGA method using NURBS basis functions and mortar couplings. The parallel implementation of these methods on SMP systems is fairly trivial and does not require separate discussion. The convergence results are reported afterwards.

5.4.2.2 Preconditioners

Iterative methods of Krylov subspace type commonly show poor convergence rates for linear systems in saddle point form [28]. The slow convergence is generally attributed to a bad conditioning of the coefficient matrix $\boldsymbol{K}$ [113] which, for the linear system $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$, denotes a high sensitivity of the solution $\boldsymbol{u} = \boldsymbol{K}^{-1}\boldsymbol{f}$ towards perturbations in $\boldsymbol{K}$ or $\boldsymbol{f}$. That sensitivity is measured by the condition number κ defined as $\kappa(\boldsymbol{K}) = \|\boldsymbol{K}\|\|\boldsymbol{K}^{-1}\|$. The purpose of a preconditioner is to improve the condition number of the coefficient matrix. Its use is generally essential when iterative methods are employed, but in particular when they are applied to saddle point problems. The challenge is to determine the linear preconditioning operator $\boldsymbol{M}$ such that

$$\kappa\left(\boldsymbol{M}^{-1}\boldsymbol{K}\right) \ll \kappa(\boldsymbol{K}) \tag{5.4}$$

Then, one can solve the left preconditioned system

$$\boldsymbol{M}^{-1}\boldsymbol{K}\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{f} \tag{5.5}$$

instead of the original, ill-conditioned system for the same solution $\boldsymbol{u}$.

An ideal preconditioner would obviously be the coefficient matrix itself. With $\boldsymbol{M} = \boldsymbol{K}$, the condition number would become $\kappa(\boldsymbol{K}^{-1}\boldsymbol{K}) = \kappa(\boldsymbol{I}) = 1$, which is a perfect value. In consequence, the linear system would reduce to

$$\boldsymbol{M}^{-1}\boldsymbol{K}\boldsymbol{u} = \boldsymbol{I}\boldsymbol{u} = \boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{f} \tag{5.6}$$

which can be solved within one iteration step. Yet, that choice of $\boldsymbol{M}$ is prohibitive for obvious reasons. The numerical cost of evaluating the preconditioner would be just as large as computing the solution of the original system with a direct method. Therefore, an operator $\boldsymbol{M}$ or

rather its inverse $\boldsymbol{M}^{-1}$ must be found, which is easy to construct but nonetheless constitutes a good approximation to $\boldsymbol{K}^{-1}$, i.e. $\boldsymbol{M}^{-1} \sim \boldsymbol{K}^{-1}$.

Above, the preconditioning operator was applied from the left side. Just as well it can be applied from the right leading to

$$\boldsymbol{K}\boldsymbol{M}^{-1}\boldsymbol{M}\boldsymbol{u} = \boldsymbol{f} \qquad \Leftrightarrow \qquad \boldsymbol{K}\boldsymbol{M}^{-1}\bar{\boldsymbol{u}} = \boldsymbol{f} \quad \text{and} \quad \boldsymbol{M}\boldsymbol{u} = \bar{\boldsymbol{u}} \tag{5.7}$$

A generalization thereof is two-sided preconditioning, which is achieved by splitting the preconditioning operator, i.e. $\boldsymbol{M} = \boldsymbol{M}_1\boldsymbol{M}_2$, resulting in the preconditioned system

$$\boldsymbol{M}_1^{-1}\boldsymbol{K}\boldsymbol{M}_2^{-1}\bar{\boldsymbol{u}} = \boldsymbol{M}_1^{-1}\boldsymbol{f} \tag{5.8}$$

which falls back to left or right preconditioning when either $\boldsymbol{M}_1$ or $\boldsymbol{M}_2$ are set to the identity matrix $\boldsymbol{I}$.

Matrix scaling

The system of equations (5.2) is initially ill-conditioned for a known reason. The entries in the matrices $\mathbf{A}$ and $\mathbf{B}$ originate from different physical backgrounds and therefore also have different physical units, as do their associated unknown DOFs. This can be overcome by scaling the system, a technique that can also be regarded as preconditioning. A simple and cheap approach known as Jacobi preconditioning is to use the diagonal of the coefficient matrix as the precondition operator. For the saddle point problem, there are zeros on the main diagonal of the coefficient matrix, which renders the standard Jacobi precondition operator $\boldsymbol{M} = \mathrm{diag}(\boldsymbol{K})$ singular and not invertible. However, with some modifications a two-sided scaling operator can be constructed as

$$\boldsymbol{M}_1 = \boldsymbol{M}_2 = \omega^{-\frac{1}{2}} \begin{bmatrix} \mathbf{M}_\mathsf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_\mathsf{B} \end{bmatrix} \tag{5.9}$$

with

$$\mathsf{M}_{\mathsf{A},ii} = sqrt\left(\mathsf{A}_{ii}\right)\big) \quad \text{and} \quad \mathsf{M}_{\mathsf{B},ii} = \max_j \left(\frac{\left|\mathsf{B}_{ij}\right|}{\mathsf{M}_{\mathsf{A},jj}} \right) \quad \text{and} \quad M_{ij} = 0 \;\forall i \neq j \tag{5.10}$$

Taking the square root of the diagonal entries in expression (5.10) is owed to the application of the scaling operator on both sides of the coefficient matrix $\boldsymbol{K}$ in order to preserve its symmetry. The scaled system of equations then becomes

$$\omega \begin{bmatrix} \mathbf{M}_\mathsf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_\mathsf{B} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{M}_\mathsf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_\mathsf{B} \end{bmatrix}^{-1} \begin{bmatrix} \overline{\mathbf{u}} \\ \overline{\boldsymbol{\lambda}} \end{bmatrix} = \omega^{\frac{1}{2}} \begin{bmatrix} \mathbf{M}_\mathsf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_\mathsf{B} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix} \tag{5.11}$$

in which all nonzero entries on the main diagonal of the scaled coefficient matrix are of equal value which also corresponds to the maximum absolute value of each row in the scaled $\mathbf{B}$ matrix. The sub-vectors $\overline{\mathbf{u}}$ and $\overline{\boldsymbol{\lambda}}$ in (5.11) denote the scaled unknown DOFs from which the true unknowns must be recovered after solving the linear system. This is done by computing

$$\begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \omega^{\frac{1}{2}} \begin{bmatrix} \mathbf{M}_\mathsf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_\mathsf{B} \end{bmatrix}^{-1} \begin{bmatrix} \overline{\mathbf{u}} \\ \overline{\boldsymbol{\lambda}} \end{bmatrix} \tag{5.12}$$

Since $\boldsymbol{M}$ is a diagonal matrix, the construction of its inverse is trivial, i.e.

$$M_{ij}^{-1} = \frac{\delta_{ij}}{M_{ij}} \qquad \text{with} \qquad \frac{0}{0} := 0 \tag{5.13}$$

Furthermore, all rows, respectively columns in $\mathbf{M}_{\mathsf{A}}^{-1}$ and $\mathbf{M}_{\mathsf{B}}^{-1}$ are independent of each other and can thus be constructed in parallel from eqs. (5.10) and (5.13). Also the scaling (5.11) itself can be done independently for the individual rows and thus in parallel.

The scalar parameter ω is a global scaling factor for the Jacobi preconditioner. Following the suggestion of Schrader [148], it is determined from the expression

$$\omega = \frac{\kappa\left(\boldsymbol{K}^{-1}\boldsymbol{K}\right)}{\kappa\left(\boldsymbol{M}_1^{-1}\boldsymbol{K}\boldsymbol{M}_2^{-1}\right)} \tag{5.14}$$

which relates the condition number of a system with perfect conditioning to the condition number of the system on which the actually evaluated preconditioning operator was applied. In [148] it is shown, that this relation can be computed from the maximum eigenvalues e_{max} of the involved matrices. Including modifications to consider left and right preconditioning, this leads to the expression

$$\omega = \frac{\left\|\boldsymbol{M}_1\right\| \left\|\boldsymbol{M}_2\right\|}{\|\boldsymbol{K}\|} = \frac{e_{max}\left(\boldsymbol{M}_1\right)\, e_{max}\left(\boldsymbol{M}_2\right)}{e_{max}(\boldsymbol{K})} \tag{5.15}$$

Thus, the eigenvalue problems

$$\det\left(\boldsymbol{K} - e_i\boldsymbol{I}\right) = 0 \qquad \text{and} \qquad \det\left(\boldsymbol{M}_\alpha - e_i\boldsymbol{I}\right) = 0 \qquad \text{for} \quad \begin{array}{l} i \in \{1,\ldots,n+m\} \\ \alpha \in \{1,2\} \end{array} \tag{5.16}$$

need to be solved. As, however, only the maximum eigenvalues are of interest, they can be efficiently approximated with a few iterations of the power method (cf. Golub and Van Loan [71, sec. 8.2]). The numerical cost of the power method is one matrix-vector multiplication and two dot products per iteration, thus it is slightly cheaper than one iteration with any of the Krylov subspace solvers, which require more dot products plus the cost for the preconditioner application. Therefore, computing the factor should reduce the number of required Krylov iterations at least by half the number of iterations used for the computation of the maximum eigenvalue.

In the practical implementation, the matrix scaling is done in two steps. First the scaled coefficient matrix is computed according to eq. (5.11) considering $\omega = 1$. As a result, all diagonal entries of the scaled matrix $\mathbf{A}$ are equal to one, as are the maximum absolute values in the scaled version of the $\mathbf{B}$ matrix. Only then, the maximum eigenvalue according to eq. (5.16) is computed – for the scaled coefficient matrix $\boldsymbol{K}$. The determination of the maximum eigenvalue of $\boldsymbol{M}_\alpha$ becomes obsolete after scaling, as reevaluating $\boldsymbol{M}_\alpha$ after scaling would lead to the identity matrix $\boldsymbol{I}$ for which the eigenvalues are known to be one. Thus, the power iteration only has to be done once in order to determine ω, which is then used to compute the final version of the scaled system of equations.

This type of matrix scaling is very cheap to evaluate and necessary for already named reasons, thus it is applied unconditionally prior to any other preconditioning and solution approaches.

In subsequent sections, the previously introduced notation for the block matrices in the saddle point problem (5.2) is continued to be used, but unless explicitly noted otherwise, it is then denoting the scaled versions of the respective matrices.

Schur preconditioning

Forestalling the convergence results of the iterative solution, it is to note that the previously outlined Jacobi type preconditioning does not suffice to solve the saddle point problem within a low number of iterations. Thus, more sophisticated preconditioning methods are required and Schur complement methods are a self-suggesting option when dealing with 2×2-block-structured matrices.

Recalling the linear system of equations in saddle point form to be

$$\begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{B} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix}$$

and presuming $\mathbf{A}$ to be invertible, block-Gaussian elimination, i.e. adding the $-\mathbf{B}\mathbf{A}^{-1}$ multiple of the first row block equation to the second, can be used to transform the system into

$$\begin{bmatrix} \mathbf{A} & \mathbf{B}^T \\ \mathbf{0} & -\mathbf{B}\mathbf{A}^{-1}\mathbf{B}^T \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ -\mathbf{B}\mathbf{A}^{-1}\mathbf{f} \end{bmatrix} \tag{5.17}$$

which can be solved independently for $\boldsymbol{\lambda}$ only and then, in a second step for $\mathbf{u}$. The block term at position $2,2$ of the coefficient matrix in eq. (5.17) is known as the Schur complement $\mathbf{S}$, i.e.

$$\mathbf{S} = -\mathbf{B}\mathbf{A}^{-1}\mathbf{B}^T \tag{5.18}$$

In [90] it was shown that the preconditioned linear system

$$\boldsymbol{M}^{-1}\boldsymbol{K}\boldsymbol{u} = \boldsymbol{M}^{-1}\boldsymbol{f} \qquad \text{with} \qquad \boldsymbol{M} = \begin{bmatrix} \mathbf{M}_\mathrm{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_\mathrm{B} \end{bmatrix} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \alpha\mathbf{S} \end{bmatrix} \tag{5.19}$$

can be solved within three iterations when $\alpha = -1$ and two iterations for $\alpha = 4$. However, for the problem considered here, $\mathbf{S}$ and the inverse of $\boldsymbol{M}$ cannot be computed as the matrix $\mathbf{A}$ is singular – it contains the unconstrained patch stiffness matrix blocks. But even if it was possible, it is prohibitive because of the associated numerical cost. Therefore, easy to compute approximations for $\mathbf{A}^{-1}$ and $\mathbf{S}^{-1}$ are required.

A simple choice following the idea of Jacobi preconditioning, is to approximate $\mathbf{A}$ with its main diagonal and use that approximation to compute the Schur complement approximation $\tilde{\mathbf{S}}$, i.e.

$$\tilde{\mathbf{A}} = \operatorname{diag}(\mathbf{A}) \qquad\qquad \tilde{\mathbf{S}} = -\mathbf{B}\tilde{\mathbf{A}}^{-1}\mathbf{B}^T \tag{5.20}$$

where $\tilde{\mathbf{S}}$ is a negative-definite symmetric matrix, provided $\mathbf{B}$ has full rank. The precondition operator, which is subsequently referred to as DIAG preconditioner, then becomes

$$\boldsymbol{M} = \begin{bmatrix} \mathbf{M}_\mathrm{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_\mathrm{B} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{A}} & \mathbf{0} \\ \mathbf{0} & \alpha\tilde{\mathbf{S}} \end{bmatrix} \tag{5.21}$$

where $\tilde{\mathbf{A}}$ is again trivial to invert. And, since the size of the Schur complement matrix is associated with the number of patch surface DOFs on coupling interfaces only, it is small compared to the size of the matrix $\mathbf{A}$ representing the volumetric patch stiffness matrix blocks. Therefore, it has been factored for all computed examples in this work, when the inverse precondition operator was required. In the case that $\mathbf{S}$ should become too large for the direct factorization, it is also possible to use a Krylov solver in an inner iteration to compute the approximate action of $\mathbf{S}^{-1}$ on a given vector $\mathbf{v}$.

An alternative suggestion for the approximation of $\mathbf{S}^{-1}$ is to use

$$\tilde{\mathbf{S}}^{-1} = -\mathbf{B}\mathbf{A}\mathbf{B}^T \tag{5.22}$$

for domain decomposition problems [29, 90] which is appealing, as no matrices have to be inverted. However, regarding the bandwidth of the coefficient matrix, which was shown in table 5.1, the general assumption of approximating $\mathbf{A}$ with diag($\mathbf{A}$) may be too simple. Therefore, more sophisticated approaches are discussed next.

Block diagonal factorization preconditioning

In afore presented methods, the inverse preconditioning operator $\boldsymbol{M}^{-1}$ was – with the partial exception of the Schur block in eq. (5.21) – computed explicitly. Therefore, these methods are named explicit preconditioners. With the inverse operator at hand, its application on the coefficient matrix is easy to parallelize, which is an attractive feature when the application has to be done in each iteration of the Krylov solver. This is generally not possible with implicit preconditioners that do not compute the inverse explicitly but resemble its action by other means. With a triangular factorization of the coefficient matrix for example, a forward substitution followed by a backward substitution has to be computed – which can only be done sequentially for a given right-hand side vector.

Nonetheless, factorizations are an established preconditioning technique. Full factorizations like the LU decomposition of a regular sparse matrix $\mathbf{A}$ lead to two triangular matrices $\mathbf{L}$ and $\mathbf{U}$. It was already pointed out that performing a full factorization coincides with solving the linear system with a direct method – which, for large systems, is prohibitive because of the memory requirement. The high memory demand is a consequence of the triangular matrices $\mathbf{L}$ and $\mathbf{U}$ being far less sparse than their source matrix. The typical fill-ratio for 3D problems is about $30 - 50\times$ that of the original matrix [103]. For this reason, incomplete factorizations and prominently incomplete LU factorizations (ILUs) are used to establish the preconditioning operator

$$\mathbf{M}_\mathsf{A} = \tilde{\mathbf{L}}\tilde{\mathbf{U}} \qquad \text{with} \qquad \mathbf{A} = \mathbf{L}\mathbf{U} = \tilde{\mathbf{L}}\tilde{\mathbf{U}} - \mathbf{R} \tag{5.23}$$

in which $\mathbf{R}$ is a residual matrix that guarantees a defined sparsity pattern of $\tilde{\mathbf{L}}$ and $\tilde{\mathbf{U}}$.

A very efficient variant with regard to memory and computation is the zero-fill incomplete LU factorization (ILU(0)). It preserves the sparsity pattern of the original matrix $\mathbf{A}$ by simply ignoring factors that would be created at positions that are zero in the original matrix. When using sparse matrix storage schemes, the same index vectors can be used for $\mathbf{A}$ and $\mathbf{M}$ reducing the memory requirement of the preconditioner to the space needed to store nnz($\mathbf{A}$) floating point numbers. In alg. 5.3, the ILU(0) factorization is compared with the standard LU decomposition.

Algorithm 5.3 ILU(0) according to Saad [141] compared with standard LU on the right

```
M_A ← A,   n ← A.cols()
for i ← 2..n do
    for k ← 1..i−1 do
        if (i,k) ∈ nz(A) then
            M_ik ← M_ik / M_kk
            for j ← k+1..n do
                if (i,j) ∈ nz(A) then
                    M_ij ← M_ij − M_ik M_kj
                end if
            end for
        end if
    end for
end for
```

```
M_A ← A,   n ← A.cols()
for i ← 2..n do
    for k ← 1..i−1 do

        M_ik ← M_ik / M_kk
        for j ← k+1..n do

            M_ij ← M_ij − M_ik M_kj

        end for

    end for
end for
```

A more sophisticated variant allowing the factorization to be of higher accuracy is the dual threshold incomplete LU factorization with pivoting (ILUTP) by Saad [139, 140]. In ILUTP, the sparsity of the factored matrices is controlled by two parameters. The first of them defines a relative drop tolerance τ. All computed factors in a given row that are smaller than the average magnitude of the original row elements multiplied with the drop tolerance are not further considered. The fill value γ is the second parameter. It limits the factored elements in a row to the fill multiple number of nonzero elements in the original row that are largest in magnitude.

The use of incomplete factorizations for the preconditioning is attractive because the NURBS patch related stiffness matrix blocks in $\mathbf{A}$ are completely uncoupled. This allows to define a true block diagonal preconditioner on the basis of incomplete factorizations[4] in which the blockwise factorizations are independent of each other and therefore can be computed in parallel. As, however, the singularities of the individual stiffness matrix blocks could cause the incomplete factorization process to fail, the singularities have to be "repaired". While doing so, the modifications on the block matrices must be confined to preserve the spectral equivalence of the preconditioner to the coefficient matrix of the linear system. With that in mind, the precondition operator is defined identical to eq. (5.21), but the approximation for the $\mathbf{A}$-block in that operator is given by

$$\tilde{\mathbf{A}} = \begin{bmatrix} \mathrm{ilu}\left(\boldsymbol{K}^{(1)} + \beta^{(1)}\boldsymbol{I}\right) & 0 & 0 \\ 0 & \mathrm{ilu}\left(\boldsymbol{K}^{(2)} + \beta^{(2)}\boldsymbol{I}\right) & 0 \\ & & \ddots \\ 0 & 0 & \mathrm{ilu}\left(\boldsymbol{K}^{(n_{sub})} + \beta^{(n_{sub})}\boldsymbol{I}\right) \end{bmatrix} \tag{5.24}$$

where $\mathrm{ilu}(\cdot)$ denotes any of the incomplete factorization variants. Unless the non-perturbed stiffness matrix blocks $\boldsymbol{K}^{(i)}$ are subjected to Dirichlet boundary conditions, they are known to be positive semi-definite. Therefore, shifting the matrices with a positive parameter $\beta^{(i)}$ guarantees the resulting matrices to be positive-definite.

Semi-definite matrices have a condition number that is equal to infinity. Though the matrices must not be singular in order to ensure a stable factorization, the precondition operators should

[4] Saddle point problems are often associated with block diagonal preconditioners. However, they are generally restricted to the four blocks of the coefficient matrix in the saddle point problem (cf. eq. (5.2)), which severely limits the potential for parallelization.

have similar spectral properties as their corresponding stiffness matrix blocks. Therefore, they should be as close to being singular as possible. Since the condition number in the l_2-norm of positive-definite matrix can be expressed by the relation of the maximum to the minimum eigenvalue, the block individual shifting parameter $\beta^{(i)}$ is determined as

$$\beta^{(i)} = \beta_{tol}\, e_{max}\left(\boldsymbol{K}^{(i)}\right) \tag{5.25}$$

in which the maximum eigenvalue $e^{(i)}_{max}$ of the coefficient matrix $\boldsymbol{K}^{(i)}$ is again determined from a few iterations with the power method. Knowing that the minimum eigenvalue $e^{(i)}_{min}$ of $\boldsymbol{K}^{(i)}$ equals zero, the condition number of the shifted matrix evaluates to

$$\kappa\left(\boldsymbol{K}^{(i)} + \beta^{(i)}\boldsymbol{I}\right) = \frac{e^{(i)}_{max} + \beta^{(i)}}{e^{(i)}_{min} + \beta^{(i)}} = \frac{e^{(i)}_{max} + \beta_{tol} e^{(i)}_{max}}{\beta_{tol} e^{(i)}_{max}} = \frac{1 + \beta_{tol}}{\beta_{tol}} \tag{5.26}$$

Hence, the condition is only controlled by the parameter β_{tol}. In accordance with the floating point precision of the computing machine, β_{tol} can be chosen as the smallest value that does not result in numerical instabilities.

The approximate Schur complement matrix

$$\tilde{\mathbf{S}} = -\mathbf{B}\tilde{\mathbf{A}}^{-1}\mathbf{B}^T \tag{5.27}$$

is then computed from the factored matrix $\tilde{\mathbf{A}}$ in eq. (5.24). This computation can be rather expensive, but considering that $\mathbf{B}$ is actually a sparse block matrix itself, it can also be efficiently parallelized. Denoting the incomplete factorized patch stiffness matrix blocks $\boldsymbol{K}^{(i)}$ by $\tilde{\boldsymbol{L}}^{(i)}\tilde{\boldsymbol{U}}^{(i)}$, the algorithm for the computation of the individual matrix blocks $\tilde{\boldsymbol{S}}$ in the approximate Schur complement (5.27) becomes

$$\tilde{\boldsymbol{S}}_{ij} = -\boldsymbol{m}_{ik}\tilde{\boldsymbol{U}}^{-1}_{kk}\tilde{\boldsymbol{L}}^{-1}_{kk}\boldsymbol{m}_{jk} \tag{5.28}$$

To illustrate the process, it is applied to the coefficient matrix associated with the four subdomain variant of the cantilever beam example in sect. 4.5.1. In block notation, that coefficient matrix is given as

$$\boldsymbol{K} = \begin{bmatrix}
\boldsymbol{K}_1 & & & & \boldsymbol{m}'_{a1} & \boldsymbol{m}'_{b2} & & & \\
& \boldsymbol{K}_2 & & & \boldsymbol{m}'_{a2} & & \boldsymbol{m}'_{c2} & \boldsymbol{m}'_{d2} & \\
& & \boldsymbol{K}_3 & & & \boldsymbol{m}'_{b3} & \boldsymbol{m}'_{c3} & & \boldsymbol{m}'_{e3} \\
& & & \boldsymbol{K}_4 & & & & \boldsymbol{m}'_{d4} & \boldsymbol{m}'_{e5} \\
\boldsymbol{m}_{a1} & \boldsymbol{m}_{a2} & & & & & & & \\
\boldsymbol{m}_{b1} & & \boldsymbol{m}_{b3} & & & & & & \\
& \boldsymbol{m}_{c2} & \boldsymbol{m}_{c3} & & & & & & \\
& \boldsymbol{m}_{d2} & & \boldsymbol{m}_{d4} & & & & & \\
& & \boldsymbol{m}_{e3} & \boldsymbol{m}_{e4} & & & & &
\end{bmatrix}$$

with numeral subscripts indexing the subdomains and lower case literal subscripts the mortar interfaces; zero blocks are not explicitly written. For reasons of space, the elsewhere used superscripts had to be replaced by subscripts and for the same reason, brackets around subdomain indices are omitted and matrix transposes are indicated by apostrophes.

Using eq. (5.27) in conjunction with eq. (5.24) and expressing $\tilde{\boldsymbol{U}}_i^{-1}\tilde{\boldsymbol{L}}_i^{-1}$ by $\tilde{\boldsymbol{K}}_i^{-1}$, the associated negative approximate Schur complement matrix becomes

$$
-\tilde{\mathbf{S}} = \begin{bmatrix}
\boldsymbol{m}_{a1}\tilde{\boldsymbol{K}}_1^{-1}\boldsymbol{m}'_{a1} + \boldsymbol{m}_{a2}\tilde{\boldsymbol{K}}_2^{-1}\boldsymbol{m}'_{a2} & \boldsymbol{m}_{a1}\tilde{\boldsymbol{K}}_1^{-1}\boldsymbol{m}'_{b1} & \boldsymbol{m}_{a2}\tilde{\boldsymbol{K}}_2^{-1}\boldsymbol{m}'_{c2} & \boldsymbol{m}_{a2}\tilde{\boldsymbol{K}}_2^{-1}\boldsymbol{m}'_{d2} & \\
\boldsymbol{m}_{b1}\tilde{\boldsymbol{K}}_1^{-1}\boldsymbol{m}'_{a1} & \boldsymbol{m}_{b1}\tilde{\boldsymbol{K}}_1^{-1}\boldsymbol{m}'_{b1} + \boldsymbol{m}_{b3}\tilde{\boldsymbol{K}}_3^{-1}\boldsymbol{m}'_{b3} & \boldsymbol{m}_{b3}\tilde{\boldsymbol{K}}_3^{-1}\boldsymbol{m}'_{c3} & & \boldsymbol{m}_{b3}\tilde{\boldsymbol{K}}_3^{-1}\boldsymbol{m}'_{e3} \\
\boldsymbol{m}_{c2}\tilde{\boldsymbol{K}}_2^{-1}\boldsymbol{m}'_{a2} & \boldsymbol{m}_{c3}\tilde{\boldsymbol{K}}_3^{-1}\boldsymbol{m}'_{b3} & \boldsymbol{m}_{c2}\tilde{\boldsymbol{K}}_2^{-1}\boldsymbol{m}'_{c2} + \boldsymbol{m}_{c3}\tilde{\boldsymbol{K}}_3^{-1}\boldsymbol{m}'_{c3} & \boldsymbol{m}_{c2}\tilde{\boldsymbol{K}}_2^{-1}\boldsymbol{m}'_{d2} & \boldsymbol{m}_{c3}\tilde{\boldsymbol{K}}_3^{-1}\boldsymbol{m}'_{e3} \\
\boldsymbol{m}_{d2}\tilde{\boldsymbol{K}}_2^{-1}\boldsymbol{m}'_{a2} & & \boldsymbol{m}_{d2}\tilde{\boldsymbol{K}}_2^{-1}\boldsymbol{m}'_{c2} & \boldsymbol{m}_{d2}\tilde{\boldsymbol{K}}_2^{-1}\boldsymbol{m}'_{d2} + \boldsymbol{m}_{d4}\tilde{\boldsymbol{K}}_4^{-1}\boldsymbol{m}'_{d4} & \boldsymbol{m}_{d4}\tilde{\boldsymbol{K}}_4^{-1}\boldsymbol{m}'_{e4} \\
& \boldsymbol{m}_{e3}\tilde{\boldsymbol{K}}_3^{-1}\boldsymbol{m}'_{b3} & \boldsymbol{m}_{e3}\tilde{\boldsymbol{K}}_3^{-1}\boldsymbol{m}'_{c3} & \boldsymbol{m}_{e2}\tilde{\boldsymbol{K}}_4^{-1}\boldsymbol{m}'_{d4} & \boldsymbol{m}_{e3}\tilde{\boldsymbol{K}}_3^{-1}\boldsymbol{m}'_{e3} + \boldsymbol{m}_{e4}\tilde{\boldsymbol{K}}_4^{-1}\boldsymbol{m}'_{e4}
\end{bmatrix}
$$

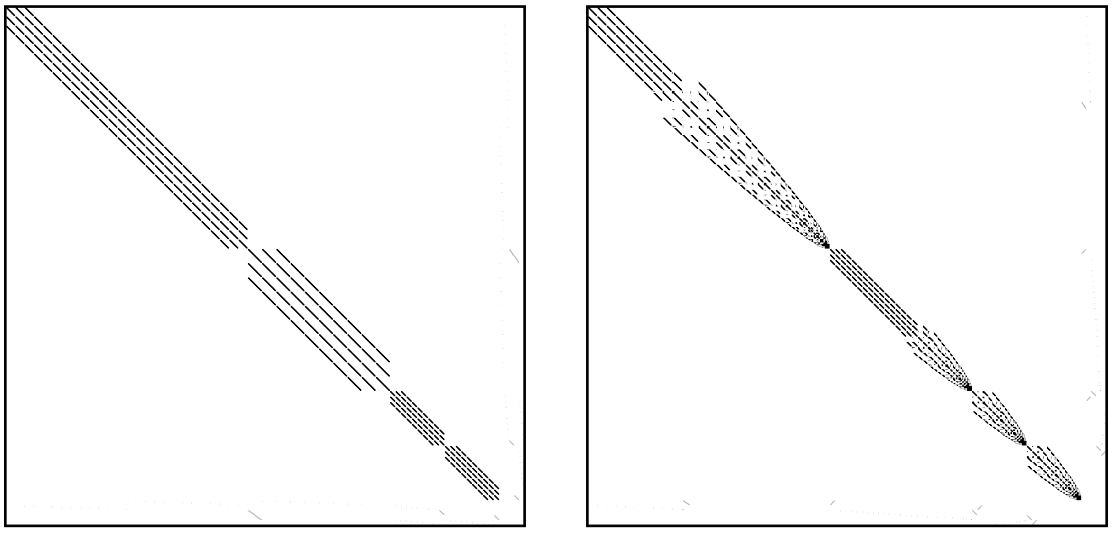
in which each block element can be computed independently. Considering the symmetry of the Schur matrix, the computation has to be done only for the lower or upper triangular block matrix. Further savings of numerical costs are achieved by observing that the results of the expensive forward – backward substitutions which are required to compute specific $\tilde{\boldsymbol{K}}_i^{-1}\boldsymbol{m}'_j$ terms can be reused at different block elements in the matrix. And finally it is to note, that the mortar matrices $\boldsymbol{m}_j$ are very sparse with most columns being zero.

Though the Schur block matrix of this example appears very dense, it is not in general. The geometry of the specific example is very compact and almost all subdomains are coupled with each other. For practically relevant examples this is unlikely to be the case. Then, the matrix will primarily contain zero blocks; only the main diagonal is guaranteed to be dense and coupling terms will exist for each subdomain that is associated with more than one mortar interface.

A final note has to be made with regard to the incomplete factorization. In the implementation referencing this section, variants of the incomplete LU factorization have been used, which in the limit of a complete factorization require the input matrix to be square and invertible. When the input matrix is also symmetric and positive-definite, which is the case for eq. (5.24), the Cholesky decomposition can be used instead. For that specific matrix type, the Cholesky factorization may be preferable because the two triangular matrices that result from the factoring are the transposes of each other leading to a higher memory efficiency. Algorithms for incomplete variants of the Cholesky factorization that are comparable to ILU(0) and ILUTP exist, see Huang et al. [81] for further details.

Ordering

Whenever a sparse matrix is to be factored, no matter whether the factorization is complete or incomplete, it is suggested to reduce its bandwidth beforehand in order to decrease the fill-in during the factorization. That reduction is achieved by reordering the matrix elements. When the symmetry of a matrix is to be preserved, rows and columns have to be permuted alike. Otherwise, either the rows or columns are reordered. Yet, the latter variant of a non-symmetric reordering is reported to not perform well in the case of discretized partial differential equations



(a) Natural ordering **(b)** RCM ordering

Figure 5.5: Sparsity pattern of $\boldsymbol{K}$ for the cantilever beam problem depicted in fig. 4.10(b) after some uniform refinement of the subdomain. The matrix size is 1894 with 78048 nonzero elements.

[141]. For the symmetric case, the reverse Cuthill-McKee reordering (RCM) [68] is a popular method that is frequently used for ILU preconditioning [37, 38]. However, for the examples of this work, a positive effect measured by a reduced number of iterations was not observed when RCM reordering was applied to $\mathbf{A}$ prior to the incomplete factorization of the contained block matrices. The likely reason for this is the superb natural ordering of these matrices as a result of the tensor product structure of the NURBS basis functions and the numbering of the unknown DOFs derived therefrom. This is visualized by the sparsity pattern of the coefficient matrix resulting from the four subdomain example in sect. 4.5.1, which is shown in fig. 5.5. When yet the action of the inverse Schur matrix is evaluated via a factorization of $\tilde{\mathbf{S}}$, it is recommendable to perform a block reordering of this matrix: The block sparsity structure of $\tilde{\mathbf{S}}$ results from the numbering of mortar interfaces, which is always arbitrary.

5.4.2.3 Convergence results

In this section, convergence results obtained with the three different iterative solvers combined with the previously outlined preconditioning techniques are presented. Preconditioners exist in many variants; those shown here were adapted to facilitate the solution process of the given problem. The selection also represents significant differences in the numerical cost related to their construction and application. The goal of this section is to develop an idea of the preconditioner suitability for solving the specific saddle point problem arising in this framework and to evaluate some of the parameters that are associated with these preconditioners.

2D case - Cantilever beam example

The iterative solvers MINRES, SYMMLQ, and SQMR use different residual norms evaluated in the course of the iterative process as their stopping criteria. In order to make the behavior of these solvers comparable, the normalized true residual of the scaled linear system defined in

eq. (5.29) is additionally evaluated in each iteration i and used for the convergence plots in this section. Whenever solution times are reported, the true residual is not evaluated and instead, the iterative process is stopped at afore determined iteration step.

$$r_i^{true} = \frac{\|\boldsymbol{f} - \boldsymbol{K}\boldsymbol{u}_i\|_2}{\|\boldsymbol{f} - \boldsymbol{K}\boldsymbol{u}_0\|_2} \tag{5.29}$$

For a first study on 2D problems, the cantilever beam example previously presented in sect. 4.5.1 and pictured in fig. 4.8(b) is used. The specific problem has four subdomains and quadratic basis functions in both parametric directions, which, after uniform refinement of the initial geometry, leads to 6, 396 unconstrained DOFs and 170 LMs. This results in a linear system with 6, 566 equations. The coefficient matrix has 296, 640 nonzero entries.

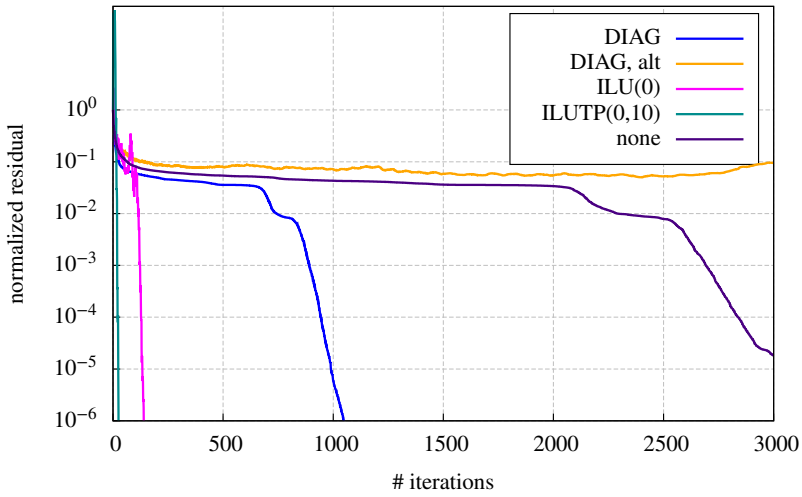
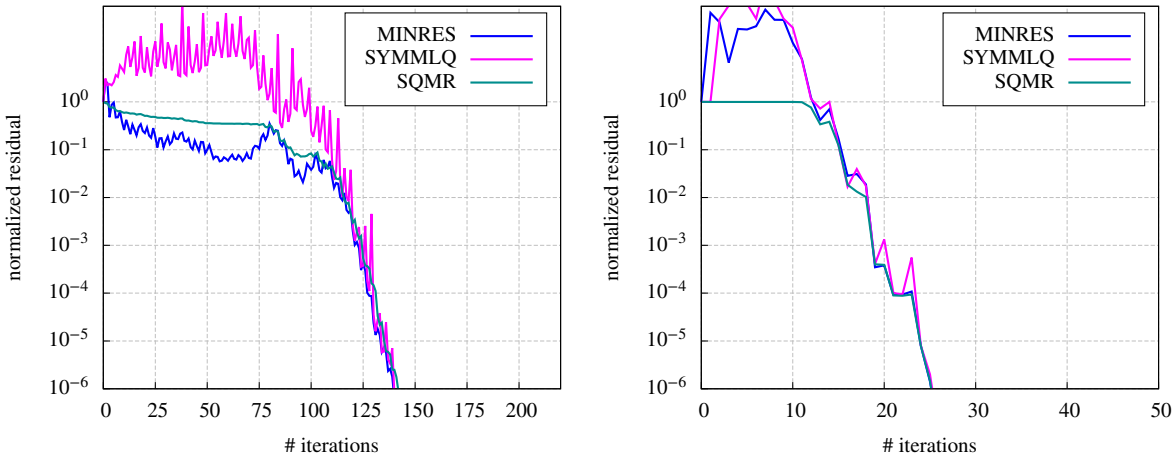


Figure 5.6: Convergence of the iterative solver MINRES and different preconditioners for a coefficient matrix with 296, 640 nonzero entries in 6, 566 rows obtained from the cantilever beam example. The first two preconditioners refer to those in eq. (5.21), with 'DIAG' denoting the definition in eq. (5.20) and 'DIAG, alt' the alternative definition in eq. (5.22). ILU(0) refers to the incomplete block diagonal factorization preconditioner with zero fill-in and ILUTP to the more sophisticated factorization variant with dual threshold and pivoting. For this, the drop tolerance is $\tau = 0$ and the fill in factor $\gamma = 10$. The Schur complement factor $\alpha = -1$ for all variants.

Figure 5.6 visualizes the convergence rates of the MINRES solver when different preconditioners are used. Their fundamental influence on the entire solution process becomes obvious. The more costly block factoring preconditioners ILU(0) and ILUTP perform significantly better than those using a diagonal matrix to approximate $\mathbf{A}$, especially the alternative formulation to approximate the Schur complement $\mathbf{S}$ defined in eq. (5.22) proves worthless, as it causes a worse convergence than the total abundance of a preconditioner.

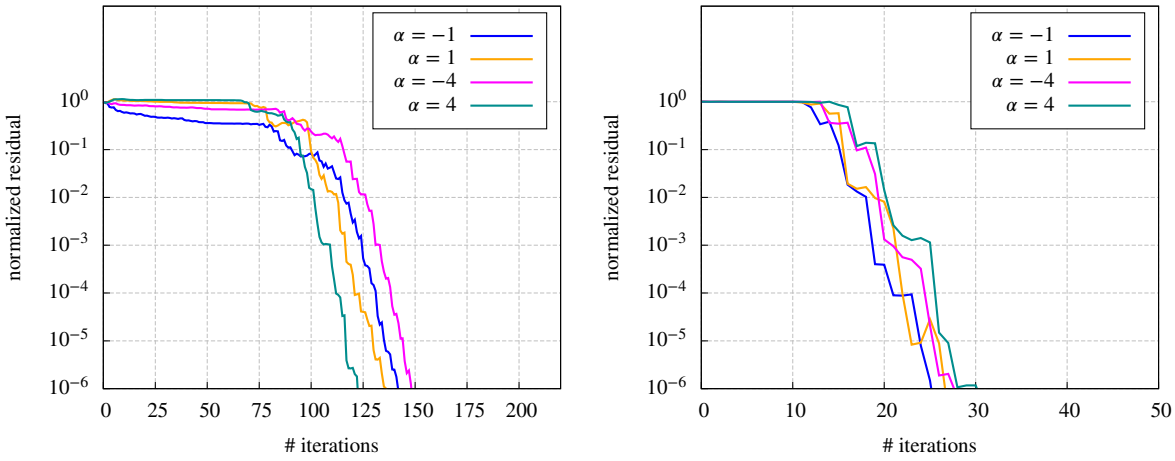
Looking closer into the convergence behavior of the different solvers reveals that a relevant difference in the behavior of these solvers does not exist. The respective curves obtained with the two ILU variants are plotted in fig. 5.7. Though the initial performance of these solvers is quite different, their convergence curves nearly coincide after a number of iterations. At the target residual of 10^{-6}, the maximum difference in the number of required iterations to reach that target is two.



(a) ILU(0) **(b)** ILUTP

Figure 5.7: Convergence of the three different iterative solvers for the ILU variants ILU(0) and ILUTP applied to the linear system of the cantilever beam example. In all evaluations, $\alpha = -1$. The ILUTP parameters are set to $\tau = 0$ and $\gamma = 10$ and the β_{tol} factor to repair the singularities in the $\tilde{\mathbf{A}}^{(i)}$ blocks is $\beta_{tol} = 10^{-5}$ for ILUTP and $\beta_{tol} = 0$ for ILU(0).

The influence of the Schur preconditioning factor α defined in eq. (5.19) is evaluated in fig. 5.8. It is to note, that MINRES and SYMMLQ solvers, though designed for indefinite problems, are limited to positive-definite preconditioners, otherwise these solver fail. This requires α to be negative in order to compensate for the also negative Schur complement definition (5.18). This restriction does not apply to the SQMR solver, which is therefore the only one used in this evaluation. For the ILU(0) preconditioner, there is a distinct influence of α. Clearly, the variant with $\alpha = 4$, leading to an indefinite preconditioner, converges faster than all other α-variants. In the case of ILUTP, the influence is not as obvious. There may be a tendency towards $\alpha = -1$ being the best choice for this example, but actually the convergence curves are too volatile to



(a) SQMR + ILU(0) **(b)** SQMR + ILUTP

Figure 5.8: Convergence of the SQMR solver with ILU(0) and alternatively with ILUTP preconditioners when the Schur preconditioning factor α is varied as ± 1 and ± 4. The linear system is that of the cantilever beam example.

make a profound assessment.

scaling	residual norm
ω acc. to eq. (5.15)	$1.42 \cdot 10^{-11}$
$\omega = 1$	$1.48 \cdot 10^{-11}$
no scaling	$1.60 \cdot 10^{-11}$

Table 5.2: Residual norm $\|\boldsymbol{f} - \boldsymbol{K}\boldsymbol{u}_{150}\|_2$ for different scalings applied to the linear system of the cantilever beam prior to its solution.

In this context, also the influence of the scaling factor ω used for the equilibration of the coefficient matrix according to eq. (5.11) was examined. For that purpose, the linear system of the cantilever beam example was solved with different scalings previously applied to the system. After obtaining the solution, the unscaled system was restored and the residual norm was computed. For all scaling variants, 150 iterations of the SQMR solver with ILU(0) preconditioning (with $\alpha = 4$) were used. Results are reported in table 5.2. The influence of the scaling factor is clearly negligible, as is the scaling in general for this example. However, the scaling is required to prevent numerical issues, but the effort for evaluating ω can be spared.

3D case - Solid cube coupling example

Next, the iterative solution process is evaluated on the 3D example of the coupled solid cubes presented in sect. 4.5.3. NURBS basis functions of degree $p = 2$ as well as of degree $p = 3$ are considered, leading to a system with 12,963 equations (12,636 active DOFs, 327 LMs) and 3,400,062 nonzero entries in the coefficient matrix for the quadratic case and 16,572 equations (16,245 active DOFs, 327 LMs) and 10,722,573 nonzero entries in the coefficient matrix for the cubic case. It is to be observed, that the number of equations grows by a factor of 1.28 while the number of nonzero entries increases by a factor of 3.15. Using a bilinear Lagrangian interpolation for the field of Lagrange multipliers causes the number of LMs to be unaffected by the increased degree of the basis functions.

As in the 2D case, there is no difference in the performance of the three iterative solvers, therefore only the MINRES convergence results are plotted in fig. 5.9. Also, the three preconditioner variants perform as expected in relation to the numerical cost of their construction and application. More of a surprise is the convergence of the variant with quadratic basis function compared to the one with cubic functions. With all preconditioners, the larger cubic problem converges faster than the respective quadratic counterpart. This behavior is credited to the decreased size of the coupling terms, i.e. the $\mathbf{B}$ matrix, compared to the size of the stiffness matrices block $\mathbf{A}$ of the saddle point formulation.

Regarding the influence of the α-factor for the Schur complement part of the preconditioners, exactly the same behavior as in the 2D cantilever beam example is obtained. The associated results for the diagonal and the ILU(0) preconditioner are plotted in fig. 5.10. In both cases, $\alpha = 4$ delivers the best performance. For ILUTP, the respective convergence curves are not plotted, as they are similarly undetermined as those in fig. 5.8(b) and thus, no clear preference for a specific value can be stated though again, $\alpha = -1$ appears to have a slight performance advantage.

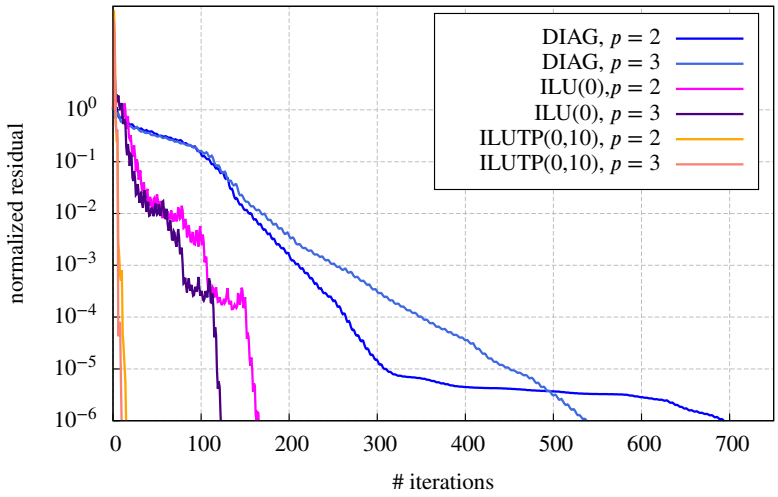
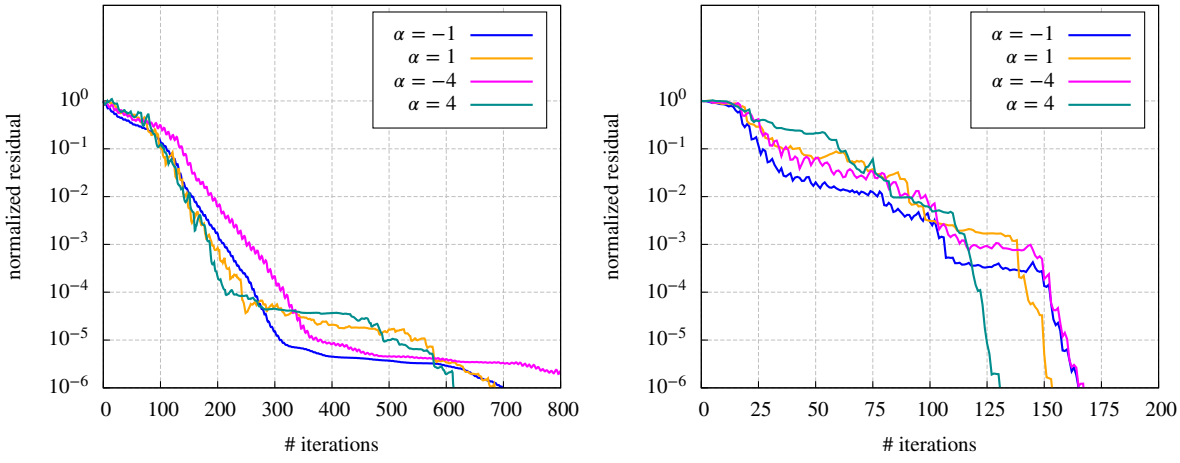


Figure 5.9: Convergence of the iterative solver MINRES and different preconditioners for the cube coupling example with basis functions degrees $p = 2$ and $p = 3$. The preconditioner denoted by 'DIAG' refers to the definition in eq. (5.20). The ILU variants ILU(0) and ILUTP denote block diagonal factorizing preconditioners. For ILUTP, the drop tolerance is $\tau = 0$, the fill in factor $\gamma = 10$, and $\beta_{tol} = 10^{-5}$. In the other cases, $\beta_{tol} = 0$. The Schur complement factor $\alpha = -1$ for all variants.

It must be pointed out, that a faster convergence and thus a lower number of necessary solver iterations is not the unconditional optimum of the iterative solution process. The convergence rate has to be considered with reference the associated cost of the preconditioner construction and its application. This is clearly revealed by fig. 5.11, which visualizes the numerical cost, measured in CPU time, of solving the coupled cube example with different preconditioners. Though the DIAG preconditioner has the worst convergence, it outperforms the ILU variants by a clear margin, as its construction and application is virtually for free. The reported time du-



(a) SQMR with DIAG

(b) SQMR with ILU(0)

Figure 5.10: Convergence of the SQMR solver with the DIAG and alternatively with the ILUTP preconditioner when the Schur preconditioning factor α is varied as ± 1 and ± 4. The linear system is that of the solid cube coupling example with quadratic basis functions.

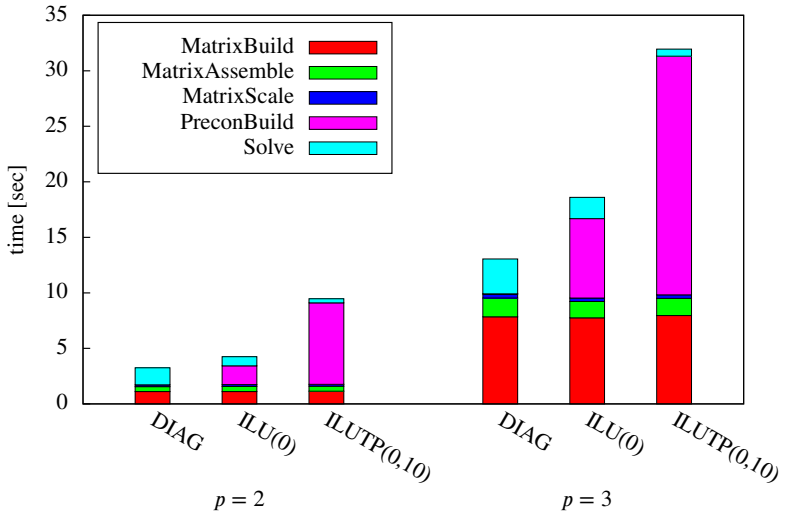


Figure 5.11: Time required for building and solving the linear system of the coupled solid cubes example with quadratic or alternatively cubic basis functions when different preconditioners are used. Executed with two threads in parallel.

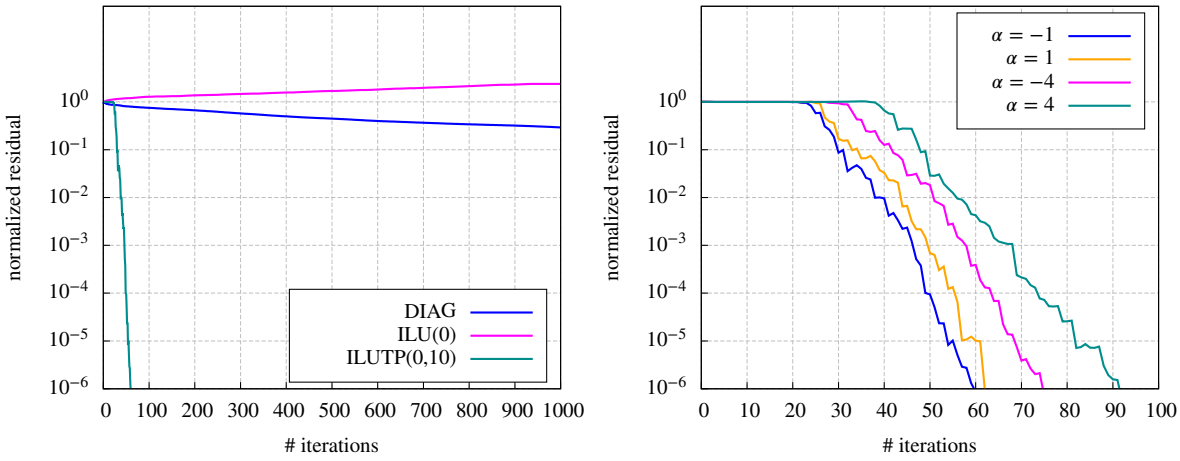
rations were obtained with a two thread parallel execution, which is optimal for the subdomain parallel factorization of the preconditioner for the given problem.

3D case - MultiStory example

After discussing two rather academic examples, convergence results shall also be shown for a problem with a higher practical relevance. The multistory example is more thoroughly presented in sect. 6.3.4. Here, only the matrix properties are briefly stated: The problem consists of 23 subdomains, leading to a system with 40,929 equations (38,352 active DOFs, 2,577 LMs) and 8,166,348 nonzero entries in the coefficient matrix.

The results shown in fig. 5.12 bring back to mind that iterative solutions may not converge. In this example, this is the case for the solution with the ILU(0) preconditioner. Neither the use of other values for the parameter α nor setting the β_{tol} parameter to a nonzero value does change that behavior. Likewise without effect is the application of RCM reordering on the coefficient matrix. Just as futile is the use of the DIAG preconditioner for this problem. Though some convergence is achieved, the rate is too low to be of practical relevance. Solely the most sophisticated preconditioner considered within this work converges within a low number of iterations. ILUTP is used with the same parameters as before. And finally, a clear performance advantage can be observed for that preconditioner for $\alpha = -1$ compared to all other α values.

In all examples, ILUTP preconditioning is realized with the implementation in the SuperLU library [104], which uses a modified dropping rule compared to the original scheme by Saad [140]. The modified scheme, named adaptive area-based dropping, regards the input parameter γ as an upper bound and computes the actual fill for a defined region of the factored matrix dynamically during the factorization. The scheme also adjusts the initial input for τ to not exceed the desired fill-in. Details on the modified dropping rule are provided by Li and Shao [105]. It is to note, that this scheme produced significantly better convergence rates than the standard ILUTP, while at the same time, the final fill ratio of the factored matrices was distinctively smaller than the input parameter γ suggested. For the multistory example, the fill ratio was set



(a) Preconditioner types

(b) ILUTP(0,10), α-factor

Figure 5.12: Solving with multistory example with the SQMR iterative solver. The convergence for different preconditioners is shown in (a), whereas (b) restricted to the ILUTP preconditioner with parameters $\tau = 0$, $\gamma = 10$, and $\beta_{tol} = 10^{-5}$ and different varying values for α.

to $\gamma = 10$, an evaluation of the factored matrices yet showed that the true fill ratio was only 2.8.

5.4.3 Substructuring methods

The previous section revealed that iterative solutions heavily depend on the quality of the preconditioner. Especially in the case of saddle point problems, the convergence can be slow or the solution may entirely fail. Though defaults can be set, the selection of an appropriate preconditioner and associated parameters is within the user's responsibilities. The use as a black-box solver is therefore problematic. And regardless of this limitation, it must be considered that practical situations often require the evaluation of the structural response to many different load cases. When using iterative solvers, the computational work of the iterative process remains nearly constant for each load variant. Merely the preconditioner construction is a one-time effort. After the construction, it can then be used for all load situations. With direct methods, the main computational cost results from the factorization of the coefficient matrix into the product of two triangular matrices. Once this is done, the structural response to a given load case is computed by a forward substitution followed by a back substitution of the load vector with the two triangular matrices. Compared to the factorization itself, the forward and back substitutions are cheap to compute. With standard LU decomposition including partial pivoting, the computational cost for factoring a nonsingular matrix is of order $\mathcal{O}(2/3n^3)$ whereas the cost for the associated triangular solve is only $\mathcal{O}(2n^2)$. Therefore, the efficiency of direct methods grows with number of load cases that are to be analyzed.

The analysis of large volumetric structures, however, quickly leads to systems of equations with a huge number of unknown DOFs that cannot be easily handled by direct solvers. Though advanced direct solver libraries such as MUMPS [6] or SuperLU [104] use the sparsity structure of the coefficient matrix to improve the computational efficiency and to parallelize the matrix

decomposition, at least some of the limitations of direct methods persist when these general purpose solvers are applied to the global problem in a black box fashion.

Substructuring methods attempt to solve a problem by subdividing it into a number of independent smaller problems that are easier to handle. With the knowledge about the physical background of the linear system, the subdivision can be done more effectively than a general purpose solver is able to do. Kleiss et al. [97] recently proposed the IETI method to combine isogeometric analysis with the dual-primal variant of the tearing and interconnecting method (FETI-DP) [61], which is a well-known substructuring method for standard C^0 finite elements. The IETI method utilizes individual NURBS patches as the basis for substructuring the global problem. The method, however, presupposes an initially conforming discretization of adjacent patches. This constitutes a severe limitation that renders it non-applicable for the purposes pursued in this work. With the linear system that results from coupling NURBS patches with the mortar method, a more flexible substructuring method for IGA can be constructed.

The starting point is a reordering of all assembled subdomain stiffness matrix blocks contained in A such that each individual block matrix $\boldsymbol{K}^{(i)}$ is ordered as

$$\boldsymbol{K}^{(i)} = \begin{bmatrix} \boldsymbol{K}_{ff}^{(i)} & \boldsymbol{K}_{fm}^{(i)} \\ \boldsymbol{K}_{mf}^{(i)} & \boldsymbol{K}_{mm}^{(i)} \end{bmatrix} \tag{5.30}$$

where the subscript m denotes those terms affected by mortar constraints and the subscript f all other DOFs, which are hence unconstrained or free. The unknown displacement DOFs and the related force vectors are split accordingly into

$$\tilde{\boldsymbol{u}}^{(i)} = \begin{bmatrix} \tilde{\boldsymbol{u}}_f^{(i)} \\ \tilde{\boldsymbol{u}}_m^{(i)} \end{bmatrix} \qquad \tilde{\boldsymbol{f}}^{(i)} = \begin{bmatrix} \tilde{\boldsymbol{f}}_f^{(i)} \\ \tilde{\boldsymbol{f}}_m^{(i)} \end{bmatrix} \tag{5.31}$$

By definition of this reordering, the mortar matrices associated with a given subdomain become

$$\boldsymbol{m}^{(i)} = \begin{bmatrix} \boldsymbol{m}_f^{(i)} & \boldsymbol{m}_m^{(i)} \end{bmatrix} = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{m}_m^{(i)} \end{bmatrix} \tag{5.32}$$

meaning that all zero columns in $\boldsymbol{m}^{(i)}$ are dropped from $\boldsymbol{m}_m^{(i)}$.

Storing the free and the constraint stiffness blocks in separate block diagonal matrices, i.e.

$$\begin{aligned} \mathsf{A}_{ff} &= \begin{bmatrix} \boldsymbol{K}_{ff}^{(1)} & 0 & 0 \\ 0 & \boldsymbol{K}_{ff}^{(2)} & 0 \\ & & \ddots \\ 0 & 0 & \boldsymbol{K}_{ff}^{(n_{sub})} \end{bmatrix} \qquad \mathsf{A}_{fm} = \begin{bmatrix} \boldsymbol{K}_{fm}^{(1)} & 0 & 0 \\ 0 & \boldsymbol{K}_{fm}^{(2)} & 0 \\ & & \ddots \\ 0 & 0 & \boldsymbol{K}_{fm}^{(n_{sub})} \end{bmatrix} \\ \mathsf{A}_{mf} &= \begin{bmatrix} \boldsymbol{K}_{mf}^{(1)} & 0 & 0 \\ 0 & \boldsymbol{K}_{mf}^{(2)} & 0 \\ & & \ddots \\ 0 & 0 & \boldsymbol{K}_{mf}^{(n_{sub})} \end{bmatrix} \qquad \mathsf{A}_{mm} = \begin{bmatrix} \boldsymbol{K}_{mm}^{(1)} & 0 & 0 \\ 0 & \boldsymbol{K}_{mm}^{(2)} & 0 \\ & & \ddots \\ 0 & 0 & \boldsymbol{K}_{mm}^{(n_{sub})} \end{bmatrix} \end{aligned} \tag{5.33}$$

and doing likewise for the force and displacement vectors with

$$\mathbf{u}_f = \begin{bmatrix} \tilde{\boldsymbol{u}}_f^{(1)} \\ \tilde{\boldsymbol{u}}_f^{(2)} \\ \vdots \\ \tilde{\boldsymbol{u}}_f^{(n_{sub})} \end{bmatrix} \qquad \mathbf{u}_m = \begin{bmatrix} \tilde{\boldsymbol{u}}_m^{(1)} \\ \tilde{\boldsymbol{u}}_m^{(2)} \\ \vdots \\ \tilde{\boldsymbol{u}}_m^{(n_{sub})} \end{bmatrix} \qquad \mathbf{f}_f = \begin{bmatrix} \tilde{\boldsymbol{f}}_f^{(1)} \\ \tilde{\boldsymbol{f}}_f^{(2)} \\ \vdots \\ \tilde{\boldsymbol{f}}_f^{(n_{sub})} \end{bmatrix} \qquad \mathbf{f}_m = \begin{bmatrix} \tilde{\boldsymbol{f}}_m^{(1)} \\ \tilde{\boldsymbol{f}}_m^{(2)} \\ \vdots \\ \tilde{\boldsymbol{f}}_m^{(n_{sub})} \end{bmatrix} \tag{5.34}$$

and the mortar matrices

$$\mathbf{B}_m = \begin{bmatrix} \boldsymbol{m}_m^{(1)} & \boldsymbol{m}_m^{(2)} & \dots \boldsymbol{m}_m^{(n_{sub})} \end{bmatrix} \tag{5.35}$$

the linear system of equations (5.2) can be rewritten as

$$\begin{bmatrix} \mathbf{A}_{ff} & \mathbf{A}_{fm} & \mathbf{0} \\ \mathbf{A}_{mf} & \mathbf{A}_{mm} & \mathbf{B}_m^T \\ \mathbf{0} & \mathbf{B}_m & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}_f \\ \mathbf{u}_m \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_f \\ \mathbf{f}_m \\ \mathbf{0} \end{bmatrix} \tag{5.36}$$

With a further concentration of the block expressions in eq. (5.36) to

$$\mathbf{Q} = \begin{bmatrix} \mathbf{A}_{mf} \\ \mathbf{0} \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} \mathbf{A}_{mm} & \mathbf{B}_m^T \\ \mathbf{B}_m & \mathbf{0} \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} \mathbf{u}_m \\ \boldsymbol{\lambda} \end{bmatrix} \qquad \mathbf{z} = \begin{bmatrix} \mathbf{f}_m \\ \mathbf{0} \end{bmatrix} \tag{5.37}$$

eq. (5.36) can be expressed as

$$\begin{bmatrix} \mathbf{A}_{ff} & \mathbf{Q}^T \\ \mathbf{Q} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \mathbf{u}_f \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_f \\ \mathbf{z} \end{bmatrix} \tag{5.38}$$

for which the Schur complement $\mathbf{S}_\mathrm{C}$ with respect to $\mathbf{C}$ and the respective right-hand side $\mathbf{g}_\mathrm{C}$ are

$$\mathbf{S}_\mathrm{C} = \mathbf{C} - \mathbf{Q}\mathbf{A}_{ff}^{-1}\mathbf{Q}^T \qquad\qquad \mathbf{g}_\mathrm{C} = \mathbf{z} - \mathbf{Q}\mathbf{A}_{ff}^{-1}\mathbf{f}_f \tag{5.39}$$

and hence, the corresponding Schur complement system is given by the expression

$$\begin{bmatrix} \mathbf{A}_{ff} & \mathbf{Q}^T \\ \mathbf{0} & \mathbf{S}_\mathrm{C} \end{bmatrix} \begin{bmatrix} \mathbf{u}_f \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_f \\ \mathbf{g}_\mathrm{C} \end{bmatrix} \tag{5.40}$$

which can be solved indepenently for $\mathbf{w}$ and in a subsequent step for $\mathbf{u}_f$ from

$$\mathbf{S}_\mathrm{C}\mathbf{w} = \mathbf{g}_\mathrm{C} \tag{5.41}$$

$$\mathbf{A}_{ff}\mathbf{u}_f = \mathbf{f}_f - \mathbf{Q}^T\mathbf{w} \tag{5.42}$$

Forming the Schur complement requires $\mathbf{A}_{ff}$ to be invertible. Since $\mathbf{A}_{ff}$ contains only those parts of the stiffness matrices that remain after eliminating all DOFs that are affected by (mortar) constraints, it is ensured to be symmetric positive-definite. Moreover, inverting $\mathbf{A}_{ff}$ as part of the solution process is the reason to rewrite the linear system (5.2) in the form of eq. (5.40). Since $\mathbf{A}_{ff}$ is block diagonal with individual blocks being symmetric positive-definite, the process

of building the inverse can be parallelized on the block matrix level. Nevertheless, it is a major cost factor. Yet, whenever a building structure is modified after an initial solution has already been obtained, the work of inverting the block matrices does not need to be done again for all blocks, but only for those whose associated patches are affected by the modifications. All other block matrices do not change, their inverted free stiffness blocks $\boldsymbol{K}_{ff}^{(i)^{-1}}$ can be reused in a new solution process, thus saving considerable numerical effort.

Re-expanding eq. (5.41) with the expressions (5.37) in conjunction with eq. (5.39) gives

$$\left[\begin{bmatrix} \mathbf{A}_{mm} & \mathbf{B}_m^T \\ \mathbf{B}_m & \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{A}_{mf} \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{ff}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{fm} & \mathbf{0} \end{bmatrix}\right] \begin{bmatrix} \mathbf{u}_m \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_m \\ \mathbf{0} \end{bmatrix} - \begin{bmatrix} \mathbf{A}_{mf} \\ \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{A}_{ff}^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{f}_f \end{bmatrix} \tag{5.43}$$

which after defining

$$\bar{\mathbf{A}} = \mathbf{A}_{mm} - \mathbf{A}_{mf}\mathbf{A}_{ff}^{-1}\mathbf{A}_{fm} \qquad \bar{\mathbf{f}} = \mathbf{f}_m - \mathbf{A}_{mf}\mathbf{A}_{ff}^{-1}\mathbf{f}_f \tag{5.44}$$

leads to the reduced inner problem of the substructuring approach

$$\begin{bmatrix} \bar{\mathbf{A}} & \mathbf{B}_m^T \\ \mathbf{B}_m & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u}_m \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{f}} \\ \mathbf{0} \end{bmatrix} \quad \text{with} \quad \bar{\boldsymbol{K}} = \begin{bmatrix} \bar{\mathbf{A}} & \mathbf{B}_m^T \\ \mathbf{B}_m & \mathbf{0} \end{bmatrix} \quad \bar{\boldsymbol{u}} = \begin{bmatrix} \mathbf{u}_m \\ \boldsymbol{\lambda} \end{bmatrix} \quad \bar{\boldsymbol{f}} = \begin{bmatrix} \bar{\mathbf{f}} \\ \mathbf{0} \end{bmatrix} \tag{5.45}$$

Observing that all matrices involved in the definition of $\bar{\mathbf{A}}$ are block diagonal matrices, it is to note that also $\bar{\mathbf{A}}$ is block diagonal and thus, all blocks can be constructed independently and in parallel. Unless Dirichlet boundary conditions were directly applied to a respective subdomain $\Omega^{(i)}$, the original subdomain stiffness matrices $\boldsymbol{K}^{(i)}$ are singular – and so are the corresponding subdomain blocks in $\bar{\mathbf{A}}$. Hence, eq. (5.45) is a saddle point problem with very similar properties as the original linear system (5.2). Yet, $\bar{\mathbf{A}}$ is only related to DOFs affected by mortar constraints. And since these are only a subset of those DOFs on the surface of the volumetric patches, the dimension of the problem is significantly reduced.

The linear system (5.45) has an analogy in the FETI-DP and IETI methods, where it represents an intermediate step on which a further reduction is performed. That second reduction leaves only the Lagrange multipliers as unknowns in the final system, which is then denoted the dual problem. Owing to the singularity of $\bar{\mathbf{A}}$, the second reduction cannot be done in a straightforward manner for eq. (5.45). With some additional effort, it is yet feasible. By identifying those DOFs in each subdomain block matrix $\bar{\mathbf{A}}^{(i)}$ that need to be constrained in order to produce a positive-definite submatrix, the outlined procedure starting with eq. (5.30) can be repeated. Though this does not reduce the system to only the Lagrange multiplier DOFs, it would allow for another reduction of the system size. As, however, solving eq. (5.45) in unaltered form is absolutely practicable, this is not essential in the scope of this framework.

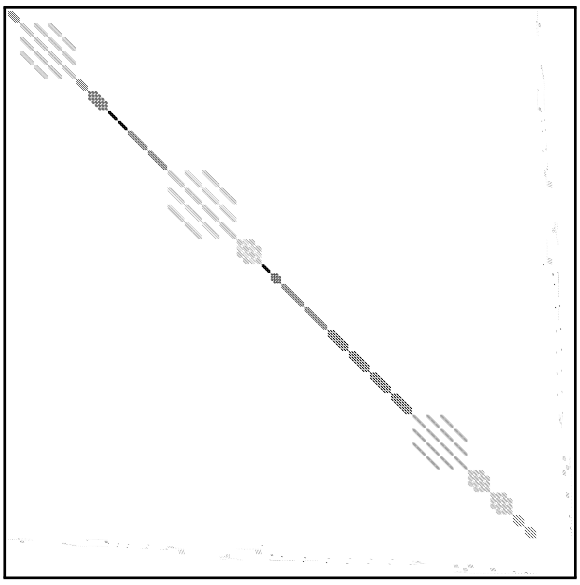
Since eq. (5.45) is of the same form as the original saddle point formulation (5.2), the iterative solution methods discussed in sect. 5.4.2 can be employed with minor modifications. With the severely reduced system size on the other hand, eq. (5.45) can also be solved with a direct method, which is desirable for already named reasons. Since all previous steps in this section can also be regarded as part of a direct solution approach, a coherent strategy for solving the large scale saddle point problem with a direct method is available.

Once the results of the inner problem (5.45) are available, the solution of the remaining unknown DOFs $\mathbf{u}_f$ is obtained from the expansion of eq. (5.42), leading to

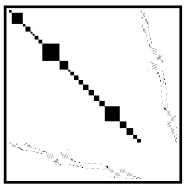
$$\mathbf{A}_{ff}\mathbf{u}_f = \mathbf{f}_{f,mod} \qquad \text{with} \qquad \mathbf{f}_{f,mod} = \mathbf{f}_f - \mathbf{A}_{fm}\mathbf{u}_m \tag{5.46}$$

which can be easily solved in parallel for the individual subdomains $\Omega^{(i)}$, as the blocks $\mathbf{A}_{ff}^{(i)}$ have previously been factored in the course of forming $\bar{\mathbf{A}}$.

The performance of the substructuring approach is shown on multistory example previously used in sect. 5.4.2.3 and in full detail explained in sect. 6.3.4. The coefficient matrices of the global linear system and of the inner problem derived with the substructuring approach are pictured in fig. 5.13. The inner problem is solved with an iterative as well as a direct method. For the iterative variant, the SQMR solver paired with the block diagonal factored preconditioner discussed in the previous section are used. As however, the blocks $\bar{\mathbf{A}}^{(i)}$ are known to be dense or nearly dense, the construction of the preconditioner is not done as an incomplete factorization, but the Cholesky LDLt factorization is used instead. In order to 'repair' the singularities of the block matrices in the inner problem, the parameter β_{tol} is set to 10^{-5}. For the direct solution of the inner problem, the SMP parallel variant of the SuperLU library [49] is used. That same solver is also applied to the global saddle point problem (5.2) in order to provide a comparison of the substructuring approach with a modern direct solver. As a fourth variant, the solution of the global saddle point problem with the iterative SQMR solver and the ILUTP preconditioner (cf. sect. 5.4.2.3) is included in the comparison.



(a) Global problem



(b) Inner problem

Figure 5.13: Size and sparsity pattern of the coefficient matrices from the global (a) and the inner problem (b) in the substructuring solution of the multistory example. The ratio of the differing matrix sizes (40,929 vs. 11,955 equations) is preserved in the images.

The time required for building and solving the linear system with the different solution strategies is displayed in fig. 5.14. The very efficient and parallel direct solver SuperLU, denoted 'global direct' in the graph, is outperformed by all approaches presented in this work. Comparing the iterative and the direct variant of solving the inner problem of the substructuring solver, one

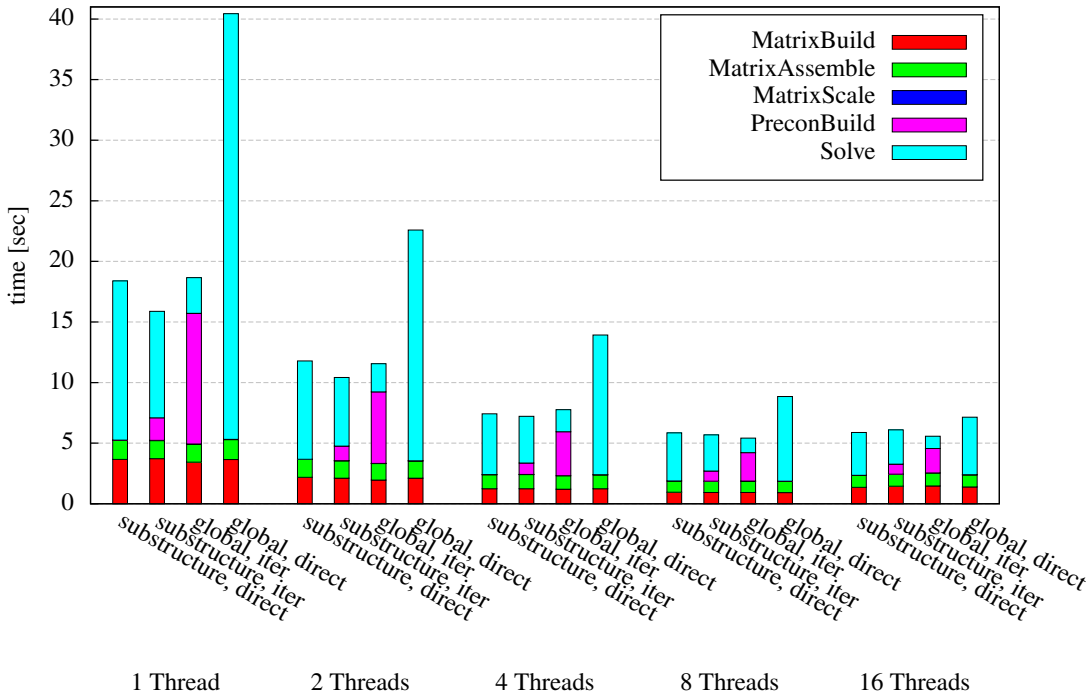


Figure 5.14: Time required for building and solving the linear system of the multistory example when direct and iterative variants of a global and the substructuring solvers are used.

can note a slight advantage for the inner iterative solver. Yet, the difference is not large and thus, there is good reason to favor the direct variant. The differing memory requirement is not as relevant when solving the inner problem. When however the limited memory resource is an issue for a given problem that prohibits its direct solution, the iterative solution strategy presented in the previous section is an alternative that performs equally well with regard to the overall computational time.

It remains to note, that the parallel execution of the substructuring solver relies on the subdomains defined by the physical problem. The time required for the sequential treatment of the largest subdomain limits the parallel scaling. From fig. 5.13 it can be seen, that the investigated problem includes three subdomains that are significantly larger than the others. These subdomains are the reason, why no further speed gains are achieved with the substructuring solver, when more than four parallel threads are used. The ‘Solve’ time duration simply represents the time for handling the largest of these subdomains – this is a problem that does not exist for the general purpose direct solver. However, with a growing number of approximately equally sized subdomains and a number of parallel threads that does not increase in a likewise manner, the issue becomes less relevant.

5.5 Analysis result processing

Solving the global linear system of equations of the standard displacement-based finite element method provides the nodal solution required (cf. eq. (3.32)) to describe the displacement field of a physical body under the action of an applied load state. Yet, the main interest is usually

not on the displacements but on the stress state of that body. With eq. (3.42), which is repeated here for convenience,

$$\underline{\boldsymbol{\sigma}}^h(\boldsymbol{P}) = \underline{\boldsymbol{D}}\boldsymbol{B}\tilde{\boldsymbol{u}}^e \qquad \forall \boldsymbol{P} \in \Omega_e$$

the stress state at any point of the body can be computed from the nodal displacement solution. However, for reasons of result accuracy, which are discussed in sect. 6.4.2, it is common practice not to express the entire stress field by eq. (3.42). The expression is rather employed to determine the stress state at the nodal points. The nodal point stress vectors $\hat{\underline{\boldsymbol{\sigma}}}^{(a)}$ are then used to express the body's stress field with the help of the standard shape functions as

$$\underline{\boldsymbol{\sigma}}^h(\boldsymbol{P}) = \boldsymbol{N}\tilde{\boldsymbol{\sigma}}^e \qquad \forall \boldsymbol{P} \in \Omega_e \tag{5.47}$$

where $\tilde{\boldsymbol{\sigma}}^e$ denotes a vector containing all nodal point stress vectors associated with element e. Typically, the nodal point values are not computed directly from eq. (3.42), but improved values are determined from a defined set of surrounding points by averaging their values or computing a least square fit, see Bathe [17, sec. 4.3.6] for details. Apart from using the improved nodal point stress values, expression (5.47) has the advantage of being cheaper to evaluate than eq. (3.42) and having a higher interpolation order at the same time.

With IGA, the solution of the saddle point problem (5.2) provides the control variables $\boldsymbol{Q}^{(i,j,k)}$. An expression equivalent to eq. (3.42) results from replacing the displacements with eq. (3.99), leading to

$$\underline{\boldsymbol{\sigma}}^h(\boldsymbol{P}) = \underline{\boldsymbol{D}}\boldsymbol{B}\tilde{\boldsymbol{q}}^e \qquad \forall \boldsymbol{P} \in \Omega_e \tag{5.48}$$

A direct computation of related stress control variables from that expression is not possible to begin with, as the control variables are not embodied by the physical domain. If one is not to calculate all points of the stress field from eq. (5.48), other methods must by applied. In this work, an algorithm originally proposed by Piegl and Tiller [129, sec. 9.4.3] is adapted to work with NURBS solids and weights for the purpose of gradient field recovery.

The original purpose of that algorithm is to approximate a given set of points with a B-spline surface. Piegl and Tiller argue that solving the full least square problem for that approximation can become expensive and that reasonable results are also obtained by evaluating the parametric directions sequentially. Here, solid domains are to be handled for which the computation of a full least square fit would be even more complex and hence, their strategy is retained. For the specific problem of this framework, the already existing parametrizations of the solid NURBS patches are to be used to express the stress field on the defined physical (sub-)domain. Thus, the polynomial degrees, the knot vectors, and the number of control variables as well as their weights are predefined and the values of the control variables are to be determined by the algorithm for the individual patches.

Recalling the definition of a NURBS volume to be

$$\boldsymbol{V}(\xi,\eta,\zeta) = \sum_{i=1}^{n_{cp}} \sum_{j=1}^{m_{cp}} \sum_{k=1}^{l_{cp}} R_{p,q,r}^{(i,j,k)}(\xi,\eta,\zeta)\, \boldsymbol{P}^{(i,j,k)} \qquad \forall (\xi,\eta,\zeta) \in \hat{\Omega}^3.$$

which is associated with knot vectors $\boldsymbol{\Xi}$, $\mathbf{H}$, and $\mathbf{Z}$ and polynomial degrees p, q, and r. $\boldsymbol{P}^{(i,j,k)}$ denotes a control point within the lattice of $n_{cp} \times m_{cp} \times l_{cp}$ control points. A corresponding

control variable for the stress shall be denoted $\boldsymbol{T}^{(i,j,k)}$ such that the stress field of the domain can be given as

$$\underline{\boldsymbol{\sigma}}^h(\xi,\eta,\zeta) = \sum_{i=1}^{n_{cp}} \sum_{j=1}^{m_{cp}} \sum_{k=1}^{l_{cp}} R_{p,q,r}^{(i,j,k)}(\xi,\eta,\zeta)\, \boldsymbol{T}^{(i,j,k)} \qquad \forall (\xi,\eta,\zeta) \in \hat{\Omega}^3. \tag{5.49}$$

Stress tensors at the location of the integration points are used as the basis of the gradient field recovery. Their numerical values can be evaluated from eq. (5.48). Due to the tensor product structure of the patches, also the integration points are arranged in a structured lattice. Integration point $P_{ip}^{(u,v,w)}$ is defined by its parametric coordinates $\xi_{ip}^{(u)}$, $\eta_{ip}^{(v)}$, and $\zeta_{ip}^{(w)}$. The total number of integration points in a single patch is given by $n_{ip} \times m_{ip} \times l_{ip}$. A specific stress tensor in that lattice is then denoted by $\underline{\boldsymbol{\sigma}}_{ip}^{(u,v,w)}$.

The evaluation of the $n_{cp} \times m_{cp} \times l_{cp}$ control variables $\boldsymbol{T}^{(i,j,k)}$ from the $n_{ip} \times m_{ip} \times l_{ip}$ stress tensors at the integration points is done successively for the three parametric dimensions of one patch. In case of a multi-patch domain, individual patches are treated independently. Starting with the ξ-direction, an array of size $n_{cp} \times m_{ip} \times l_{ip}$ is defined, that will store temporary control variables $\boldsymbol{T}_{tmp,1}$ after fitting the first dimension. Values on the boundary of that parametric direction are set as

$$\boldsymbol{T}_{tmp,1}^{(1,v,w)} = \underline{\boldsymbol{\sigma}}_{ip}^{(1,v,w)} \qquad \text{and} \qquad \boldsymbol{T}_{tmp,1}^{(n_{cp},v,w)} = \underline{\boldsymbol{\sigma}}_{ip}^{(n_{ip},v,w)} \tag{5.50}$$

and the remaining $n_{cp} - 2 \times m_{ip} \times l_{ip}$ temporary control variables are computed from

$$\underset{\boldsymbol{T}_{tmp,1}^{(i,v,w)}}{\arg\min}\, f\left(\boldsymbol{T}_{tmp,1}^{(i,v,w)}\right) \qquad \forall\, (v,w) \in \left\{1,\dots,m_{ip}\right\} \times \left\{1,\dots,l_{ip}\right\} \tag{5.51}$$

with

$$\begin{aligned} f\left(\boldsymbol{T}_{tmp,1}^{(i,v,w)}\right) &= \sum_{u=2}^{n_{ip}-1} \left| \underline{\boldsymbol{\sigma}}_{ip}^{(u,v,w)} - \underline{\boldsymbol{\sigma}}^h\left(\xi_{ip}^{(u)}, \eta_{ip}^{(v)}, \zeta_{ip}^{(w)}\right) \right|^2 \\ &= \sum_{u=2}^{n_{ip}-1} \left| \underline{\boldsymbol{\sigma}}_{ip}^{(u,v,w)} - \sum_{i=1}^{n_{cp}} \sum_{j=1}^{m_{cp}} \sum_{k=1}^{l_{cp}} R_{p,q,r}^{(i,j,k)}\left(\xi_{ip}^{(u)}, \eta_{ip}^{(v)}, \zeta_{ip}^{(w)}\right) \boldsymbol{T}_{tmp,1}^{(i,v,w)} \right|^2 \\ &= \sum_{u=2}^{n_{ip}-1} \left| \underline{\boldsymbol{\sigma}}_{ip}^{(u,v,w)} - \sum_{i=1}^{n_{cp}} R_p^{(i)}\left(\xi_{ip}^{(u)}\right) \boldsymbol{T}_{tmp,1}^{(i,v,w)} \right|^2 \end{aligned} \tag{5.52}$$

in which the solid NURBS basis functions reduce to those of a curve in the first parametric dimension, as v and w are fixed (cf. sect. 3.4.6).

Defining

$$\boldsymbol{\varphi}^{(u)} = \underline{\boldsymbol{\sigma}}_{ip}^{(u,v,w)} - R_p^{(1)}\left(\xi_{ip}^{(u)}\right) \boldsymbol{T}_{tmp,1}^{(1,v,w)} - R_p^{(n_{cp})}\left(\xi_{ip}^{(u)}\right) \boldsymbol{T}_{tmp,1}^{(n_{cp},v,w)} \qquad \forall u \in \{2,\dots,n_{cp}-1\} \tag{5.53}$$

and introducing that into eq. (5.52) results in

$$f\left(\boldsymbol{T}_{tmp,1}^{(i,v,w)}\right) = \sum_{u=2}^{n_{ip}-1} \left| \boldsymbol{\varphi}^{(u)} - \sum_{i=2}^{n_{cp}-1} R_p^{(i)}\left(\xi_{ip}^{(u)}\right) \boldsymbol{T}_{tmp,1}^{(i,v,w)} \right|^2 \tag{5.54}$$

which is an expression that contains the $n_{cp} - 2$ temporary control variables $\boldsymbol{T}_{tmp,1}^{(i,v,w)}$, as the only unknowns given a fixed pair of indices (v, w).

Defining the matrix

$$\boldsymbol{N} = \begin{bmatrix} R_p^{(2)}\left(\xi_{ip}^{(2)}\right) & R_p^{(3)}\left(\xi_{ip}^{(2)}\right) & \cdots & R_p^{(n_{cp}-1)}\left(\xi_{ip}^{(2)}\right) \\ R_p^{(2)}\left(\xi_{ip}^{(3)}\right) & R_p^{(3)}\left(\xi_{ip}^{(3)}\right) & \cdots & R_p^{(n_{cp}-1)}\left(\xi_{ip}^{(3)}\right) \\ \vdots & \vdots & \ddots & \vdots \\ R_p^{(2)}\left(\xi_{ip}^{(n_{ip}-1)}\right) & R_p^{(3)}\left(\xi_{ip}^{(n_{ip}-1)}\right) & \cdots & R_p^{(n_{cp}-1)}\left(\xi_{ip}^{(n_{ip}-1)}\right) \end{bmatrix} \tag{5.55}$$

with size $(n_{ip} - 2) \times (n_{cp} - 2)$ and the block vectors

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{\varphi}^{(2)^T} \\ \boldsymbol{\varphi}^{(3)^T} \\ \vdots \\ \boldsymbol{\varphi}^{(n_{ip}-1)^T} \end{bmatrix} \qquad \boldsymbol{p} = \begin{bmatrix} \boldsymbol{T}_{tmp,1}^{(2,v,w)^T} \\ \boldsymbol{T}_{tmp,1}^{(3,v,w)^T} \\ \vdots \\ \boldsymbol{T}_{tmp,1}^{(n_{cp}-1,v,w)^T} \end{bmatrix} \tag{5.56}$$

with $\boldsymbol{z}$ having length $(n_{ip} - 2)$, $\boldsymbol{p}$ length $(n_{cp} - 2)$ and both – corresponding to the number of stress tensor components – the width 6, the objective function f can be reformulated as

$$f\left(\boldsymbol{T}_{tmp,1}^{(i,v,w)}\right) = \|\boldsymbol{z} - \boldsymbol{N}\boldsymbol{p}\|^2 \tag{5.57}$$

from which a least square fit to eq. (5.51) can be obtained by solving the normal equations [71]

$$\boldsymbol{N}^T\boldsymbol{N}\boldsymbol{p} = \boldsymbol{N}^T\boldsymbol{z} \tag{5.58}$$

with $(n_{cp} - 2)$ unknowns and a dense, symmetric, positive-definite coefficient matrix $\boldsymbol{N}^T\boldsymbol{N}$.

Setting up and solving eq. (5.58) $m_{cp} \times l_{cp}$ times provides all missing control variables $\boldsymbol{T}_{tmp,1}^{(i,u,v)}$ for the fit in the first parametric dimension. Once these are computed, the least square fit is performed in a similar manner on the data in $\boldsymbol{T}_{tmp,1}^{(i,u,v)}$ to evaluate a second array of temporary control variables $\boldsymbol{T}_{tmp,2}^{(i,j,v)}$ with size $n_{cp} \times m_{cp} \times l_{ip}$. Starting with

$$\boldsymbol{T}_{tmp,2}^{(i,1,w)} = \boldsymbol{T}_{tmp,1}^{(i,1,w)} \qquad \text{and} \qquad \boldsymbol{T}_{tmp,2}^{i,(m_{cp},w)} = \boldsymbol{T}_{tmp,1}^{(i,n_{ip},w)} \tag{5.59}$$

the remaining $n_{cp} \times m_{cp} - 2 \times l_{ip}$ temporary control variables are computed as before from

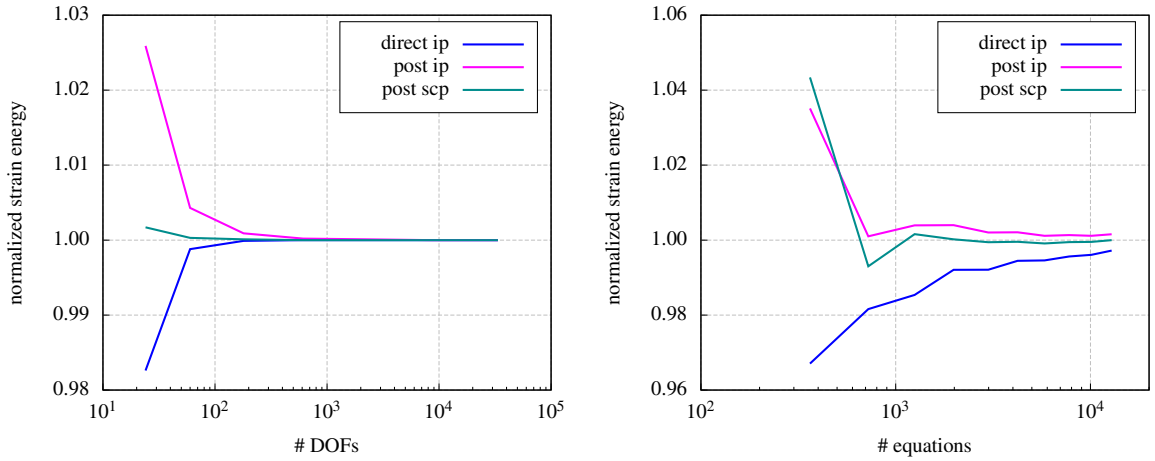
$$\underset{\boldsymbol{T}_{tmp,2}^{(i,j,w)}}{\arg\min}\, f\left(\boldsymbol{T}_{tmp,2}^{(i,j,w)}\right) \qquad \forall\,(i,w) \in \left\{1,\ldots,n_{cp}\right\} \times \left\{1,\ldots,l_{ip}\right\} \tag{5.60}$$

With this second array of temporary control variables being available, the procedure is repeated a last time for the third parametric dimension in order to obtain the final stress control variables $\boldsymbol{T}^{(i,j,k)}$ from the temporary data $\boldsymbol{T}_{tmp,2}^{(i,j,w)}$.

Computing the fitted stress control variables from the integration point stress tensors allows to express the stress field of any (sub-)domain by eq. (5.49). Though this computation involves solving quite a few linear systems, their sizes are generally very small when compared to the global linear system and hence, the computation is fast. Furthermore, it can be done in parallel for individual subdomains, i.e. NURBS patches. Also within the subdomains, there is potential for parallelization which, however, was not further exploited. Obviously, the same algorithm can be used to approximate other than stress data with a NURBS interpolation and also, the data basis for that approximation can be given at other locations than the integration points. The only requirement for these points is to be arranged in a structured grid. Their number must be larger than that of the control points in the respective parametric dimension.

According to eq. (5.50), the control variables on the boundary are directly set from the respective outmost integration point values. Especially in the case of coarse discretizations, this procedure can introduce a relevant error, as the recovered stress field only represents that part of the original field, which is enveloped by the integration points. Stresses between the domain boundary and the envelope are ignored in the recovery process. For that reason, it was found to significantly improve the recovered fields, when additional sampling points located on the patch boundary are introduced to the lattice of integration points, yet only when the degree of the basis functions in the respective parametric direction is greater than one, i.e. $p \geq 2$. In the linear case, the accuracy of the gradients evaluated on the boundary is too low to improve the overall quality of the recovered field.

Apart from smoothing the solution data and simplifying the representation of gradient fields, the definition of the solution fields in terms of control variables also has the advantage to enable a sophisticated visualization of the volumetric domain results. Schollmeyer and Froehlich [147], for example, provide a related algorithm, which allows the direct GPU rendering of transparent isosurfaces for control variable based result data in isogeometric analysis. Such visualization methods are especially helpful to inspect the results within solid bodies.



(a) Cantilever beam

(b) Solid cube coupling

Figure 5.15: Comparison of the normalized strain energy evaluated by three different methods on the single patch variant of the cantilever beam example and on the cube coupling example. The methods are explained in the text.

In order to demonstrate the quality of the fitted and thus smoothed solution data, strain energy results for the single patch cantilever beam example (cf. sect. 4.5.1) and the cube coupling

example (cf. sect. 4.5.3) are compared in fig. 5.15. For both examples, the strain energy stored in the domain is evaluated by utilizing three different approaches.

direct ip The strain energy density evaluated at the integration points is integrated directly by Gauss quadrature.

post ip The strain energy density evaluated at all integration points is used as input data to the previously outlined algorithm. The resulting strain energy density field in terms of control variables is used to once again evaluate the strain energy density at the integration points. These postprocessed values are then integrated by Gauss quadrature.

post scp The same procedure as before is used, yet the input data is not evaluated at the integration points but at the superconvergent points, which is discussed in detail in the next chapter (cf. table 6.5).

Full integration and quadratic basis functions are used in all cases. It is to be noted that no smoothing is performed across subdomain boundaries, i.e. across mortar interfaces. In general, the recovered fields tend to overestimate the true strain energy, while the directly evaluated results underestimate the true solution. For these two examples, the strain energy obtained by integrating the postprocessed integration point results is of approximately the same accuracy as the directly integrated data. Both variants are outperformed by the postprocessed superconvergent point result data.

Chapter 6

Refinement strategies

6.1 Introduction

Mesh refinement is an essential topic of the integrated analysis approach. Operations that modify the properties of a given NURBS patch with regard to the degree of the basis functions and the size of the inherent finite elements are presented in sect. 3.5.5. This chapter is dedicated to the strategies that steer the associated refinement process. It is discussed how to decide, when and where refinement operations have to be applied.

Patches in a geometric model are expected to be as "fine" as to correctly describe the geometry of the structural entity they represent. Establishing the finite element matrices for these patches is possible, independent of the refinement level, as is computing the solution of the associated finite element problem. The accuracy of the obtained results, however, depends strongly on the function spaces used within the finite element model and thus on the inherent discretization of the NURBS patches. Since these patches are typically not created with their employment in an isogeometric analysis in mind, but are rather meant for geometry representation only, the level of initial patch refinement does commonly not suffice for an acceptable result accuracy – which motivates the need for patch refinement prior to the analysis. The general objective of the refinement process is to provide a discretized model that allows the computation of solution fields, which are of equal quality over the entire domain. Since the description of such fields is likely to require varying functions spaces, the optimal mesh is not going to be homogeneous.

Limiting the numerical cost of the analysis constitutes another objective during the creation of the finite element model – one, that conflicts with result accuracy. Enriching the function spaces increases the cost of the analysis. A suitable refinement strategy should thus balance both: Patches shall be sufficiently fine to obtain acceptable results but not any finer than that, in order to keep the numerical cost as low as possible.

In this context, result accuracy is to be understood antonymously with the discretization error, i.e. the error that arises when solving a continuous mechanical problem on the basis of discrete variables. Both, reducing the spacing of the discrete variables and thus the size of the finite elements and increasing the order of interpolation for the elements reduces the discretization error. In the limit of zero spacing, the discretization error should vanish completely. Other sources of erroneous results, such as round-off errors due to the representation of rational numbers with the limited precision of computer systems or modeling errors originating in an abstraction that represents the underlying physical phenomenon only insufficiently are not subject of this chapter.

In the light of the analysis costs, it is of course unfeasible to reduce the element size close to zero and thus to nearly eliminate the discretization error. Neither is it possible to precalculate a refinement level that leads to results with a defined accuracy. A priori error estimates merely provide information about the theoretical convergence behavior [73, sec. 3.2], the prediction of absolute error values for a given problem is not possible. Such values can, at least, be estimated with an a posteriori error estimator.

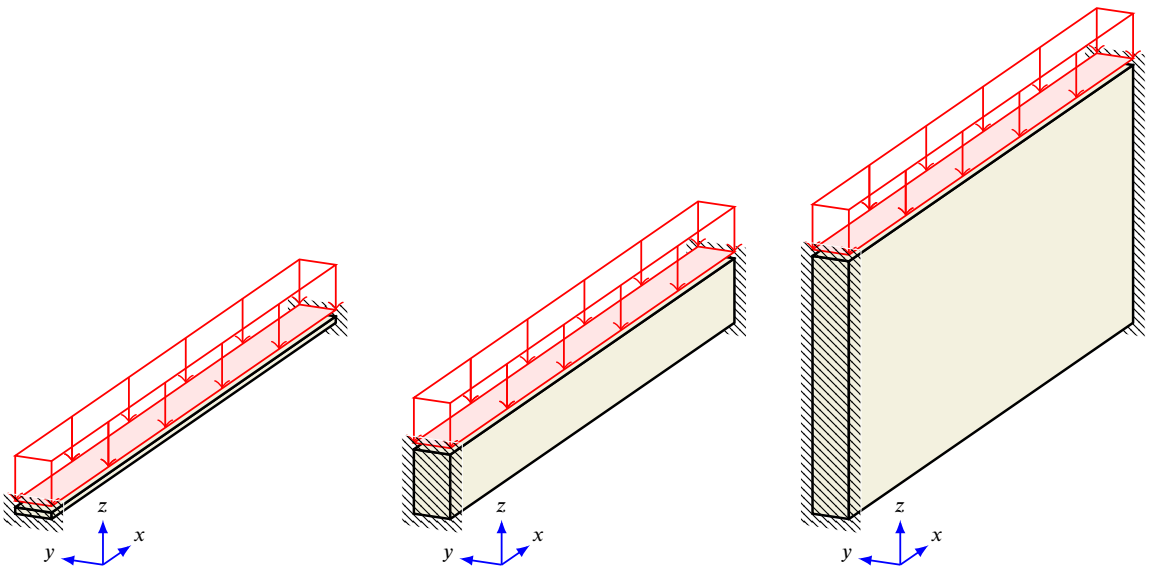
Independent of the type of a posteriori error estimator and the significance of its estimates, there are two drawbacks associated with the application in practical structural calculations. The linear system must be solved before any error estimate becomes available. In consequence, at least a second solution iteration is required to make use of error estimation based model refinement. With regard to the large volumetric models and the cost for their analysis discussed in chapter 5, this may be undesirable. The second drawback emanating from the nature of the estimator and the finite element method itself, is the restriction of the error estimate's validity to a specific load situation. In the analysis of civil engineering structures, it is common practice and a consequence of existing regulations, to evaluate many different load cases and to superpose the results in numerous variants for the dimensioning of the structure. Using an error estimator to determine the need for model refinement is likely to result in diverging refinement indications, each of these satisfying the peculiarities of a specific load case only. In consequence, this procedure would lead to different model discretizations with distinct linear systems that have to be computed and solved for each load variant. The superposition of results from these analyses with a multitude of underlying meshes introduces additional complexity. The alternative of combining all indicated refinements in a single mesh for all load variants, however, counteracts the general idea of refining the model only selectively.

With these considerations, it seems expedient to use a rather traditional approach for the transformation of the initial discretization inherent to the patches into one that allows the computation of meaningful results – an approach which is based on the knowledge and the experience of the structural engineer. A methodology to combine this procedure with isogeometric analysis and an automatic model extraction from the BIM data set is presented in this chapter. Before doing so, the potential and effectivity of anisotropic patch refinement, which is an obvious refinement option in IGA, is investigated on a simple solid beam example. And despite of its presumably restricted applicability in practical work, also the use of an error estimator that is specific to problems of IGA is investigated. The related discussion is presented at the end of this chapter.

6.2 Anisotropic refinement example

A solid beam of length $l = 5.0$ is clamped at both ends and loaded by a constant surface load on its top face. The cross sectional height is considered as $h = 0.05$, $h = 0.50$, and $h = 2.00$, whereas the width of the rectangular cross section is constant at $w = 0.3$ and thus the same for all three model variants. The distinct models are pictured in fig. 6.1. Their material behavior is assumed linear-elastic and isotropic, defined by the parameters Young's modulus $E = 3.0 \cdot 10^{10}$ and Poisson ratio $\nu = 0.25$.

The simple model is expressed by single NURBS patch that has its parametric ξ, η, and ζ axes aligned with the global x, y, and z coordinate axes. At the coarsest discretization, the beam can



(a) $h = 0.05$ **(b)** $h = 0.50$ **(c)** $h = 2.00$

Figure 6.1: Model problem – clamped solid beam with constant surface load on top face and varying beam height h.

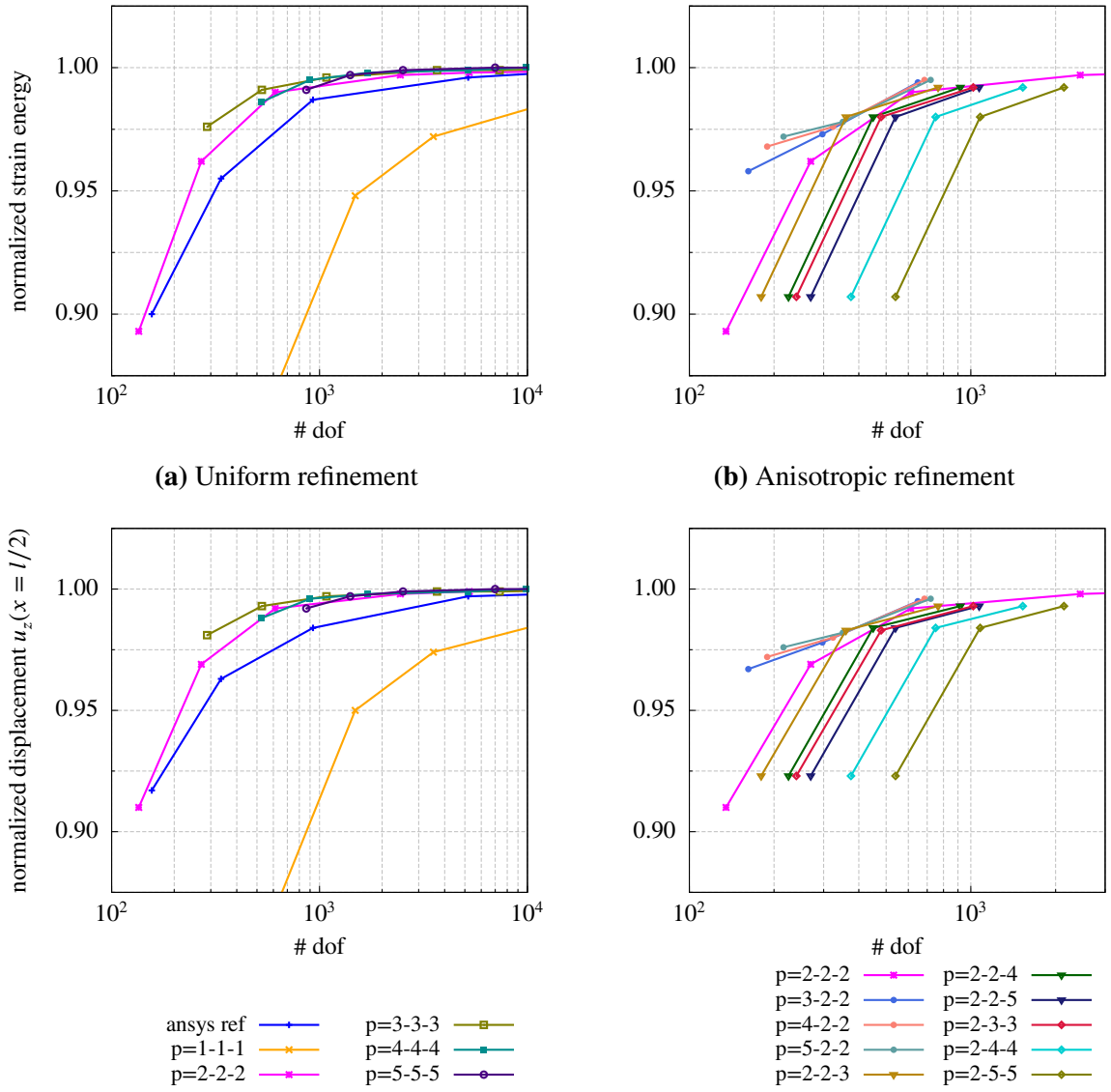
be modeled with linear basis functions and a single element, which, however, is unreasonable for a numerical analysis. Specifying the maximum element edge length as $l_e = 1.0$ defines a coarse analysis model that has a more appropriate discretization. A further reduction of the value of l_e refines the model. The number of elements per beam axis resulting from the employed values of l_e are listed in table 6.1.

max element edge length l_e	$h = 0.05$			$h = 0.50$			$h = 2.00$		
	x	y	z	x	y	z	x	y	z
1.000	5	1	1	5	1	1	5	1	2
0.500	10	1	1	10	1	1	10	1	4
0.300	17	1	1	17	1	2	17	1	7
0.150	34	2	1	34	2	4	34	2	14
0.100	50	3	1	50	3	5	50	3	20
0.050	100	6	1	100	6	10	100	6	40
0.025	200	12	2	200	12	20	200	12	80

Table 6.1: Number of finite elements used to discretize the solid beam model in the direction of the respective beam axis.

The discretizations variants listed in table 6.1 represent a natural approach to the uniform refinement of the solid beam. In this example, these variants are analyzed for basis functions of degree $p = 1$ to $p = 5$. The results obtained with this procedure are then compared to results from respective analyses in which models with the same number of elements but selectively raised degrees of the basis functions were used. The purpose of this investigation is to evaluate the impact of anisotropic p-refinement, which is an apparent option with solid isogeometric analysis.

Figure 6.2 visualizes the normalized strain energy and vertical displacements $u_z(0.5l, 0.5w, 0)$ of the beam model with height $h = 0.50$. For a reference, all model variants were also computed



(a) Uniform refinement

(b) Anisotropic refinement

(c) Uniform refinement

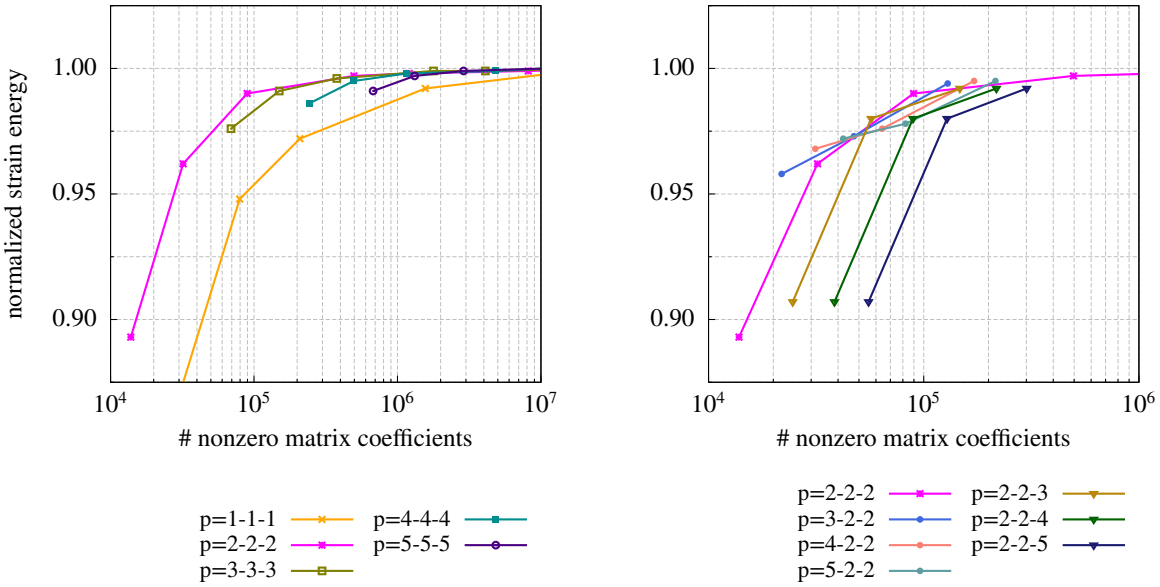
(d) Anisotropic refinement

Figure 6.2: Normalized strain energy, (a) and (b), and displacement, (c) and (d), results vs. degrees of freedom for the model problem with beam height $h = 0.50$, cf. fig. 6.1(b), under uniform and anisotropic refinement. The uniform refinement results are compared with an Ansys reference solution, whereas, for comparison, the anisotropic refinement results are plotted together with the uniform IGA solution obtained with NURBS of degree $p = q = r = 2$. The key $p = a - b - c$ indicates the degree of the basis functions in the parametric ξ, η, and ζ directions that coincide with the beam's longitudinal (x), width (y), and height (z) axes.

with Ansys.[1] As of the simple model geometry, the same structured brick type discretization that is common in IGA could also be realized with Ansys. In fig. 6.2, the left plots represent the uniform refinement. Using the number of DOFs in the system as a measure of numerical cost, there is a notable performance difference between linear and quadratic elements, but for basis functions with degree $p > 2$, this difference is only marginal at coarse discretizations and

[1] Ansys Mechanical APDL, v16.0, ANSYS Inc., www.ansys.com
SOLID186 element, 20-node structural solid element with quadratic displacement behavior.

vanishes almost completely for reduced element edge lengths. Anisotropic refinement results are displayed on the right. As of its good performance in uniform refinement, the model with quadratic basis functions in all parametric directions is used as the base for the anisotropic refinement. The degree of the basis functions is then raised only in longitudinal direction (x-axis), only in the direction of the load (z-axis), or in both cross sectional directions (y- and z-axis). A significant performance improvement is only observed for degree elevation in longitudinal direction. Raising the degree in both cross sectional directions is even detrimental under performance aspects. And apparently, the observed behavior is similar for strain energy and displacement results. Results for the latter are therefore not plotted in the graphs that are yet to be shown for this example.



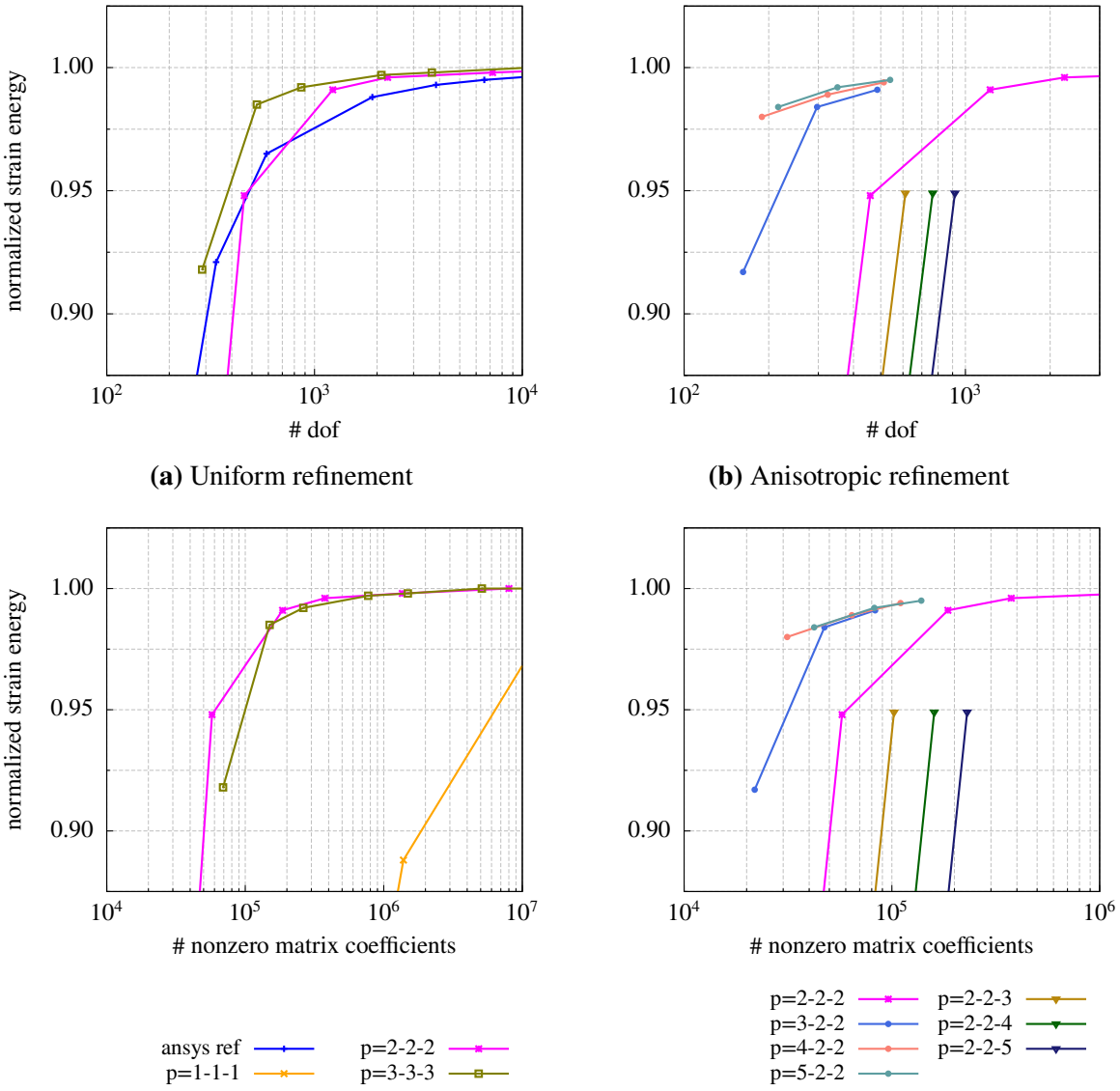
(a) Uniform refinement

(b) Anisotropic refinement

Figure 6.3: Normalized strain energy results for the model problem with beam height $h = 0.50$, cf. fig. 6.1(b), under uniform and anisotropic refinement. In contrast to fig. 6.2, the result data is plotted in dependence of the number of nonzero coefficients in the stiffness matrix.

Using the number of nonzero coefficients in the global stiffness matrix to measure the numerical cost, see fig. 6.3(a), shifts the result curve for linear elements a lot closer to those of higher degree elements and the uniform quadratic interpolation outperforms all other degree basis functions at any discretization level. This also reflects in the anisotropic refinement results on the right (fig. 6.3(b)). The slight advantage for degree elevation in longitudinal beam direction disappears for all but the coarsest discretization. Degree elevation in z-direction only is clearly inferior to uniform refinement. The curves associated with degree elevation in both cross sectional axes are not shown for the clarity of the plot.

The study on strain energy convergence is repeated for the model with beam height $h = 0.05$, for which the slenderness l/h rises from 10 to 100. For uniform refinement, the results plotted in figs. 6.4(a) and 6.4(c) display a notable difference to those of the previous model: They are strikingly poor at coarse discretizations and for linear elements, they are to a large extent even outside the plot range. Obviously, this behavior is caused by shear locking. In such a situation with high slenderness, anisotropic refinement in longitudinal direction can alleviate



(a) Uniform refinement

(b) Anisotropic refinement

(c) Uniform refinement

(d) Anisotropic refinement

Figure 6.4: Normalized strain energy results vs. degrees of freedom, (a) and (b), or nonzero matrix coefficients, (c) and (d), for the model problem with beam height $h = 0.05$, cf. fig. 6.1(a), under uniform and anisotropic refinement. The key $p = a - b - c$ indicates the degree of the basis functions in the parametric ξ, η, and ζ directions that coincide with the beam's longitudinal (x), width (y), and height (z) axes.

the negative shear locking effects, which is illustrated by the anisotropic refinement plots in figs. 6.4(b) and 6.4(d). Raising the degree in longitudinal direction to $p = 3$ eases the negative influence, and for $p = 4$ and above, shear locking is not visible anymore. This coincides with the findings reported by Echter and Bischoff [60].

In contrast to the thin beam, the model with height $h = 2.00$ represents a variation of the beam example with very low slenderness for which shear locking is irrelevant. Associated uniform refinement results are plotted on the left of fig. 6.5, anisotropic refinement results on the right. Again, it is observed that the variant with quadratic degree basis functions under uniform refinement is the most effective in terms of result accuracy versus nonzero stiffness matrix

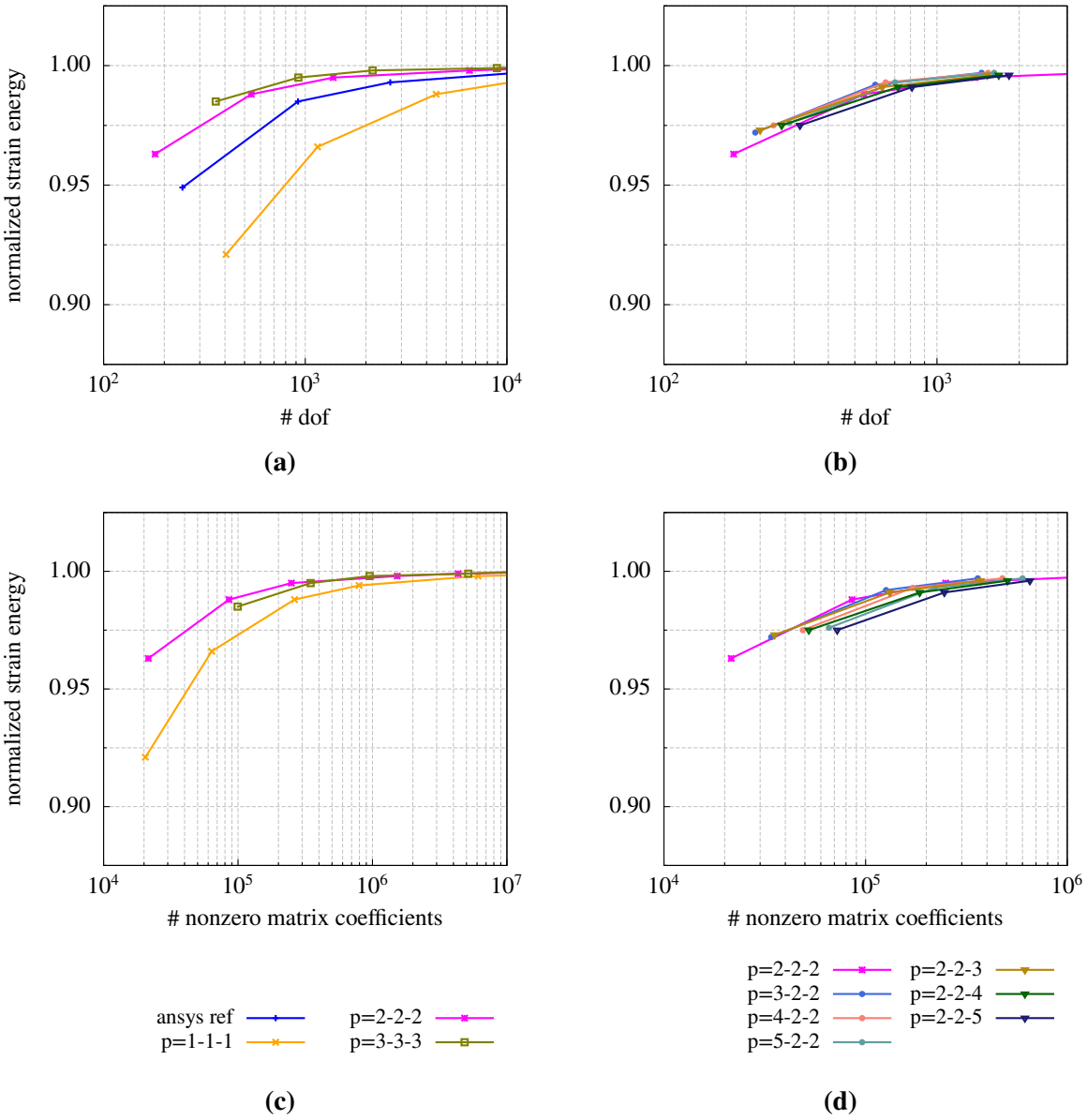


Figure 6.5: Normalized strain energy results vs. degrees of freedom, (a) and (b), or nonzero matrix coefficients, (c) and (d), for the model problem with beam height $h = 2.00$, cf. fig. 6.1(c), under uniform and anisotropic refinement. The key $p = a - b - c$ indicates the degree of the basis functions in the parametric ξ, η, and ζ directions that coincide with the beam's longitudinal (x), width (y), and height (z) axes.

coefficients. The advantage over linear elements is, however, reduced compared to previous models. Anisotropic refinement has no relevant effect, neither positive nor negative.

The uniform refinement achieved by defining a maximum element edge length leads to several layers of elements over the beam's height when $h = 2.00$, which, of course, introduces additional DOFs, when compared to $h = 0.05$ and $h = 0.50$, where this is not the case at coarse refinement levels. For that reason, the behavior of the $h = 2.00$ model variant is also investigated with each cross sectional plane represented by single element only. The corresponding results are plotted in fig. 6.6. In that case, higher basis function degrees in longitudinal direction have a detrimental effect, as they apparently introduce additional DOFs without improving

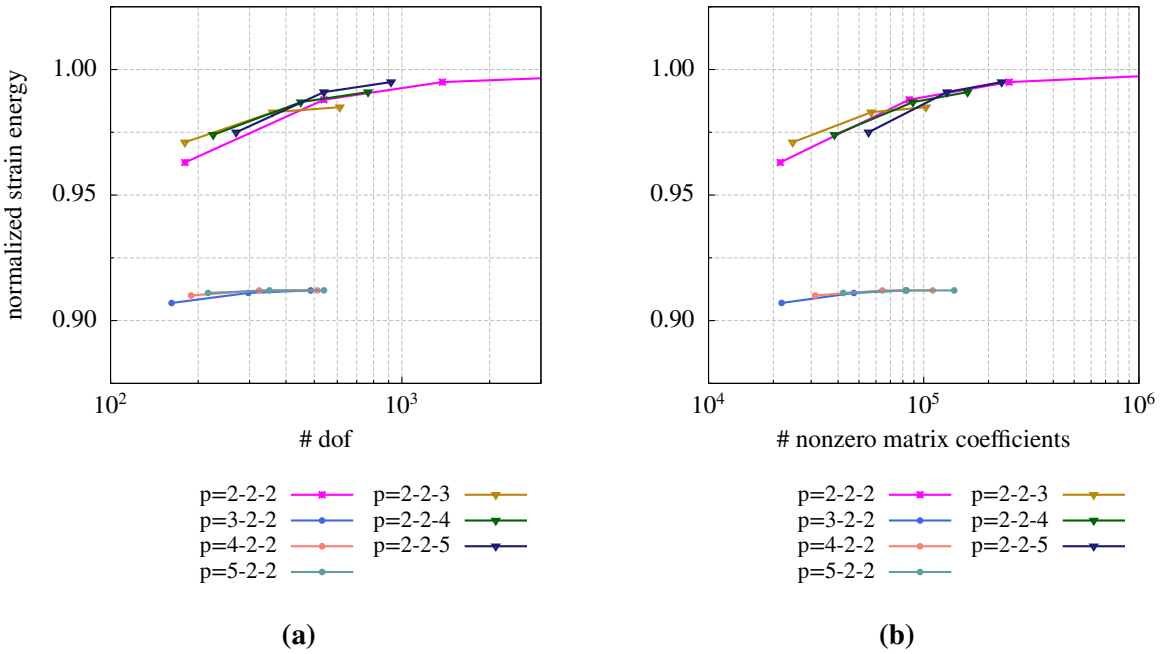


Figure 6.6: Normalized strain energy results vs. degrees of freedom (a) or nonzero matrix coefficients (b) for the model problem with beam height $h = 2.00$, cf. fig. 6.1(c), under anisotropic refinement. In contrast to fig. 6.5, the beam's height is discretized with a single element at any h-refinement level. The curve for $p = 2 - 2 - 2$ represents the original discretization from fig. 6.5 and is included for reference.

the result accuracy accordingly. Raising the degree in z-axis, on the other hand, compensates the discretization of the beam's height with a single element. Performance advantages over the original discretization with multiple elements over the height, however, remain marginal but do exist when compared to the uniformly refined model.

In conclusion of this convergence study it is to note that anisotropic p-refinement does not relevantly contribute to an improved model performance unless locking effects are an issue. In that case, however, anisotropic p-refinement is essential insofar as a standard displacement element formulation is used.

6.3 Automated empirical refinement

6.3.1 Refinement for geometrical types

Regardless of the proposed volumetric modeling of structures for the purpose of their analysis, the individual building elements making up a complete structure can be categorized according to their spatial dimensionality as being linear, surface or volumetric structural elements when their extension in one or two spatial dimensions is considerably larger than in the other(s). Considering also the type of their load-bearing behavior, each group can be further subdivided. Linear elements, for example, can be classified as rods and columns with mainly axial loading or as beams with axial and transverse loading. Similarly, surface-like structural elements are

grouped as shells or plates with loads mainly applied in-plane or perpendicular to the element's plane, respectively.

structural type	refinement type	parameter	default value	comment
linear	degree	cross section min	2	1*)
		longitudinal min	2	1*)
	elements	cross section total min	1	2*)
		cross section size max	0	3*)
		longitudinal total min	4	2*)
		longitudinal size max	0	3*)
surface	degree	in-plane min	2	1*)
		out-of-plane min	2	1*)
	elements	in-plane total min	5	2*)
		in-plane size max	0	3*)
		out-of-plane total min	1	2*)
		out-of-plane size max	0	3*)
volumetric	degree	min	2	1*)
	elements	total min	4	2*)
		size max	0	3*)
undefined	degree	min	2	1*)
	elements	total min	5	2*)
		size max	0	3*)

Table 6.2: Refinement parameters for NURBS patches according to the type or class of structural elements they represent. Comments: 1*) minimum degree of the basis functions, 2*) minimum number of elements per patch dimension, 3*) maximum edge length of a single element. All refinement parameters are only valid for the direction they specify with their name. A value of zero denotes a parameter that shall not be considered in the refinement.

In contrast to the classic finite element method in which the finite elements constitute the highest level of geometric entities, the NURBS patch formulation in IGA is as advanced as to represent complete structural elements. This circumstance, which is already discussed with regard to BIM (cf. chapter 2), allows to associate each patch with its structural element type. Since the respective information is basically contained in the BIM data, the association can be easily established for all patches, provided the analysis model preserves the link with the BIM data and also presuming that the necessary information is correctly supplied in that data.

In case the simulation module is not aware of the underlying BIM model or that the model does not contain all of the required information, it is generally also possible to derive the spatial dimensionality from the pure geometric information provided with the patches. Therewith, one obtains an indication of the structural element type a patch represents. The definite determination, though, is not as simple in that case. For this purpose, the main load on each patch must be known prior to the analysis. This is fairly simple when loads are applied directly to the structural element, yet, it is not when loads are introduced through the link with other elements.

For typical civil engineering structures with the type of structure known to the software, it also seems feasible to determine the structural element type for all patches algorithmically, e.g.

from the orientation in space and the location in reference to other patches. As this constitutes a separate problem, it is not further pursued. Instead, this work restricts itself to distinguish the three spatial classes – linear, surface, and volumetric structural elements. The general concept is, however, readily extended to a more accurate differentiation of these elements.

For each of the three structural element classes (plus a fallback in case a patch is not unambiguously assigned to one of the spatial classes) specific refinement parameters are defined. For all of these parameters, which are listed in table 6.2, default values are supplied that can be modified by the structural engineer conducting the simulation. In the process of the analysis, the structural type of all patches contained in the model is evaluated. Then, the patch compliance with the specific refinement parameters of the respective structural type is individually tested. In case a patch does not meet the requirements, it is refined accordingly. If necessary, the degree is elevated first and only then additional elements are introduced to always retain the k-refinement strategy (cf. sect. 3.5.5).

The default parameters in table 6.2 are set by the author's experience to preserve a sufficiently coarse discretization for which the resulting linear system is solved at low computational cost, but which, at the same time, provide results with some significance. Thus, the analysis model can be extracted from the BIM data and directly analyzed in order to gain an idea of the structure's behavior in a defined load situation. In general, however, the provided default values are likely to be modified in order to obtain a finer discretization that leads to results with higher accuracy.

6.3.2 Refinement for contact

The geometrical type refinement strategy is complemented by a second strategy that refines the initial geometric model obtained from the BIM data on the basis of the contact state of the patches. In the vicinity of yet to create mortar couplings, additional knots are introduced to those patches involved in a coupling situation, meaning h-refinement is applied. This approach guarantees the discretization of the contact interfaces not to fall below a minimum refinement level, thereby ensuring the appropriate interpolation of the interface traction field. The refinement does, however, not only concern the interface tractions, but often, the areas of patch contact are located at the corners of structural elements where internal forces like bending moments and shear forces frequently reach a maximum. Increasing the refinement level in these areas contributes to a better quality of these maximal values.

The general procedure for contact refinement follows the one already outlined for structural element type refinement. Again, different categories of refinement parameters, each with a number of subentries, are defined and supplied with a default value. In the course of the analysis process, all contact interfaces are automatically evaluated. The master and slave patches associated with a detected coupling are refined until they meet the minimum refinement level established by specifying the values for the parameters in table 6.3.

The refinement parameters are divided into three classes. The *normal* class refers to the refinement of affected patches in the parametric direction that is normal to the contact interface. Since the patch extension in that direction can be small or large compared to its other extensions, different default values are provided for each parameter. When, for example, a beam is attached to a contact interface with the front face of its longitudinal axis, the *long* parameter

class	parameter	default values short	intermediate	long	comment
normal	count	0	1	2	1*)
	rel dist min	0.0	0.0	0.0	2*)
	rel dist max	0.15	0.10	0.05	2*)
	phys size max	0.0	0.0	0.0	3*)
	phys size min	0.05	0.15	0.25	3*)
adjacent	count	0	−1	−1	1*)
	rel dist min	0.01	0.01	0.01	2*)
	rel dist max	0.05	0.075	0.10	2*)
	phys size max	0.0	0.0	0.0	3*)
	phys size min	0.05	0.15	0.15	3*)
self	count	0	3	4	1*)
	phys size max	0.00	2.50	5.00	3*)
	phys size min	0.05	0.15	0.25	3*)

Table 6.3: Refinement parameters for NURBS patches associated with a mortar contact interface. The *normal* class refers to refinement normal to the contact interface, the *adjacent* class to the refinement in-plane but outside of the contact interface, and the *self* class to the refinement of the contact interface itself.
Comments:
1*) Minimum number of elements in a defined interval of the relevant parametric direction, where the interval is either defined by the contact interface or by other parameters.
2*) Minimum and maximum distance from the contact interface defining the beginning and the end of the interval that must contain the *count* number of elements. In the case of the *normal* class, the distance is relative to the patch length in normal direction and for the *adjacent* class it is relative to the contact interface length.
3*) The number of required elements is overruled by the minimum and maximum element edge length parameters. If the element edge length would exceed the *phys size max* value, the element *count* parameter is increased, if it falls below the *phys size min* value, the number is reduced.
The negative value for the *adjacent_count* parameters denotes that number of required elements is not explicitly specified but instead the degree of the basis functions in the respective direction is to be used.

is used. The *short* parameter is used instead, in case the contact interface is perpendicular to the beam somewhere along the longitudinal axis. The *intermediate* parameter is intended for patches that show a significant difference in their extension along parametric axes, which are – on the basis of their geometrical type – actually expected to be in a similar range. The two in-plane directions of a surface type structural element, for example, would both qualify for the *long* default parameter, as opposed to the out-of-plane direction, which would use the *short* parameter. Yet, in case the shorter in-plane extension is less than half the size of the longer one, the intermediate parameter is used instead of the *long* parameter. This distinction of the three default values for *long*, *intermediate*, and *short* geometric extensions applies equally to the two other parameter classes. The *adjacent* class refers to the patch refinement in the parametric directions that are in-plane with the contact interface but outside thereof. To specify the refinement of the patches in the contact interface itself, the *self* refinement parameter class is used. The meaning of the individual parameters is explained in table 6.3.

As in the case of the structural element type refinement, the default values specified in table 6.3 are defined with the aim of retaining a relatively coarse mesh that guarantees a fast solution of the associated linear system. Adjusting these parameters prior to the analysis is not as relevant as in the previous case. Yet, it is obvious that also for the contact refinement strategy, a finer discretization results in a higher result accuracy. With increasing refinement requirements due to the geometrical type refinement settings, the contact refinement loses its relevance. This is because the contact interfaces are automatically meshed in a fine manner when the entire patch is.

6.3.3 Remarks

For completeness it is noted that the structural type refinement and the contact refinement strategy are meant to supplement each other and thus they are actually two components of the same strategy.

In the light of the anisotropic refinement results given in sect. 6.2, it appears expedient to further distinguish the linear and surface type degree refinement parameters of the geometrical type refinement strategy according to the slenderness of the respective structural type and the different directions. As, however, it does not alter the general approach and with regard to the general complexity, this was dispensed with in table 6.2. Furthermore, a more practical implementation of the framework could consider the use of a more advanced element formulation for slender structural elements, e.g. the locking-free solid-shell element proposed by Bouclier et al. [33]. In the following example, shear locking, at least, is not a major issue.

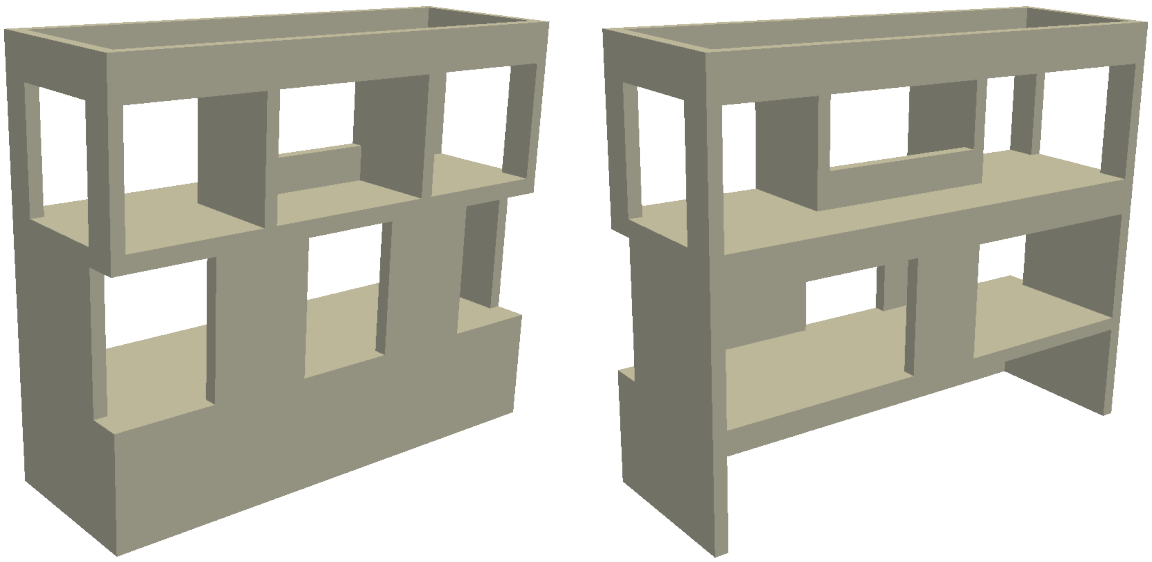
In general, it is conceivable to extend the list of structural element types. Also the properties leading to the classification of a NURBS patch as a specific type in that list are worth a discussion, as are the specified default values for the different refinement parameters. The primary intention of the outlined refinement approach is hence not the final definition of these parameters and their default values but rather a demonstration of what is achievable with this fairly simple strategy: The computation of relevant results can be completely automatized starting from a provided BIM model. At the same time, a structural engineer remains in full control of the refinement process by simply modifying a few relevant parameters.

Thanks to the weak coupling of the patches, an applied refinement does not propagate through the entire model but is limited to a single patch. Therewith, the structural engineer also retains the flexibility to locally modify the mesh in areas of special interest after the automated refinement procedure has created a discretization that is initially of the same quality for all patches – which would, however, suffice for most applications.

6.3.4 Example

The automated empirical refinement strategy is demonstrated on an example that can be interpreted as the structure of an academic building. Subsequently, it is referred to as multistory example. The geometry of the structure is depicted in fig. 6.7. In order to describe this geometry with NURBS, twenty-three volumetric patches are defined. In fig. 6.8, they are shown with their bounding edges at the coarsest possible refinement level – with linear basis functions and a single element per patch. In terms of an isogeometric analysis, these patches constitute

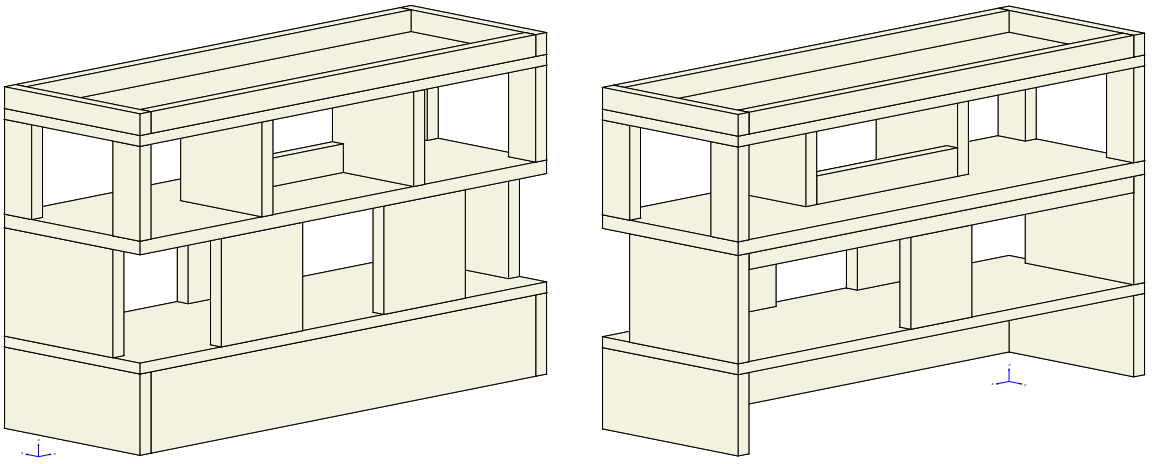
individual subdomains, which are to be coupled with the mortar method. As in previous examples, the coupling is done automatically by the software. The only geometry related input to the analysis framework is the definition of the NURBS patches. These definitions, which also provide the exact geometry for this example, can be found in appendix B. The global model dimensions are $15.0 \times 5.0 \times 12.6$.



(a) Front **(b)** Back

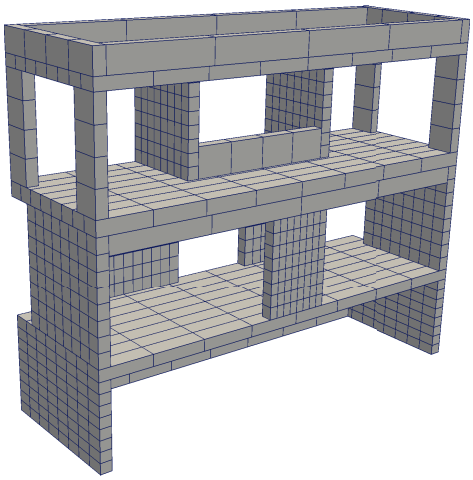
Figure 6.7: Geometry of the structure that is to be analyzed in this example.

The material of the structure is assumed to be linear-elastic and isotropic, defined by the parameters Young's modulus $E = 3.05 \cdot 10^{10}$, Poisson ratio $\nu = 0.25$, and density $\gamma = 2500$. Being subjected to a gravitational field $g = 9.81$ oriented in vertical direction, the structure is loaded by its self-weight applied as body load. All displacement DOFs at the structure's base are fixed, i.e. $u_x = u_y = u_z = 0$ at $z = 0$.

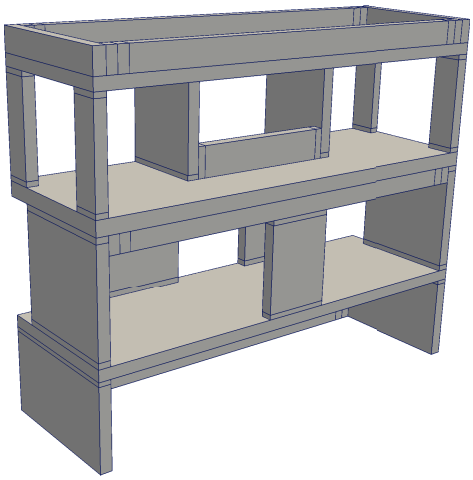


(a) Front **(b)** Back

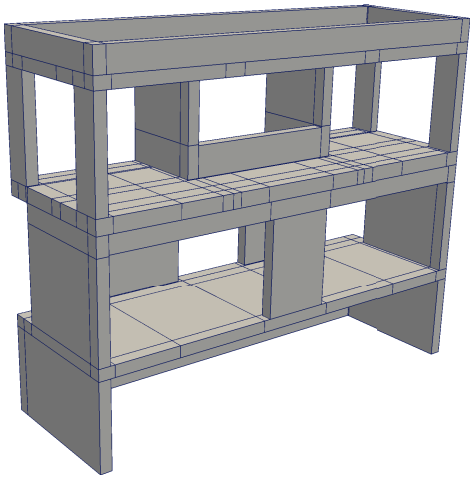
Figure 6.8: Visualization of the individual NURBS patches that are used to represent the structure depicted in fig. 6.7. Each NURBS patch is considered a subdomain in the isogeometric analysis with mortar couplings.



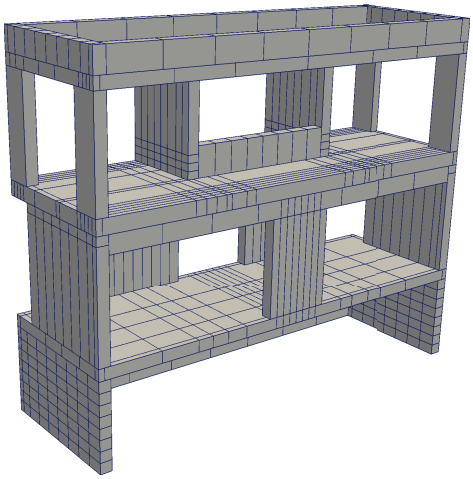
(a) Geometrical type refinement



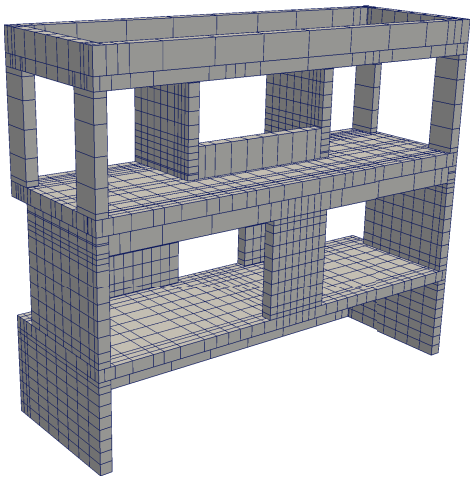
(b) Contact refinement – Normal



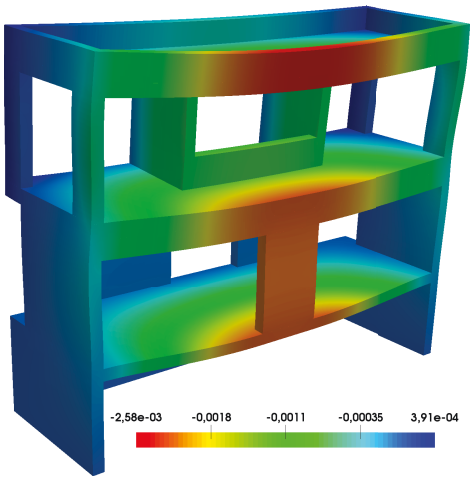
(c) Contact refinement – Adjacent



(d) Contact refinement – Self



(e) Combined refinement



(f) Displacement u_z - deformed shape

Figure 6.9: Inherent mesh of the NURBS patches when the two empirical refinement strategies are separately applied with their default values. The final combined mesh (e) and the therewith obtained displacement result results (f) correspond to model *nurbs 03*.

In order to demonstrate the effect of the automated empirical refinement strategy on the results obtained for this structure, it is analyzed multiple times. At the coarsest possible discretization, the analysis delivers nearly meaningless results. Therefore, the initial model is refined prior to any of the analysis runs. At first, the default settings supplied with tables 6.2 and 6.3 are used to refine the structure. In order to demonstrate the difference between geometrical type and contact refinement, these two strategies are separately applied for the first two analysis models. The resulting discretizations are depicted in figs. 6.9(a) to 6.9(d). In all subsequent analysis models, both strategies are applied together. Starting with model number 06, the default refinement parameters are adapted to increase the result quality obtained with the refined models.

A second model of identical geometry, material parameters, and boundary conditions is created with Ansys to provide a reference solution with the standard FEM. This model is naively refined by setting a global size parameter that determines the maximum edge length of any element in the discretized model.

In sum, 13 model variants are analyzed. The properties of the models *nurbs 01* to *nurbs 12* and the *ansys* model, as well as their differences, are explained below in detail. Results are provided afterwards and in table 6.4.

nurbs 01 The initial model is refined with the default settings for the geometrical type refinement as defined in table 6.2. Contact refinement is not applied. The resulting mesh is pictured in fig. 6.9(a). For the representation of the Lagrange multiplier field the bilinear Lagrangian interpolation is used. This model without contact refinement serves for demonstration purposes only.

nurbs 02 The initial model is refined with the default settings for contact refinement as defined in table 6.3. Geometrical type refinement is not applied but the degree of all patches is raised to $p = 2$. The resulting mesh is a combination of the meshes pictured in figs. 6.9(b) to 6.9(d). As before, the bilinear Lagrangian interpolation is used for the representation of the Lagrange multiplier field. This model serves for demonstration purposes only.

nurbs 03 The initial model is refined with the default settings for geometrical type and contact refinement, i.e. the mesh is a union of the meshes obtained for models *nurbs 01* and *nurbs 02*. The resulting mesh is pictured in figs. 6.9(e). Again, the bilinear Lagrangian interpolation is used for the representation of the Lagrange multiplier field. The refinement is applied in the following order:

p-refinement according to the geometrical type
h-refinement according to the geometrical type
h-refinement according to the contact evaluation

nurbs 04 The model parameters are identical to those of model *nurbs 03*, but the refinement order is modified to:

p-refinement according to the geometrical type
h-refinement according to the contact evaluation
h-refinement according to the geometrical type

nurbs 05 The model parameters are identical to model *nurbs 03*, but the interpolation of the Lagrange multiplier field is changed to NURBS interpolation.

All the following models base on model *nurbs 03*, i.e. they use the bilinear Lagrangian interpolation for the Lagrange multiplier field. Geometrical type as well as contact refinement is applied. The default refinement settings are adapted.

nurbs 06 The values of the following geometrical type refinement parameters are changed from 1 to 2, i.e.

$$\textit{linear–elements–cross\_section\_total\_min} = 2$$
$$\textit{surface–elements–out-of-plane\_total\_min} = 2$$

Thus, there are at least two elements in the through-thickness direction of surface-like patches and also at least two elements in any cross-sectional dimension of linearly oriented patches.

nurbs 07 In addition to the modifications of model *nurbs 06*, the size of the elements in the longitudinal directions of linearly oriented patches and in-plane direction of surface-like patches is limited to 0.5 by setting

$$\textit{linear–elements–longitudinal\_size\_max} = 0.5$$
$$\textit{surface–elements–in-plane\_size\_max} = 0.5$$

Contact interfaces are refined by setting

$$\textit{normal–count–intermediate} = 2$$
$$\textit{normal–phys\_size\_min–intermediate} = 0.05$$
$$\textit{normal–phys\_size\_min–long} = 0.05$$
$$\textit{adjacent–phys\_size\_min–intermediate} = 0.05$$
$$\textit{adjacent–phys\_size\_min–long} = 0.05$$

nurbs 08 Model *nurbs 07* is further refined by raising the degree of all patches in any direction to $p = 3$ via the following settings

$$\textit{linear–degree–cross\_section\_min} = 3$$
$$\textit{linear–degree–longitudinal\_min} = 3$$
$$\textit{surface–degree–in-plane\_min} = 3$$
$$\textit{surface–degree–out-of-plane\_min} = 3$$

nurbs 09 Again based on model *nurbs 07*, this model is further refined by reducing the element size in longitudinal directions via the settings

$$\textit{linear–elements–longitudinal\_size\_max} = 0.25$$
$$\textit{surface–elements–in-plane\_size\_max} = 0.25$$

nurbs 10 Based on model *nurbs 09*, the element size in the cross-sectional and out-of-plane directions is reduced by increasing the minimum element count in these directions with the settings

$$\textit{linear–elements–cross\_section\_total\_min} = 3$$
$$\textit{surface–elements–out-of-plane\_total\_min} = 3$$

nurbs 11 With the knowledge of the previous analysis results, this model is created on the basis of model *nurbs 09* and the selective elevation of the degree in the through-thickness and cross-sectional directions via the additional settings

linear–degree–cross_section_min = 3
surface–degree–out-of-plane_min = 3

nurbs 12 The final *nurbs* model is identical to *nurbs 11*, but instead of the bilinear Lagrangian interpolation, the NURBS interpolation is used for the Lagrange multiplier field.

ansys The reference solutions are obtained with Ansys Mechanical APDL, v16.0. The geometry is automatically meshed with *SOLID187* elements, which are 10-node tetrahedron-shaped elements with quadratic displacement interpolation behavior. In each reference solution the element edge length is globally limited to a specific value that ranges between 0.50 and 0.05 in the different analysis runs. The results from the finest mesh, i.e. those results obtained with a maximum element edge length $l_e = 0.05$, are used to normalize the results of the isogeometric analysis. Those are also the results given as absolute values in table 6.4.

Model	Unknowns			Normalized result values			
	Active DOFs	LMs	total	Strain energy	max $\|u_x\|$	max $\|u_y\|$	max $\|u_z\|$
nurbs 01	13,050	1,182	14,232	0.871	0.883	0.885	0.868
nurbs 02	24,390	1,782	26,172	0.872	0.814	0.886	0.864
nurbs 03	29,448	1,854	31,302	0.988	0.981	0.990	0.982
nurbs 04	25,308	1,782	27,090	0.954	0.944	0.957	0.951
nurbs 05	29,448	3,150	32,598	0.973	0.962	0.976	0.968
nurbs 06	40,728	2,862	43,590	0.986	0.973	0.989	0.982
nurbs 07	61,824	3,600	65,424	0.994	0.993	0.996	0.993
nurbs 08	103,770	3,744	107,514	0.999	1.000	1.001	0.999
nurbs 09	128,064	5,634	133,698	0.996	0.996	0.998	0.996
nurbs 10	203,976	7,974	211,950	0.993	0.991	0.993	0.992
nurbs 11	165,030	5,796	170,826	1.000	1.001	1.001	0.999
nurbs 12	165,030	9,915	174,945	0.987	0.985	0.990	0.988
				Absolute result values			
ansys	44,413,224	0	44,413,224	1.383×10^3	3.18×10^{-4}	2.49×10^{-3}	2.62×10^{-3}

Table 6.4: Selected analysis results obtained from the differently refined models.

Displacement and strain energy results of the various models at their respective discretization level are given in table 6.4. All IGA results are normalized with corresponding results from the standard FEM solution obtained at the highest refinement. For a better visualization, strain energy and maximum $|u_z|$ displacement results are additionally provided as charts in figs. 6.10 and 6.11. Furthermore, there is a colored isoplot for the u_z-displacements of model *nurbs 03* shown in its deformed state (factor 200×). The evaluation of these results allows to make several statements:

Comparing results from models *nurbs 01* and *nurbs 02* with those from *nurbs 03* shows that the geometrical type and contact refinement strategies complement each other very well at coarse discretization levels. The results from the combined application of the two strategies (model *nurbs 03*) are significantly better than those obtained with either strategy being applied by itself, while the overall computational cost is not dramatically increased. In general, a coarse discretization of a single NURBS patch with basis function that are at least of degrees $p = 2$ is in many cases sufficient to deliver acceptable results. Yet, when multiple patches are coupled, it must be granted that the coupling conditions

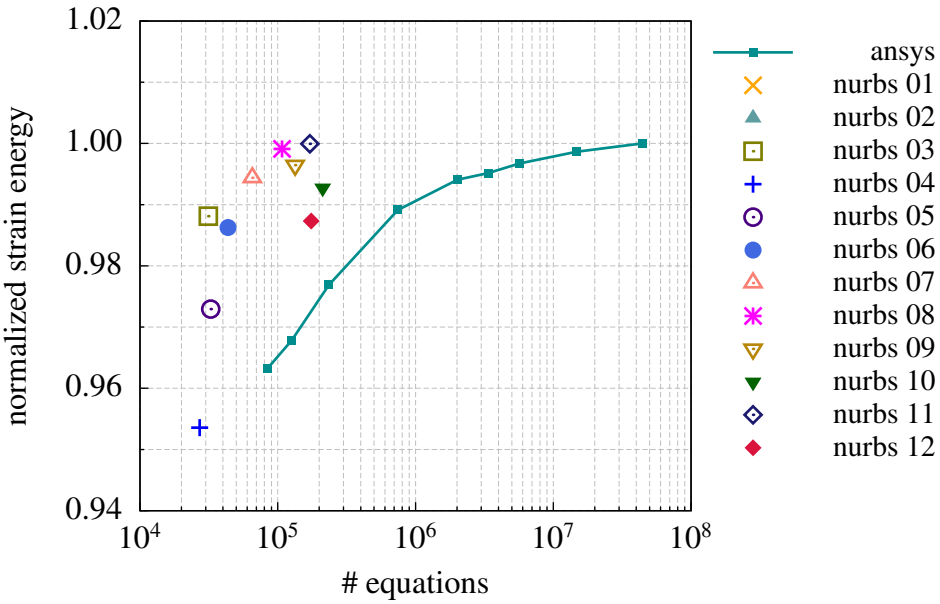


Figure 6.10: Visualization of the normalized strain energy results that are obtained from the isogeometric analysis model after applying contact and geometrical type refinement with different empirical settings. The IGA results are compared to those of standard FEM – denoted *ansys* – at different element edge lengths. The *nurbs 02* results are outside the range of this graph.

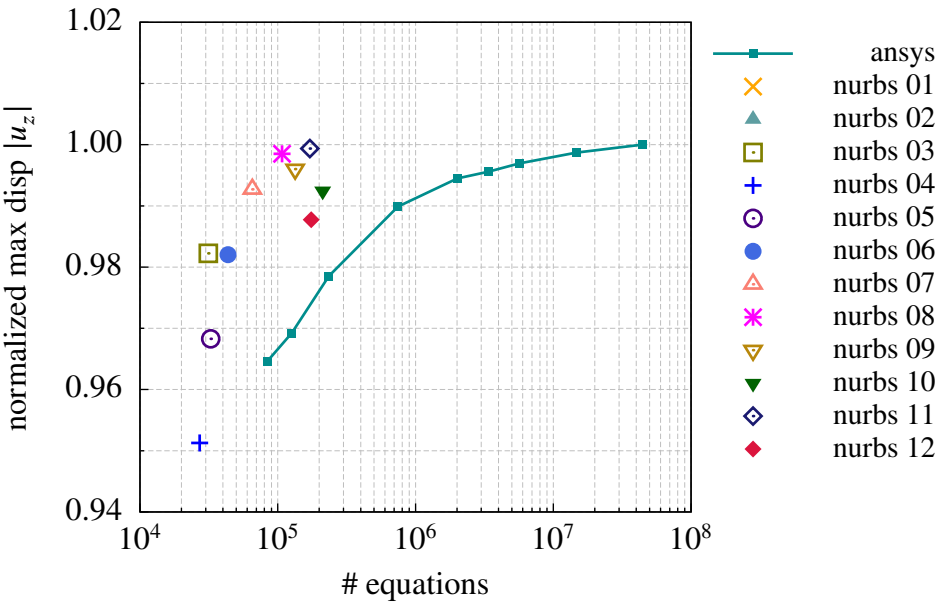


Figure 6.11: Visualization of the normalized maximum absolute displacement values in vertical direction $|u_z|$ that are obtained from the isogeometric analysis model after applying contact and geometrical type refinement with different empirical settings. The IGA results are compared to those of standard FEM – denoted *ansys* – at different element edge lengths. The *nurbs 01* and *nurbs 02* results are outside the range of this graph.

are adequately expressed. For that reason, the contact refinement strategy is required to retain the good interpolation behavior of coarsely discretized patches.

Comparing results from models *nurbs 03* and *nurbs 04* with each other indicates that the h-refinement according to the geometrical type settings should be applied prior to that of the contact refinement settings. When the geometrical type refinement requires only a minimum number of elements but not a maximum element size – which is in general not a good idea, but ensures a coarse discretization with default settings while the model is still unknown – the patches are likely to be refined in the vicinity of contact interfaces only. Then, the patch discretization away from the contact interfaces remains insufficient for results with acceptable accuracy.

Comparing the results from model *nurbs 03* with those from model *nurbs 05* and again those from *nurbs 11* with *nurbs 12* indicates a non-negligible difference in the results when either bilinear Lagrangian or NURBS interpolation is used for the Lagrange multiplier fields. At least in this example, the NURBS interpolation variant causes the model to be stiffer, which, in the direct comparison, shows by the lower strain energy and lower absolute displacement values. At the same time, the NURBS interpolation has a higher numerical cost due to the increased number of Lagrange multipliers. Hence, also in this more practical example, the bilinear Lagrangian interpolation appears to be superior to the NURBS interpolation.

In general, it is observed that results obtained with the default refinement settings (models *nurbs 03* and *nurbs 05*) are of sufficient quality for most structural engineering applications. Considering finer discretizations by modified refinement settings, the IGA models with empirical refinement outperform the standard FEM model in terms of achieved result accuracy related to the number of unknown solution variables. Here, it must be noted that the manual effort put into the refinement is negligible for IGA and the classic FEM with Ansys – which corresponds to a primary goal of this work: to allow for a fast evaluation of design alternatives – with a low amount of human labor. For the IGA model, these results can be obtained from the pure geometry description in a nearly fully automatic manner.

With models *nurbs 06* to *nurbs 10*, the discretization of the models is gradually improved, e.g. by limiting the maximum element extension to smaller values or by increasing the degree of the basis functions – possibly only for selected parametric dimensions. This leads to a gradual convergence of the strain energy and displacement results towards the reference solution, cf. fig. 6.11.

Associating the individual patches with their geometrical type – that basically also represent a structural type – allows for a selective refinement that is not possible with solid modeling in standard FEM. This becomes clear on the evolution of models *nurbs 07* to *nurbs 11*:

> Moving from model *nurbs 07* to model *nurbs 08*, the degree of the basis function is increased from $p = 2$ to $p = 3$ in all parametric directions. This leads to a significant improvement of the strain energy result – at the cost of solving a matrix with a significantly increased bandwidth (cf. table 5.1 in sect. 5.4).
>
> Moving again from model *nurbs 07*, now to model *nurbs 09*, instead of modifying the degree of the basis functions, the maximum element size for the in-plane and

longitudinal directions only is reduced from 0.50 to 0.25. As can be seen from fig. 6.10, this does not have a comparable positive effect on the strain energy result as the modifications from model *nurbs 07* to *nurbs 08* do.

Moving on from model *nurbs 09* to *nurbs 10*, now also the number of elements in the out-of-plane and cross-sectional patch directions is increased, from a minimum number of two elements to at least three per parametric direction. Despite the numerical cost emanating from nearly doubling the number of equations that are to be solved, the refined model does not lead to a further improvement of the strain energy and displacement results.

In order to reduce the computational cost evolved from model *nurbs 10*, the element count in the out-of-plane and cross-sectional patch directions of model *nurbs 11* is chosen as in model *nurbs 09*. A compensation with regard to result accuracy is realized by selectively increasing the degree of the basis functions in the respective directions, which leads to an anisotropic element formulation with regard to the degree of the basis functions. In the outcome, the results are close to the optimum. The matrix bandwidth, however, did not expand as dramatically as in model *nurbs 08* and the number of equations is significantly reduced when compared to model *nurbs 10*.

6.4 Adaptive refinement

6.4.1 Preliminaries

The introduction to this chapter named a general drawback of adaptive refinement in practical engineering work: it is expensive in terms of the overall numerical cost. The final discretized model, really just one of its instances – a model can solely be adapted for a specific load situation – is only obtained after an iterative process of repeatedly applied selective refinements, where each iteration must be finalized with a new solver run on the current model. Considering the example previously given in sect. 6.3.4, it is obvious that some initial refinement should be applied before pursuing the adaptive strategy, simply to not start from an unreasonably coarse discretization. Hence, the empirical refinement for isogeometric models outlined in sect. 6.3 remains relevant also for the adaptive strategy.

The initial motivation to exploit the adaptive strategy despite its cost based on the specific properties of the isogeometric models. Individual patches are capable of representing a complete structural entity or at least one of its principal parts. In general, it can be assumed that the parametric coordinate axes are aligned with major axes of the respective structural entity. Since the refinement operations knot insertion and degree elevation are tied to the parametric axes, they are also tied to the major axes of the associated structural entity, which in consequence can be refined according to their directional properties. The impact of anisotropic refinement was previously investigated on the solid beam example. This example shows that direction specific refinement does not necessarily perform better than the unspecific isotropic approach. A refinement of the “wrong” direction, however, may be detrimental to the numerical performance. Yet, the decision which direction is best refined is not obvious.

Considering h-refinement, it is always possible to subdivide an identified hexahedral element into eight smaller elements by splitting it along the normal planes of its three parametric axes. Alternatively, it can be divided only once in the direction of its largest extension. Obviously, the second approach generates fewer new DOFs, yet, it is somewhat arbitrary, as the choice is neither related to the properties of the associated structural entity nor to the actual quality of the solution field in that direction. Regarding selective p-refinement, there is no simple indication of the direction at all – assuming the degrees of basis functions in all parametric directions are identical at first. Not only is there no simple indication for the direction of either refinement operation, but in contrast to standard FEM, where these operations are limited to a single or, by choice, to a group of elements, the decision has a relevant impact on the global model size in NURBS based IGA. Though the mortar coupling approach prevents the propagation of degree elevations and knot insertions through an entire model, these operations remain patch global and thus, can severely increase the number of unknowns.

The standard procedure used to manage the evolution of a finite element mesh in an adaptive refinement approach is the application of an error estimator. The general objective of error estimators is to provide an estimate of the true, yet unknown, discretization error. The estimate should be cheap to compute but as accurate as possible at the same time. Since it is an estimate only, guaranteed upper and lower bounds are desirable to assess the quality of numerical results obtained for a specific discretization. Regardless of the fact, that these objectives are barely achieved in practically relevant simulations,[2] these requirements can be significantly eased, when the main goal is to steer an adaptive refinement process. Though the availability of the full error information is desirable, only a qualitative assessment of the error distribution is really required for this purpose. The associated computational work, however, should remain low, as otherwise, a finer mesh could be used in the first place.

6.4.2 Error estimates

A general overview on frequently used error estimators in the context of standard finite element analysis is, for example, provided by Ainsworth and Oden [3], Verfürth [160], or Grätsch and Bathe [73]. Though the literature on a posteriori error estimation is vast in general, only few publications exist that deal with that specific topic in IGA. Some of them are [53, 96, 165], yet none is known to deal with the Zienkiewicz-Zhu type error estimator that, as of its simplicity, is implemented in many standard FEM codes [51] and consequently, is also particularly popular among engineers [3]. An adapted variant thereof that is shown to work for NURBS based IGA is presented subsequently.

In the course of developing the basic equations of the finite element method, the true displacement field $\boldsymbol{u}$, see eq. (3.1), is approximated with $\boldsymbol{u}^h$, as defined by eq. (3.32) for standard FEM and by eq. (3.99) for the corresponding isogeometric formulation. The error associated with the approximation is expressed as

$$\boldsymbol{e}^h(\boldsymbol{P}) = \boldsymbol{u} - \boldsymbol{u}^h \qquad \forall \boldsymbol{P} \in \Omega \tag{6.1}$$

The basis of the finite element formulation is formed by the virtual work principle defined in eq. (3.19). In that equation, the displacement field appears only in the term associated with the

[2] See the conclusions of Grätsch and Bathe [73] after reviewing many of the a posteriori error estimator concepts from the view of practical FEA.

internal virtual work. Thus, the error in the description of the displacement field leads to an error in the energy that is considered to be stored in the deformed body. The energy norm of the error, the square of which is defined by

$$\left\| e^h\left(\Omega^h\right)\right\|_E^2 = \int_\Omega \boldsymbol{\epsilon}(\boldsymbol{e}^h) : \boldsymbol{\sigma}(\boldsymbol{e}^h)\,\mathrm{d}\Omega \tag{6.2}$$

is regarded an appropriate norm for the error representation, as the true displacement field $\boldsymbol{u}$ minimizes the total potential energy of the structure and the approximate solution $\boldsymbol{u}^h$ minimizes it within the limited function space $\mathscr{U}^h$.

The error in the energy norm can be expressed as the difference of the internal virtual work statements once using the continuous and once the discrete displacement fields, leading to

$$\begin{aligned}\left\| e^h\left(\Omega^h\right)\right\|_E^2 &= \int_\Omega \boldsymbol{\epsilon}(\delta\boldsymbol{u}) : \boldsymbol{\sigma}(\boldsymbol{u})\,\mathrm{d}\Omega - \int_\Omega \boldsymbol{\epsilon}(\delta\boldsymbol{u}) : \boldsymbol{\sigma}(\boldsymbol{u}^h)\,\mathrm{d}\Omega \\ &= \int_\Omega \delta\boldsymbol{u}\cdot\bar{\boldsymbol{b}}\,\mathrm{d}\Omega + \int_{\Gamma_t} \delta\boldsymbol{u}\cdot\bar{\boldsymbol{t}}\,\mathrm{d}\Gamma_t - \int_\Omega \boldsymbol{\epsilon}(\delta\boldsymbol{u}) : \boldsymbol{\sigma}(\boldsymbol{u}^h)\,\mathrm{d}\Omega\end{aligned} \tag{6.3}$$

where in the second line, the internal virtual work statement of the continuous displacement field is replaced by the corresponding external virtual work, leaving the virtual displacements as the only unknown field variable. It is to note that the virtual displacements $\delta\boldsymbol{u}$ in eq. (6.3) are not subject to eq. (3.44), i.e. they are not restricted to the finite dimensional space $\mathscr{V}^h$.

Equation (6.3) constitutes the starting point for many residual-based error estimators that build upon the fundamental work of Babuška and Rheinboldt [14, 15]. Integration by parts applied to the remaining internal virtual work expression and a rearrangement of the resulting terms, as for example shown by Ainsworth and Oden [3], transforms that equation into the following element-based expression for the error in the energy norm

$$\left\| e^h\left(\Omega^h\right)\right\|_E = \sum_{\Omega_e}\int_{\Omega_e} Res_{\Omega_e}\left(\boldsymbol{u}^h\right)\delta\boldsymbol{u}\,\mathrm{d}\Omega_e + \sum_{\Gamma_e}\int_{\Gamma_e} Res_{\Gamma_e}\left(\boldsymbol{u}^h\right)\delta\boldsymbol{u}\,\mathrm{d}\Gamma_e \tag{6.4}$$

with

$$Res_{\Omega_e}\left(\boldsymbol{u}^h\right) = \nabla\cdot\boldsymbol{\sigma}\left(\boldsymbol{u}^h\right) + \bar{\boldsymbol{b}} \tag{6.5}$$

$$Res_{\Gamma_e}\left(\boldsymbol{u}^h\right) = \begin{cases} \frac{1}{2}\left(\boldsymbol{n}\cdot\boldsymbol{\sigma}\left(\boldsymbol{u}^h\right) + \boldsymbol{n}'\cdot\boldsymbol{\sigma}'\left(\boldsymbol{u}^h\right)\right) & \text{if } \Gamma_e \not\subset \Gamma \\ \bar{\boldsymbol{t}} - \boldsymbol{n}\cdot\boldsymbol{\sigma}\left(\boldsymbol{u}^h\right) & \text{if } \Gamma_e \subset \Gamma_t \\ 0 & \text{if } \Gamma_e \subset \Gamma_u \end{cases} \tag{6.6}$$

where the prime symbols in $\boldsymbol{n}'$ and $\boldsymbol{\sigma}'$ denote the normal and stress values from neighboring elements with a shared element boundary.

The residual terms in eqs. (6.5) and (6.6) are essential parts of both, implicit and explicit residual-based error estimators. According to Zienkiewicz et al. [174, sec. 13.7], the main error contribution stems from the gradient discontinuity across inter-element boundaries. As however, a fundamental feature of IGA is the C^{p-1}-continuity of the basis functions across element

boundaries,[3] the discontinuity vanishes for all basis functions of degree $p \geq 2$, a condition that can be presumed for most isogeometric analyses. Hence, the result of the first term in eq. (6.6) evaluates to zero in all standard cases. Furthermore considering the architecture and structural engineering context of this work, the prevailing body load is the specific weight, which in many cases can also be assumed constant on the element level and hence, linear stress fields suffice to express the equilibrium associated with eq. (6.5). In consequence, also error contributions from eq. (6.5) tend be to zero for $p \geq 2$, leaving only the mortar interfaces and the Neumann boundary as main sources of the error estimate. Obviously, the additional information provided by such an error estimator is limited, mortar interfaces and Neumann boundaries can be refined a priori – without the application of an error estimator. Explicit error estimators of the Babuška and Rheinboldt type are thus discarded, despite their computational inexpensiveness.

Instead, this work follows the idea of postprocessing the gradient of the displacement solution in order to improve its quality. Presuming it is possible to construct improved strain and stress fields that are "closer" to the true fields than their non-postprocessed counterparts, the error in the energy norm as defined by eq. (6.2) can be approximated with

$$\left\| e^h\left(\Omega^h\right)\right\|_E^2 \approx \left\| \check{e}^h\left(\Omega^h\right)\right\|_E^2 = \int_\Omega \left(\underline{\boldsymbol{\epsilon}}^* - \underline{\boldsymbol{\epsilon}}^h\right)^T \left(\underline{\boldsymbol{\sigma}}^* - \underline{\boldsymbol{\sigma}}^h\right) \mathrm{d}\Omega \tag{6.7}$$

where the star* symbol denotes improved quantities. This type of error estimator is often referred to as recovery error estimator or, after the main contributors to this variant of error estimation, Zienkiewicz and Zhu [171] type[4] error estimator. In standard FEM, the displacement solution is processed with eqs. (3.41) and (3.42) to evaluate the secondary analysis results, the strain and stress fields. Yet, these gradient fields are discontinuous at element boundaries – a rather unphysical limitation inherent to the method. A natural approach of constructing improved fields is therefore one that leads to continuous gradients across element boundaries. One option to obtain such continuous fields is to express the strains and stresses as

$$\boldsymbol{\epsilon}^*(\boldsymbol{P}) = \sum_{a=1}^{n_{en}} N^{(a)}(\boldsymbol{P})\ \hat{\boldsymbol{\epsilon}}^{(a)} \qquad \boldsymbol{\sigma}^*(\boldsymbol{P}) = \sum_{a=1}^{n_{en}} N^{(a)}(\boldsymbol{P})\ \hat{\boldsymbol{\sigma}}^{(a)} \qquad \forall \boldsymbol{P} \in \Omega_e \tag{6.8}$$

which is an equivalent formulation to the displacement interpolation defined in eq. (3.32). Expression (6.8) requires the availability of nodal strain and stress values. In standard FEM, it is generally possible to evaluate these nodal values directly, however, they are commonly recovered from element interior points that are known to deliver results of higher accuracy for these gradient fields.

For one-dimensional elements with polynomial interpolation functions of order p, the displacement solution is known to converge at a rate of order $\mathcal{O}(h^{p+1})$. The associated strain and stress fields are built from the displacement derivatives. Hence, they are expected to show a convergence rate of order $\mathcal{O}(h^p)$. Strains and stresses evaluated at the Gauss points,[5] however,

[3] Presuming that affected elements are part of the same patch and that there are no repeated knots in the associated knot vectors, which would reduce the continuity at the element boundary.

[4] also known as ZZ or Z^2 error estimator

[5] In detail, those Gauss points that for a given element formulation constitute the minimum number of points to deliver sufficiently accurate integration results and which retain the element's theoretical convergence behavior, i.e., for general quadrilateral elements, those points used for reduced integration with the integration order being equal to the polynomial degree of the displacement interpolation functions.

converge at least with order $\mathcal{O}(h^{p+1})$ and thus exceed the expected convergence rate by at least one order [16, 175]. Accordingly, these points are often referred to as superconvergent. Various techniques have been proposed that utilize the results evaluated at the superconvergent points to recover nodal strain and stress values, which, in conjunction with eq. (6.8), lead to strain and stress fields of assumingly improved accuracy. The most prominent of these methods is the superconvergent patch recovery technique by Zienkiewicz and Zhu [172, 173].

Presupposing the absence of repeated knots in the interior of the knot vectors and $p \geq 2$, the first derivatives of the basis functions in NURBS based IGA are patchwise continuous across element boundaries (cf. sect. 3.5.4). Hence, strain and stress fields are continuous across element boundaries as well, a desired feature of the method; which, however, limits the potential of improving the respective fields for the purpose of error estimation on the basis of eq. (6.7). Enhancing these fields with the help of superconvergent point results remains a viable option, nonetheless. The existence of superconvergent points also for first derivatives of the B-spline basis was recently shown by Anitescu et al. [8] in the context of isogeometric collocation methods. They computed the respective coordinates in parent element space for a non-open but uniform knot vector on the reference interval $[-1, 1]$. The results for basis functions with degrees $p = 1$ to 7 are reproduced in table 6.5.

degree	coordinate r
$p = 1$	0
$p = 2$	$\pm 1/\sqrt{3}$
$p = 3$	0 ± 1
$p = 4$	$\pm\sqrt{225 - 30\sqrt{30}}/15$
$p = 5$	0 ± 1
$p = 6$	± 0.5049185675126533
$p = 7$	0 ± 1

Table 6.5: Superconvergent points for the first derivatives of the B-spline basis on the interval $[-1, 1]$ according to Anitescu et al. [8].

The superconvergent property of the coordinates in table 6.5 does strictly speaking only apply to the polynomial B-spline basis defined in eq. (3.76) when evaluated with uniform knot vectors (cf. fig. 3.5). The NURBS basis used in this analysis framework, which is defined in eq. (3.83), coincides with that B-spline basis only in the case of uniform weights and only on patch interior elements. The basis on the p elements on the patch boundary is influenced by the repeated knots at the beginning and the end of the utilized open knot vectors. The same would apply for possibly repeated knots on the patch interior. The sampling points defined by the tensor product of the parent element coordinates in table 6.5 do in consequence not guarantee superconvergence of strain and stress results for all finite elements within the NURBS patches that may be present in this analysis framework. Nonetheless, they do for many of them and for the rest, they are assumed to be an approximation of the superconvergent points.

Allegedly improved strain and stress fields in the notation of NURBS based IGA are, in equiv-

alence to the respective formulation of standard FEM (6.8), expressed as

$$\boldsymbol{\epsilon}^*(\xi,\eta,\zeta) = \sum_{\hat{a}=1}^{n_{en}} \boldsymbol{R}_{p,q,r}^{f(\hat{a},e)} \boldsymbol{E}^{f(\hat{a},e)} \qquad \boldsymbol{\sigma}^*(\xi,\eta,\zeta) = \sum_{\hat{a}=1}^{n_{en}} \boldsymbol{R}_{p,q,r}^{f(\hat{a},e)} \boldsymbol{T}^{f(\hat{a},e)} \qquad \forall(\xi,\eta,\zeta) \in \hat{\Omega}_e^{(i)} \tag{6.9}$$

Here, $\boldsymbol{E}^{f(\hat{a},e)}$ and $\boldsymbol{T}^{f(\hat{a},e)}$ denote the control variables of strains and stresses, respectively. As already noted for the displacements, these control variables do not allow for a direct physical interpretation. They are not fully interpolated by the basis and thus, are technically not a part of the discretized domain Ω^h. Consequently, it is not possible, to evaluate the strain and stress control variables directly. Instead, they are patchwise recovered from the respective superconvergent point results using the analysis result processing method outlined in sect. 5.5.

With the availability of these improved fields and the following reformulation of eq. (6.7) to allow for an elementwise estimation of the error as

$$\left\| \check{e}^h \left(\Omega^h \right) \right\|_E^2 = \sum_{\Omega^{(i)} \in \Omega^h} \sum_{\Omega_e^{(i)} \in \Omega^{(i)}} \left\| \check{e}^h \left(\Omega_e^{(i)} \right) \right\|_E^2 \tag{6.10}$$

with

$$\left\| \check{e}^h \left(\Omega_e^{(i)} \right) \right\|_E^2 = \int_{\Omega_e^{(i)}} \left(\underline{\boldsymbol{\epsilon}}^* - \underline{\boldsymbol{\epsilon}}^h \right)^T \left(\underline{\boldsymbol{\sigma}}^* - \underline{\boldsymbol{\sigma}}^h \right) \, \mathrm{d}\Omega_e^{(i)} \tag{6.11}$$

it is possible to steer an adaptive h-refinement process. The performance of the estimator in that process is demonstrated on the cantilever plate example in the following section.

With eq. (6.11), one obtains a scalar error value for each element that, however, does not provide any information on a preferred refinement direction. A straight-forward approach to obtain that information nonetheless is to split the element energy error into the error contributions associated with the parametric directions. Generally, the parametric coordinates form a curvelinear system when projected into physical space. Since eq. (6.11) is evaluated by numerical integration, the strains and stresses are obtained for the local basis of the curvelinear system at a given integration point. Corresponding integration point values are then weighted and summed over the element. Defining the vector $\boldsymbol{\varphi}$ as

$$\boldsymbol{\varphi} = \begin{bmatrix} \epsilon^*_{\xi\xi} & - & \epsilon^h_{\xi\xi} \\ \epsilon^*_{\eta\eta} & - & \epsilon^h_{\eta\eta} \\ \epsilon^*_{\zeta\zeta} & - & \epsilon^h_{\zeta\zeta} \\ \gamma^*_{\eta\zeta} & - & \gamma^h_{\eta\zeta} \\ \gamma^*_{\xi\zeta} & - & \gamma^h_{\xi\zeta} \\ \gamma^*_{\xi\eta} & - & \gamma^h_{\xi\eta} \end{bmatrix} \circ \begin{bmatrix} \sigma^*_{\xi\xi} & - & \sigma^h_{\xi\xi} \\ \sigma^*_{\eta\eta} & - & \sigma^h_{\eta\eta} \\ \sigma^*_{\zeta\zeta} & - & \sigma^h_{\zeta\zeta} \\ \sigma^*_{\eta\zeta} & - & \sigma^h_{\eta\zeta} \\ \sigma^*_{\xi\zeta} & - & \sigma^h_{\xi\zeta} \\ \sigma^*_{\xi\eta} & - & \sigma^h_{\xi\eta} \end{bmatrix} \tag{6.12}$$

allows to rewrite eq. (6.11) as

$$\left\| \check{e}^h \left(\Omega_e^{(i)} \right) \right\|_E^2 = \int_{\Omega_e^{(i)}} \| \boldsymbol{\varphi} \|_1 \, \mathrm{d}\Omega_e^{(i)} \tag{6.13}$$

but instead of integrating the ℓ_1-norm of φ, its contents are integrated individually as

$$\boldsymbol{\Phi}_e^{(i)} = \int_{\Omega_e^{(i)}} \varphi \, \mathrm{d}\Omega_e^{(i)} \tag{6.14}$$

finally providing the element energy error as

$$\left\| \check{e}^h \left(\Omega_e^{(i)} \right) \right\|_E^2 = \left\| \boldsymbol{\Phi}_e^{(i)} \right\|_1 \tag{6.15}$$

As the contents of $\boldsymbol{\Phi}_e^{(i)}$ are related to the parametric directions of the associated finite element, it can be used to indicate the direction of refinement. That direction is defined by the index of the coefficient in $\boldsymbol{\Phi}_e^{(i)}$ that has the maximum value. The first three indices are related to the parametric directions ξ, η, and ζ respectively. In case one of the shear error values, i.e. indices 4, 5, or 6, has the maximum value, the element is split in both associated parametric directions. The direction specific refinement is discussed in sect. 6.4.3.2.

6.4.3 Examples

6.4.3.1 Cantilever plate example

Owing to the patch-global impact of h-refinement in NURBS based IGA, the efficiency of adaptive refinement is limited in some cases. It is, however, not futile in all situations. An example with relevant mesh optimization potential is presented in this section. A square plate is clamped on the right side and loaded by a constant traction along its top edge. The situation along with the relevant parameters used in the analysis of this two-dimensional example is pictured in fig. 6.12. With linear-elastic material behavior, stress singularities occur at the top and bottom corners of the clamped edge. To achieve a fast convergence of the strain energy stored in the plate, the mesh should be concentrated around the singularities.

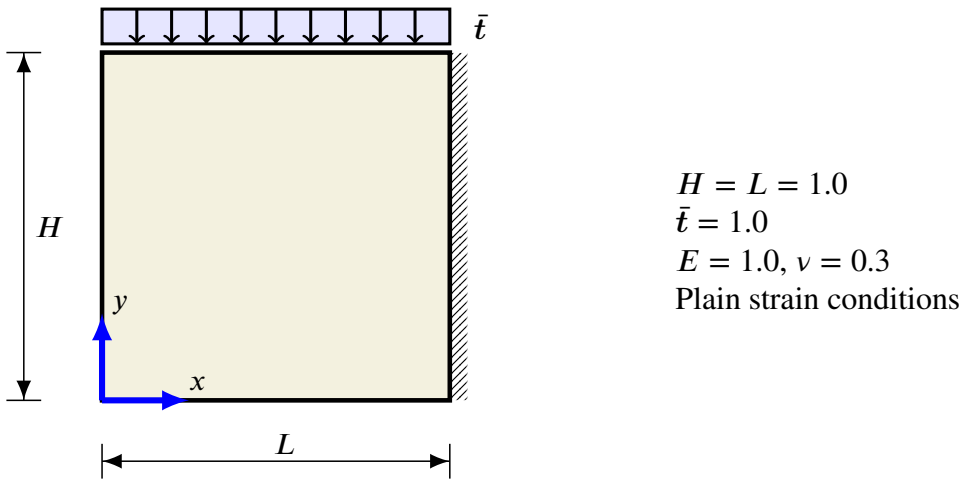
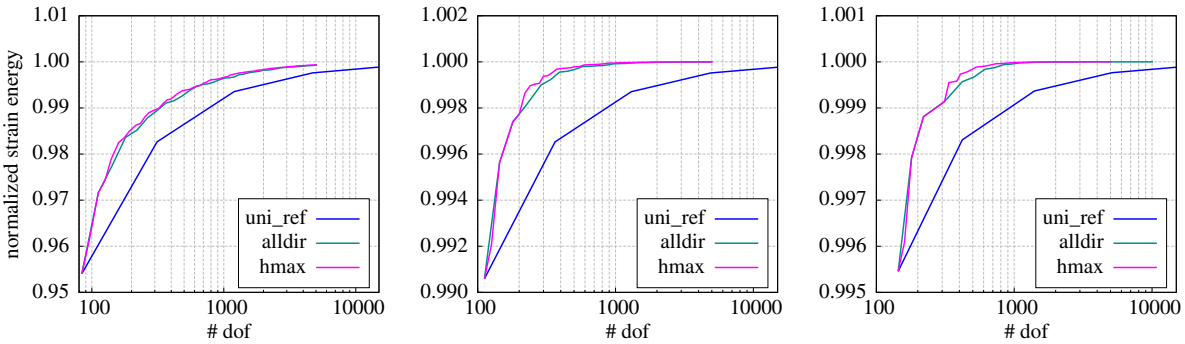


Figure 6.12: Cantilever plate

The performance of the Zienkiewicz-Zhu recovery error estimator for this problem and standard FEM was investigated by Ainsworth et al. [4]. Using the same parameters, the efficiency

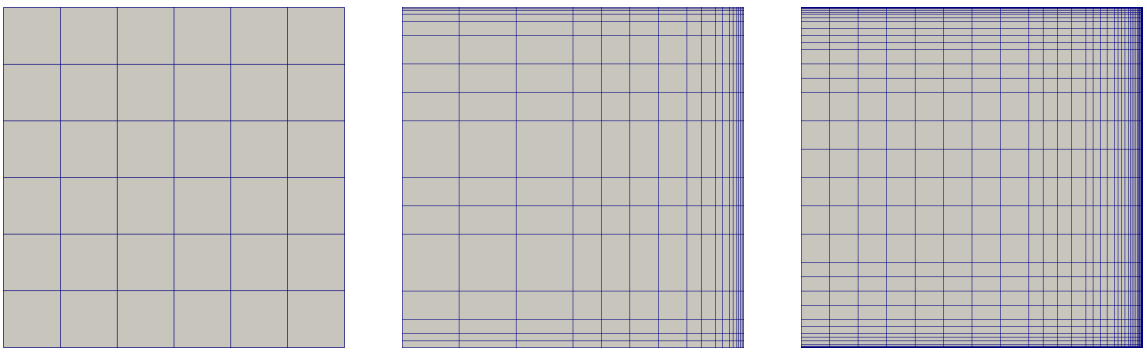
of the estimator based on the adapted recovery process is subsequently also shown for the isogeometric problem.

The plate is analyzed with linear, quadratic and cubic basis functions, the initial mesh for all variants is shown in fig. 6.14(a). Stresses (and strains) are evaluated at the superconvergent points of the respective degree of the basis functions (cf. table 6.5). For the quadratic variant, additional points are introduced by extrapolating the existing grid of stress sampling points to the patch boundary. In the cubic case, the superconvergent points are located on the patch boundaries anyway and in the linear case, the extrapolation was not found to improve the accuracy of the recovered fields. The grid of evaluated stress vectors is then used to recover the stress fields by the method described in sect. 5.5, which then forms the basis of the element-wise error evaluation with eq. (6.11). It is to note that in the linear case, where only a single superconvergent point exists per element, the recovery process reduces to averaging the values of adjacent points.



(a) Linear **(b)** Quadratic **(c)** Cubic

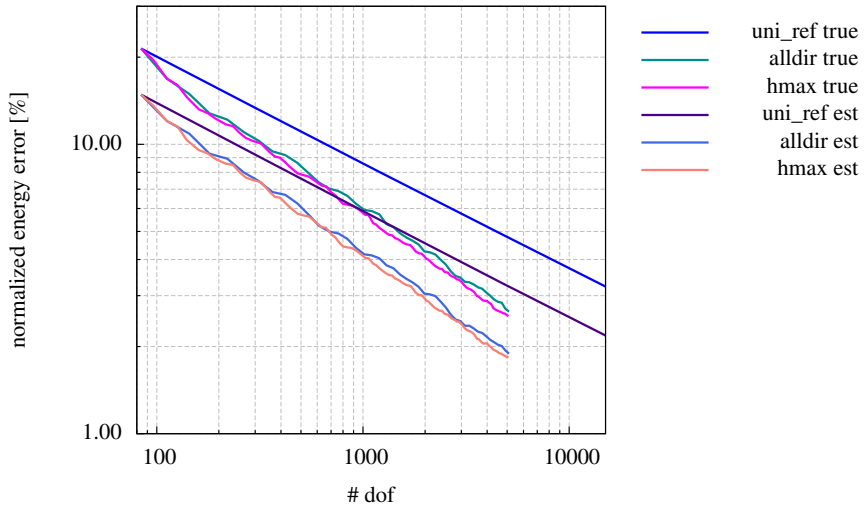
Figure 6.13: Normalized strain energy results of the cantilever plate example, sorted by the degree of the basis functions. The abbreviations in the key refer to uniform refinement (*uni_ref*), concurrent refinement in all parametric directions (*alldir*), and refinement in the parametric direction with the maximum element extension (*hmax*). The reference strain energy $U = 0.951838$ is obtained from a highly refined uniform mesh NURBS analysis with cubic basis functions.



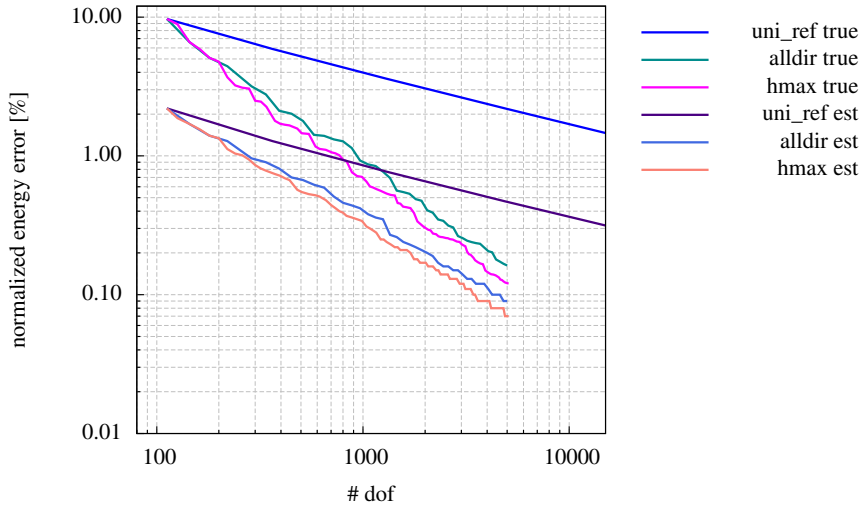
(a) Initial **(b)** Quadratic alldir step 10 **(c)** Quadratic alldir step 30

Figure 6.14: Meshes obtained during the adapted refinement process of the cantilever plate example.

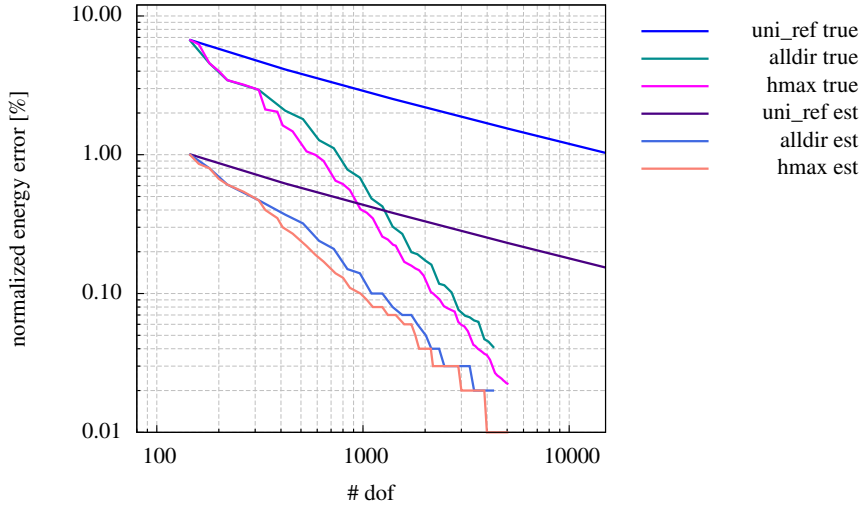
The element error values steer the adaptive process. In each iteration, the element with the largest error is refined – either in the direction of its largest extension or alternatively in all



(a) Linear



(b) Quadratic



(c) Cubic

Figure 6.15: True and approximate relative error of the iteratively refined cantilever plate.

parametric directions. Furthermore, the size of the p adjacent elements in the refinement direction is checked. In case the size of any of these elements is larger than 3× the size of the refined element, the respective element is also refined. The normalized strain energy results obtained with this procedure are plotted in the graphs of fig. 6.13. This figure demonstrates the performance advantage of adaptive refinement over uniform refinement with the proposed error recovery method. Independent of the degree of the basis functions does the strain energy converge significantly faster with adapted meshes than it does with uniform refinement. Yet, there is no relevant advantage for either the refinement in one or in all parametric directions. Typical meshes obtained during the adaptive process are shown in fig. 6.14.

In this example the iterative process is stopped, when the total number of DOFs reaches the limit of $5,000$. In a practical situation, it is desirable to specify a stopping criteria that is related to the error. Such a criteria is the relative error defined as

$$e_{rel} = \frac{\left\| \boldsymbol{e}^h \left(\Omega^h \right) \right\|_E}{\left\| \boldsymbol{u} \left(\Omega \right) \right\|_E} \tag{6.16}$$

Of course, the relative error is not computable without the knowledge of the true solution. Therefore, it is approximated by

$$e_{rel} \approx \check{e}_{rel} = \frac{\left\| \check{\boldsymbol{e}}^h \left(\Omega^h \right) \right\|_E}{\sqrt{\left\| \boldsymbol{u}^h \left(\Omega^h \right) \right\|_E^2 + \left\| \check{\boldsymbol{e}}^h \left(\Omega^h \right) \right\|_E^2}} \tag{6.17}$$

For this example, the approximate relative error and the true relative error are compared in fig. 6.15. In all cases, the approximated error underestimates the true error, which is not surprising, as the source of the estimation is the improved gradient field that cannot be expected to have the accuracy of the true gradient field. Yet, it is noted that the underestimation is higher in the quadratic and cubic cases. For the linear interpolation, the recovery process averages the discontinuous gradients across the element boundaries, which has a positive effect on the improved field that is not present in the higher degree examples. Nonetheless, the estimated error monotonically declines with a growing number of DOFs and also converges to the true error for all analyzed variants. Both are necessary conditions to specify a limit of the relative error at which the refinement is to stop.

6.4.3.2 Direction specific refinement

Surface under tension example

The simple 2D problem depicted in fig. 6.16 is examined under the aspect of direction specific refinement. A square plate with statically determined support at its top edge is loaded by a constant line load in horizontal direction and a constant body load in vertical direction. The relevant parameters are also listed in fig. 6.16.

Assuming isotropic, linear-elastic material behavior and using a linear NURBS element formulation, the stresses for the given problem setting are computed as shown in fig. 6.17. Owing to the characteristics of the applied loads, the stress field in horizontal direction is nearly constant whereas in vertical direction, it varies linearly with the height. The load parameters were

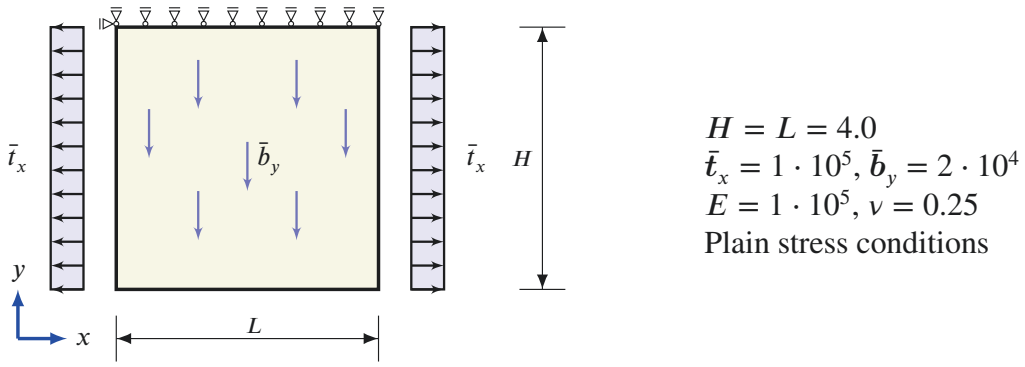
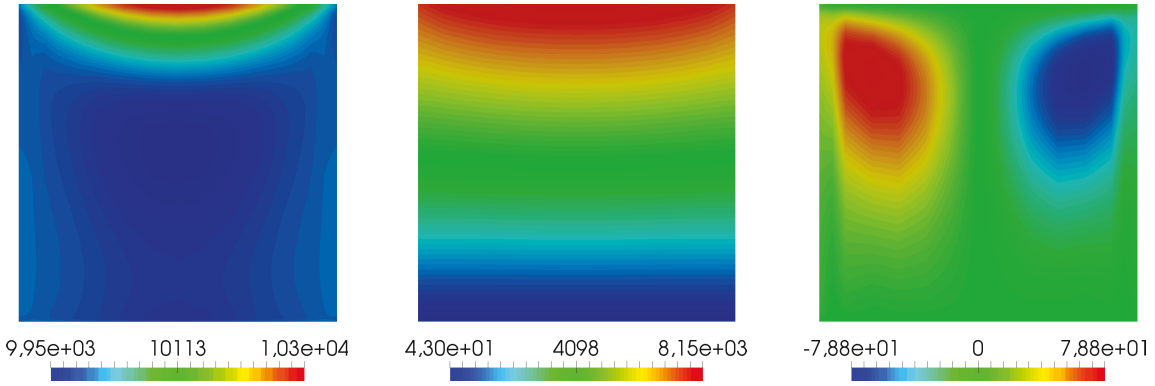


Figure 6.16: Surface under tension problem

selected such that the minimum value of the σ_{xx} stress is larger than the maximum value of the σ_{yy} stress. The magnitude of the shear stress is significantly smaller than that of the two normal components. That way, the x-direction contributes more to the global strain energy than the y-direction does.



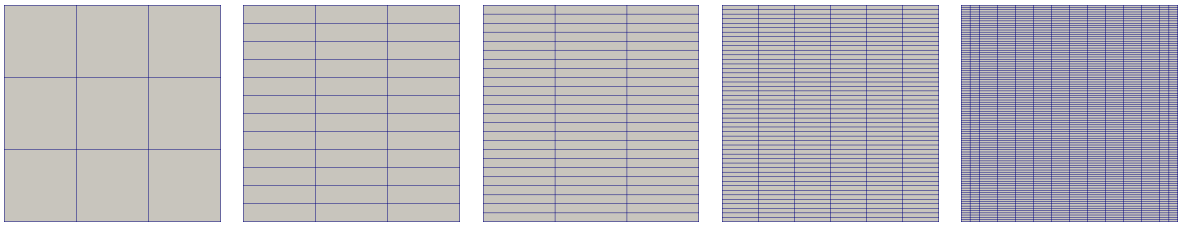
(a) σ_{xx} **(b)** σ_{yy} **(c)** σ_{xy}

Figure 6.17: Stress results for adaptive, direction specific refinement at step 93

With the linear elements only being capable of representing a constant stress field, it is evident that a larger number of elements is required in vertical direction to make up for the linear stress field in that direction as compared to the constant stresses in the x-direction. As there are no locations with a stress concentration or singularities, a clustering of elements is not expected.

The problem is computed and adaptively refined with the error estimator outlined in sect. 6.4.2. In each iteration, the element with the largest error value is refined according to one of the three strategies: (a) refinement in all parametric directions, (b) refinement in the direction of the largest element extension, and (c) according to the direction indicator $\boldsymbol{\Phi}_e^{(i)}$ defined in eq. (6.14). For reference, also a uniform refinement was conducted. All computations start with the initial mesh depicted in fig. 6.18(a).

Whereas the strategies (a) and (b) result in uniform refinement, the direction specific approach leads to a mesh that matches the initial considerations. The associated mesh evolution is illustrated with fig. 6.18. This evolution also finds expression in the chart on the true energy error, which is provided in fig. 6.19. Strategies (a) and (b) perform similar to the uniform refinement. The direction specific refinement, however, performs significantly better up to approximately 100 DOFs, which is associated with refinement step 9 depicted in fig. 6.18(b). Beyond that, the



(a) Step 0 **(b)** Step 9 **(c)** Step 21 **(d)** Step 45 **(e)** Step 93

Figure 6.18: Mesh evolution for adaptive, direction specific refinement

respective error curve runs parallel to that of the uniform refinement error. At that point, the nearly constant stresses in x-direction are assumingly of the same quality as the linear stresses in y-direction. Regarding the further mesh evolution in figs. 6.18(c) to 6.18(e), it can be noted that the element aspect ratio remains approximately constant and hence, uniform refinement is also performed by the direction specific approach after step 9.

When the Poisson ratio is set to $\nu = 0$, stresses in x and y are completely decoupled. The shear stress is zero throughout the domain and $\sigma_{xx} = \bar{t}_x$. Analyzing that variant with direction specific refinement causes a refinement of the y-direction only. The stress in x-direction is expressed exactly with a single element. These results, which are not further illustrated here, support the idea that at refinement step 9 of the original problem, the stress fields in x- and y-direction are of the same quality.

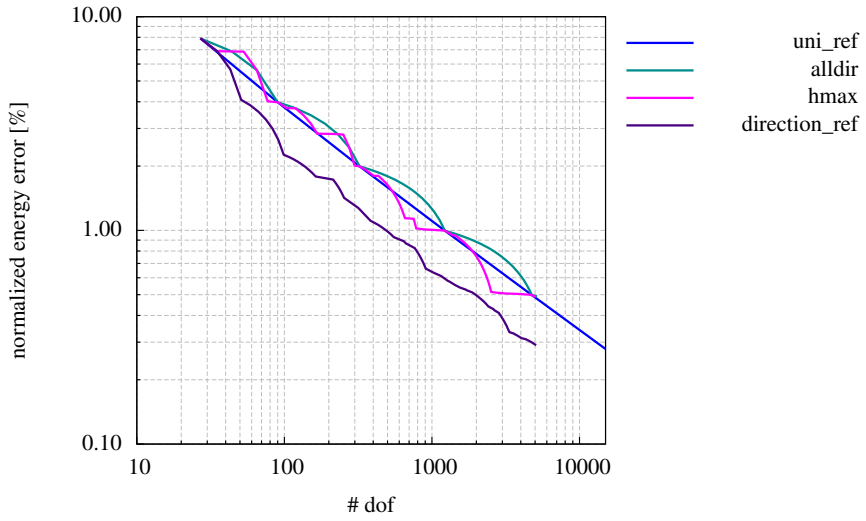
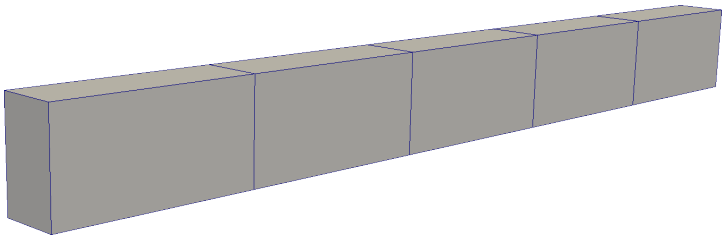


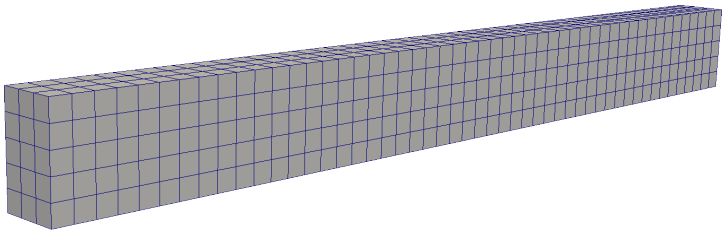
Figure 6.19: True relative error for the iteratively refined surface under tension example.

Clamped solid beam example

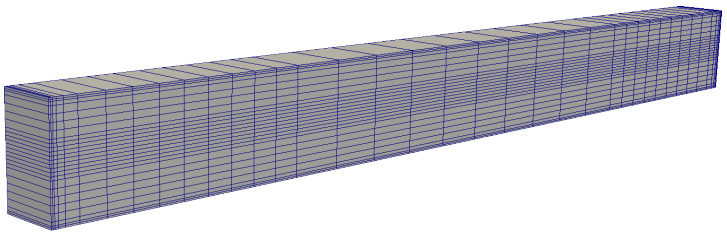
In a second investigation on the direction specific refinement, the previously examined clamped solid beam example is analyzed under the aspect of adaptive refinement. Here, the beam variant with $h = 0.50$ and quadratic basis functions in all parametric directions is used. The problem setting for this variant is illustrated in fig. 6.1(b) and also the relevant model parameters are given in sect. 6.2.



(a) Initial



(b) Maximum element edge length, $l_e = 0.1$



(c) Adaptive direction specific refinement, step 35

Figure 6.20: Mesh variants for the clamped solid beam example

The initial mesh of the model is depicted in fig. 6.20(a). In sect. 6.2, that mesh is h-refined by gradually decreasing the maximum allowed element edge length l_e. That approach initially refines the elements in the longitudinal beam axis only. Refinement over the height and the width happens only after the value for the maximum allowed edge length is below the respective dimension of the cross section. At lower values of l_e, the elements constitute small cubes and the mesh is very uniform as can be seen in fig. 6.20(b), in which the mesh is depicted for $l_e = 0.1$. In this section, the approach is compared to the direction specific refinement, starting from the same initial mesh. And for reference, also the case of uniform refinement is included.

The performance, which can be seen from the error chart in fig. 6.21 is ambivalent. The first refinement iterations of the direction specific approach refine the x- and the z-directions simultaneously due to large xz shear error contributions. That significantly increases the number of DOFs compared to the x-direction only refinement of the maximum edge length approach. At coarse discretizations, the performance of the direction refinement is thus inferior to the edge length variant and similar to uniform refinement. With progressing refinement, however, the direction specific approach outperforms the other two strategies; at least until a certain refinement level is reached. Beyond that point, the error estimator starts to pick up on the stress singularities at the top and bottom edges of the clamped beam faces. When that happens, the performance deteriorates. Due to the patch global impact of the refinement, too many new DOFs are created in the vicinity of these edges, which are not made up for by a related reduction of the global energy error. The final mesh before that happens is shown in fig. 6.20(c). Considering the applied boundary conditions, this mesh appears to be very reasonable. Espe-

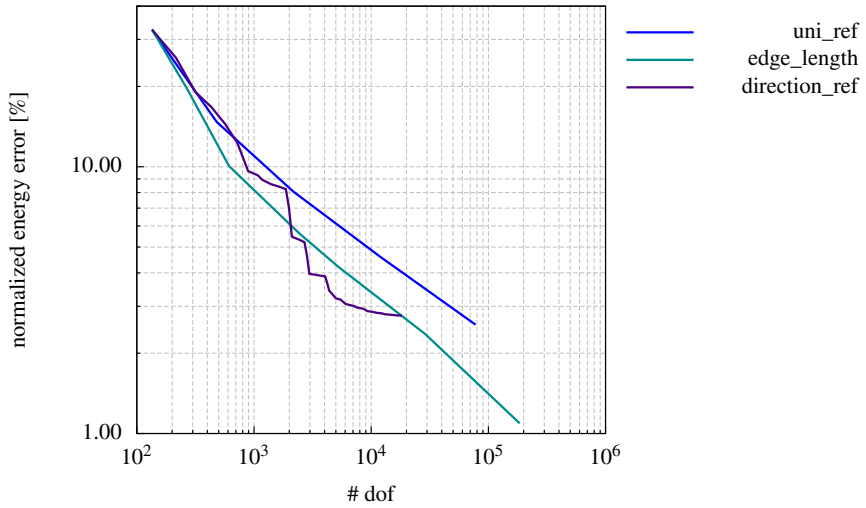


Figure 6.21: True relative error for the iteratively refined clamped solid beam example.

cially, it is to note that no refinement occurred in y-direction.

Using NURBS, refinement is always patch global. The poor behavior due to the existence of stress singularities can therefore not be cured by an improved error estimator but only with a spline formulation that allows for local refinement. Evaluating the final mesh in fig. 6.20(c), points out that many elements were created in z-direction, which is likely due to shear error contributions. One approach to improve the performance might be to weight normal and shear error contributions differently.

Finally, it is noted that the direction indicator may also be used to decide on the direction of p-refinement. Adding the direction specific error contributions over all elements of a patch allows identifying the direction with the highest error contribution in a patch. Anisotropic p-refinement, however, has not proved to be very advantageous for this solid example.

Chapter 7

Summary, conclusions, and outlook

The prototypical implementation of the integrated structural analysis framework developed in the course of this work (also cf. appendix C) proves the concept of automatically obtaining and analyzing a structural analysis model from the pure geometric description of a civil engineering structure to work. The necessary geometry data should be retrieved from a comprehensive digital model, the building information model, in which the description of the geometry constitutes an essential part. The concept enables structural engineers to quickly evaluate new design ideas that are proposed by architects and provided through an updated model in the BIM database; it allows them for their part to provide architects with profound feedback regarding the structural feasibility of their ideas. Necessary modifications of the design that ensure the structural stability may, after incorporating them into the analysis model, be directly forwarded to the BIM database, as both models found on the same geometry description. The involved parties perform incremental modifications on a shared digital model and hence, easily exchange mutually important information. Pursuing this procedure significantly improves the digital collaboration of architects and engineers, which was the main goal of this work.

It is yet to note that the collaboration between architects and engineers is just one aspect in the complex relations among all specialists involved in a construction project. The practical implementation of the proposed concept requires all parties to adopt the use of the common digital model. This will require many modifications on the software side as well as changes of the current workflow that are not easy to achieve. It is, however, the strong belief of the author that an efficient digital collaboration is only possible if the latest revision of shared information, of which the building geometry is the most important, is available to and used by everyone involved. This requires redundancies to be eliminated and information to be stored such that it is understandable, i.e. processable, by everyone. Sector specific abstractions counteract this idea and will thus not lead to the desired result. This work demonstrates the possibility of finding a common basis for a comprehensive digital model with regard to the geometry and the structural analysis. Comparable work is yet required in the other sectors.

The following list provides an overview of the main steps involved in developing the integrated structural analysis framework, complemented by the respective findings.

Collaboration

- An investigation of the state of building information modeling on the basis of industry foundation classes revealed some deficiencies with regard to their suitability for the digital collaboration. In the opinion of the author, the current state of the IFC schema may be suited to support the collaboration of specialists from the same field, but it is not for the cooperation across the individual domain boundaries of architecture and structural

engineering. On the contrary, by providing independent schemes for the storage of respective geometric and topological data, it supports model redundancies that are likely to cause discrepancies, which are not easily resolved.

- The redundancies in the definition of IFC based BIM models are a direct result of the different disciplines using different abstractions of reality in their models. In this work, the structural analysis is therefore based on 3D solid models. Such an analysis model is obtained directly from the geometric description of the building provided with the BIM data. This approach removes complexity from the data model and furthermore ensures consistency between architecture and structural engineering, as their respective models base on the same data. Respective modifications on the data representation with the IFC scheme are suggested.
- The proposed procedure allows to create the analysis model automatically on the basis of the provided geometry thus simplifying new structural calculations for updated design variants. Presuming the load state, material parameters, and alike were previously defined, or at maximum, have to be suplemented for modified parts of the structure, the updated BIM dataset is easily transferred into a new analysis model, which allows for a fast evaluation of that specific design. This is of special importance in the early drafting phase in order to perform a manual exploration of the space of possible designs and to eventually find an optimal solution for the design task.

Analysis model

- The solid geometry of the structure is expressed by volumetric NURBS. Thus, the concept of isogeometric analysis is incorporated into the integrated analysis approach. The structural analysis model does not only base on the same geometry description as the architectural model, but it uses that formulation throughout the entire analysis process, which allows to retain the link between entities in the BIM model and individual patches representing complete structural elements or major parts thereof. Though, it is often necessary to enrich the function space of the solution variable and thus to refine the parametrization of individual patches, the link to the BIM data is preserved at any time. And as usual in IGA, the exact geometry of the structure is considered in the analysis.
- Geometrical compatibility between individual patches is a condition that the geometry model must fulfill. Any requirements on the compatibility of the mesh inherent to adjacent patches are, however, not expedient, as it would severely restrict the geometrical modelling process. The presumably non-conforming meshes of these patches are therefore coupled with the mortar method with Lagrange multipliers, a weak coupling method that enforces the consistency of interface tractions.
- The mortar method was shown to work for volumetric NURBS patches in a true multi-subdomain environment. The Lagrange multiplier fields representing the interface tractions were interpolated with bilinear Lagrangian functions and alternatively with 2D NURBS functions inherited from the underlying patches. Both options work and the magnitude of the displacement incompatibility at the interfaces, which is a direct result of the weak coupling, does, in neither case, appear to be significant for civil engineering purposes. The presumed superiority of the NURBS interpolation could yet not be confirmed with regard to the global model behavior. It only has its merits when stress

results in the direct vicinity of coupling interfaces are of special interest. Also the choice of the non-mortar side was evaluated. It was not found to have a relevant effect on the result quality, yet it does have an effect on the numerical cost. Therefore, it is suggested to choose the non-mortar side such that the number of Lagrange multipliers is minimized – provided all underlying patches are reasonably discretized.

- In order to not rely on coupling information within the BIM data, which would constitute a source of redundancy in the model, all necessary information on the patch coupling should be deduced from the geometric model. This is not trivial because of the parametric geometry representation with NURBS. This work proposes a hierarchical procedure for this purpose and provides the related algorithms as pseudocode. The procedure allows for an efficient evaluation of the coupling interfaces and provides a discretization thereof that constitutes the basis for the numerical integration of the mortar matrices.
- The isogeometric concept does not require to mesh the geometry in a costly procedure as the utilized spline formulations feature an inherent subdivision, which is used as the basis for finite elements. Yet, that discretization cannot be presumed to be appropriate for obtaining sufficiently accurate results. Therefore, it is necessary to refine the patches that are extracted from the BIM data. Two approaches were suggested to steer that refinement, one is based on empirical model knowledge, the other is steered by an error estimator.
- Automated empirical refinement is the favored variant to steer model refinement in the integrated analysis approach for civil engineering purposes. It allows for an automatic solution procedure based on the BIM model, does not necessarily require repeated solution iterations, and incorporates the knowledge and the specific requirements of the structural engineer in charge. The refinement procedure builds on the geometric and possibly also the structural properties of the patches and their contact relations; it is controlled by related parameters and allows to consider anisotropic patch or structural element properties respectively. The performance of this approach was demonstrated with an example, it could be shown to be superior to the use of naively refined meshes.
- The second approach of steering the refinement bases on a recovery error estimator of the Zienkiewicz and Zhu type, which was adapted to be used with NURBS based isogeometric analysis. For this purpose, superconvergent point results are recovered with an also adapted least-square-fit algorithm that was originally proposed to approximate point clouds with NURBS surfaces. The algorithm indicates elements with large energy error contributions. That indication is used to steer the refinement process and though h-refinement is always patch-global, also this approach could be shown to perform better than uniform refinement.

Numerical solution

- The numerical analysis of civil engineering structures with solid models results in a large number of active degrees of freedom. Using also higher degree basis functions in combination with solid elements for that analysis causes the linear system's stiffness matrix not only to be large but to have a high bandwidth as well. The application of the mortar method with Lagrange multipliers does furthermore produce an indefinite coefficient matrix, the associated system of equations is known as saddle point problem. In sum, the complexity and the cost of the solution process increase distinctively. For that reason,

various approaches to solving the linear system efficiently are investigated, the specific system properties arising from the integrated analysis method are thereby considered and utilized.

- The large system size calls for the use of iterative methods. Different iterative methods of the Krylov subspace family were tested together with a number of preconditioners. Out of those tested, the SQMR solver is considered the most adequate because of its flexibility regarding the use of a preconditioner. The convergence of the iterative solution relies substantially on the preconditioner. Its selection thus has a much higher impact than the selection of the iterative solver. The best performance was achieved with a preconditioner on the basis of an incomplete LU factorization and adaptive area-based dropping. In general, it is to note that the cost of the preconditioner evaluation can become significant as can the related memory requirement. Convergence of the solution is yet not guaranteed.
- Owing to the shortcoming of the iterative solution, also direct methods were investigated. The specific layout of the coefficient matrix motivates a substructuring solution procedure. The large global problem is reduced to a significantly smaller inner problem, which is much better suited for direct solution. The reduction of the matrix is computed in parallel on the patch level. Considering that a typical structural analysis model contains many patches, each of which with a comparatively small patch stiffness matrix, this introduces a great amount of parallelism to the solution process. Local model modifications, like changing a beams cross section or moving it to a different location, do not require to discard an existing substructuring solution completely. The affected patches have to be covered in a new reduction process and the inner problem must be solved again, but the reduction of all non-affected patches can be reused and thus, a lot of the numerical effort can be saved.
- The performance of the solvers was tested and compared on a practical example, for which the substructuring approach could be shown to perform best. In practical civil engineering applications, the system behavior is usually assumed to be linear and a large number of load situations has to be considered. Therefore, direct methods are likely to outperform the iterative solvers anytime. Their numerical effort does not significantly increase after solving for the first load state, which is not the case with iterative methods. This circumstance strongly supports the use of the substructuring solver in the integrated analysis framework.

Future work on the outlined framework has to be considered from two viewpoints, the more global or general perspective on the practical implementation and the particular structural analysis view. On behalf of the latter, specific tasks can be formulated whose fulfillment would improve the performance of the simulation approach suggested in this work. One issue is the extension of the framework to volumetric spline formulations that allow local refinement, e.g. to T-splines or PHT-splines, with the former also facilitating the geometrical modeling of solid bodies with genus higher than zero, i.e. to allow holes. With patch local refinement, the total numerical effort could be reduced without compromising on result accuracy. And though at least T-splines do in theory also allow for a structure-wide conforming mesh, that should not be a goal when introducing another spline formulation, as such a mesh impedes the hierarchical breakdown into building elements. And during geometrical modeling, the requirement to create a conforming mesh would have even more implications than the solid modeling with

non-conforming solid splines. A second useful enhancement would be provided by a postprocessing procedure that allows to smooth primary and secondary solution fields across weak coupling interfaces. Owing to the non-interpolatory nature of the spline formulations, this, however, is not a trivial task.

With regard to the more general outlook, the availability of a geometric modeler that allows the creation of volumetric NURBS with the standard tools of CAD software would greatly improve the prospects for a practical implementation. Though research on this topic was done in the past, the author is not aware of any software with a graphical user interface that would allow such modelling. The lack thereof is a not only an obstruction to this work, but to isogeometric analysis with solid geometry in general.

Bibliography

[1] M. Abramovici and O. C. Sieg. Status and development trends of product lifecycle management systems. In *Proceedings of the International Converence on Integrated Product and Process Development, Wroclaw, 21.-22.11.2002*, 2002.

[2] C. Adam, T. J. R. Hughes, S. Bouabdallah, M. Zarroug, and H. Maitournam. Selective and reduced numerical integrations for NURBS-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 284:732–761, 2015.

[3] M. Ainsworth and J. T. Oden. A posteriori error estimation in finite element analysis. *Computer Methods in Applied Mechanics and Engineering*, 142(96):1–88, 1997.

[4] M. Ainsworth, J. Z. Zhu, A. W. Craig, and O. C. Zienkiewicz. Analysis of the Zienkiewicz-Zhu a-posteriori error estimator in the finite element method. *International Journal for Numerical Methods in Engineering*, 28(February):2161–2174, 1989.

[5] H. Altenbach. *Kontinuumsmechanik*. Springer Vieweg, 2nd edition, 2012. ISBN 978-3642241185.

[6] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[7] R. Amor. Technical challenges for integrated design and delivery solutions in civil and building engineering. In *Int. Conf. on Computing in Civil and Building Engineering, University of Nottingham, Nottingham, UK*, 2010.

[8] C. Anitescu, Y. Jia, Y. J. Zhang, and T. Rabczuk. An isogeometric collocation method using superconvergent points. *Computer Methods in Applied Mechanics and Engineering*, 284:1073–1097, 2015.

[9] A. Apostolatos, R. Schmidt, R. Wüchner, and K. U. Bletzinger. A Nitsche-type formulation and comparison of the most common domain decomposition methods in isogeometric analysis. *International Journal for Numerical Methods in Engineering*, 97:473–504, 2014.

[10] Associated General Contractors of America. *The Contractor's Guide to BIM*. AGC Research Foundation, Las Vegas, NV, 1st edition, 2005.

[11] C. E. Augarde and A. J. Deeks. The use of Timoshenko's exact solution for a cantilever beam in adaptive analysis. *Finite Elements in Analysis and Design*, 44:595–601, 2008.

[12] F. Auricchio, F. Calabro, T. J. R. Hughes, A. Reali, and G. Sangalli. A simple algorithm for obtaining nearly optimal quadrature rules for NURBS-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 249:15–27, 2012.

[13] S. Azhar. Building information modeling (BIM): Trends, benefits, risks, and challenges for the AEC industry. *Leadership and Management in Engineering*, 11(3):241–252, 2011.

[14] I. Babuška and W. C. Rheinboldt. A-posteriori error estimates for the finite element method. *International Journal for Numerical Methods in Engineering*, 12(10):1597–1615, 1978.

[15] I. Babuška and W. C. Rheinboldt. Analysis of optimal finite-element meshes in R^1. *Mathematics of Computation*, 33(146):435 – 463, 1979.

[16] J. Barlow. Optimal stress locations in finite element models. *International Journal for Numerical Methods in Engineering*, 10(2):243–251, 1976.

[17] K.-J. Bathe. *Finite Element Procedures*. Prentice-Hall, Upper Saddle River, 1996. ISBN 978-8126529988.

[18] Y. Bazilevs and T. Hughes. NURBS-based isogeometric analysis for the computation of flows about rotating components. *Computational Mechanics*, 43(1):143–150, 2008.

[19] Y. Bazilevs, V. Calo, Y. Zhang, and T. J. R. Hughes. Isogeometric fluid–structure interaction analysis with applications to arterial blood flow. *Computational Mechanics*, 38(4-5):310–322, 2006.

[20] Y. Bazilevs, V. M. Calo, J. A. Cottrell, J. A. Evans, T. J. R. Hughes, S. Lipton, M. A. Scott, and T. W. Sederberg. Isogeometric analysis using T-splines. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):229 – 263, 2010.

[21] Y. Bazilevs, M.-C. Hsu, I. Akkerman, S. Wright, K. Takizawa, B. Henicke, T. Spielman, and T. Tezduyar. 3D simulation of wind turbine rotors at full scale. Part I: Geometry modeling and aerodynamics. *International Journal for Numerical Methods in Fluids*, 65(1-3):207–235, 2011.

[22] Y. Bazilevs, M.-C. Hsu, J. Kiendl, R. Wüchner, and K.-U. Bletzinger. 3D simulation of wind turbine rotors at full scale. Part II: Fluid–structure interaction modeling with composite blades. *International Journal for Numerical Methods in Fluids*, 65(1-3):236–253, 2011.

[23] J. Beetz, L. van Berlo, R. de Laat, and P. van den Helm. BIMserver.org – An open source IFC model server. In *Proceedings of the CIP W78 conference*, Cairo, Egypt, Nov. 2010.

[24] L. Beirão da Veiga, A. Buffa, D. Cho, and G. Sangalli. Isogeometric analysis using T-splines on two-patch geometries. *Computer Methods in Applied Mechanics and Engineering*, 200(21-22):1787 – 1803, 2011.

[25] F. B. Belgacem. The Mortar finite element method with Lagrange multipliers. *Numerische Mathematik*, 84:173–197, 1999.

[26] D. J. Benson, Y. Bazilevs, M. C. Hsu, and T. J. R. Hughes. Isogeometric shell analysis: the Reissner-Mindlin shell. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):276 – 289, 2010.

[27] D. J. Benson, Y. Bazilevs, M. C. Hsu, and T. J. R. Hughes. A large deformation, rotation-free, isogeometric shell. *Computer Methods in Applied Mechanics and Engineering*, 200(13-16):1367 – 1378, 2011.

[28] M. Benzi and A. J. Wathen. Some preconditioning techniques for saddle point problems. In W. H. A. Schilders, H. A. van der Vorst, and J. Rommes, editors, *Model order reduction: Theory, research aspects and applications*, volume 13 of *Mathematics in Industry*, pages 195–211. Springer, 2008. ISBN 978-3-540-78840-9. doi: 10.1007/978-3-540-78841-6_10. URL `http://dx.doi.org/10.1007/978-3-540-78841-6_10`.

[29] M. Benzi, G. H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14: 1–137, 2005.

[30] C. Bernardi, Y. Maday, and A. T. Patera. A new nonconforming approach to domain decomposition: The Mortar element method. In H. Brezis and J. L. Lions, editors, *Nonlinear partial differential equations and their applications. Collège de France Seminar, Vol. XI (Paris, 1989-1991)*, volume 299 of *Pitman Research Notes in Mathematics Series*, pages 13–51. Harlow, Longman Scientific & Technical, 1994.

[31] S. Bernstein. Démonstration du théorème de Weierstrass fondée sur le calcul des probabilités. *Communications of the Kharkov Mathematical Society*, XIII:1–2, 1912.

[32] P. E. Bézier. How Renault uses numerical control for car body design and tooling. Technical report, SAE Technical Paper, 1968.

[33] R. Bouclier, T. Elguedj, and A. Combescure. An isogeometric locking-free NURBS-based solid-shell element for geometrically nonlinear analysis. *International Journal for Numerical Methods in Engineering*, 101(10):774–808, 2015.

[34] A. Buffa and C. Giannelli. Adaptive isogeometric methods with hierarchical splines: error estimator and convergence. *arXiv preprint arXiv:1502.00565*, 2015.

[35] A. Buffa, G. Sangalli, and R. Vázquez. Isogeometric analysis in electromagnetics: B-splines approximation. *Computer Methods in Applied Mechanics and Engineering*, 199(17):1143–1152, 2010.

[36] P. P.-S. Chen. The entity-relationship model – Toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.

[37] E. Chow and A. Patel. Fine-Grained Parallel Incomplete LU Factorization. *SIAM J. Sci. Comput.*, 37: C169–C193, 2015.

[38] E. Chow and Y. Saad. Experimental study of ILU preconditioners of indenite matrices. *J. Comput. Appl. Math.*, 86:387–414, 1997.

[39] E. Cohen, T. Martin, R. Kirby, T. Lyche, and R. Riesenfeld. Analysis-aware modeling: Understanding quality considerations in modeling for isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 199(5–8):334–356, 2010. Computational Geometry and Analysis.

[40] N. Collier, L. Dalcin, D. Pardo, and V. M. Calo. The cost of continuity: Performance of iterative solvers on isogeometric finite elements. *SIAM Journal on Scientific Computing*, 35:A767–A784, 2013.

[41] J. A. Cottrell, A. Reali, Y. Bazilevs, and T. J. R. Hughes. Isogeometric analysis of structural vibrations. *Computer methods in applied mechanics and engineering*, 195(41):5257–5296, 2006.

[42] J. A. Cottrell, T. J. R. Hughes, and A. Reali. Studies of refinement and continuity in isogeometric structural analysis. *Computer Methods in Applied Mechanics and Engineering*, 196(41-44):4160 – 4183, 2007.

[43] J. A. Cottrell, T. J. R. Hughes, and Y. Bazilevs. *Isogeometric analysis: toward integration of CAD and FEA*. Wiley Publishing, 1st edition, 2009.

[44] M. G. Cox. The numerical evaluation of B-splines. *IMA Journal of Applied Mathematics*, 10(2):134–149, 1972.

[45] L. B. Da Veiga, A. Buffa, G. Sangalli, and R. Vázquez. Analysis-suitable t-splines of arbitrary degree: definition and properties. *Math. Models Methods Appl. Sci*, 23:1979–2003, 2013.

[46] L. Dagum and R. Enon. OpenMP: An industry standard API for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[47] C. de Boor. On calculating with B-splines. *Journal of Approximation Theory*, 6(1):50–62, 1972.

[48] L. De Lorenzis, P. Wriggers, and G. Zavarise. A mortar formulation for 3D large deformation contact using NURBS-based isogeometric analysis and the augmented Lagrangian method. *Computational Mechanics*, 49:1–20, 2012.

[49] J. W. Demmel, J. R. Gilbert, and X. S. Li. An asynchronous parallel supernodal algorithm for sparse gaussian elimination. *SIAM J. Matrix Analysis and Applications*, 20(4):915–952, 1999.

[50] J. Deng, F. Chen, X. Li, C. Hu, W. Tong, Z. Yang, and Y. Feng. Polynomial splines over hierarchical T-meshes. *Graphical Models*, 70(4):76 – 86, 2008.

[51] P. Díez, J. J. Egozcue, and A. Huerta. A posteriori error estimation for standard finite element analysis. *Computer Methods in Applied Mechanics and Engineering*, 163(1):141–157, 1998.

[52] T. Dokken, T. Lyche, and K. F. Pettersen. Polynomial splines over locally refined box-partitions. *Computer Aided Geometric Design*, 30(3):331–356, 2013.

[53] M. R. Dörfel, B. Jüttler, and B. Simeon. Adaptive isogeometric analysis by local h-refinement with T-splines. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):264 – 275, 2010.

[54] W. Dornisch, G. Vitucci, and 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*, 103:205–234, 2015.

[55] D. A. Dunavant. High degree efficient symmetrical Gaussian quadrature rules for the triangle. *International Journal for Numerical Methods in Engineering*, 21(6):1129–1148, 1985.

[56] A. Düster, J. Parvizian, Z. Yang, and E. Rank. The finite cell method for three-dimensional problems of solid mechanics. *Computer methods in applied mechanics and engineering*, 197(45):3768–3782, 2008.

[57] C. Eastman, P. Teicholz, R. Sacks, and K. Liston. *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors*. John Wiley & Sons, Hoboken, NJ, 2nd edition, 2011.

[58] C. M. Eastman. *Building product models: computer environments, supporting design and construction*. CRC press, 1999.

[59] C. M. Eastman and A. Siabiris. A generic building product model incorporating building type information. *Automation in Construction*, 3(4):283 – 304, 1995.

[60] R. Echter and M. Bischoff. Numerical efficiency, locking and unlocking of NURBS finite elements. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):374–382, 2010.

[61] C. Farhat, M. Lesoinne, P. LeTallec, K. Pierson, and D. Rixen. FETI-DP: A dual-primal unified FETI method–Part I: A faster alternative to the two-level FETI method. *International journal for numerical methods in engineering*, 50(7):1523–1544, 2001.

[62] G. Farin. *Curves and surfaces for CAGD (Fifth edition)*. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, San Francisco, 5th edition, 2002. ISBN 978-1-55860-737-8.

[63] M. Fastabend, T. Schäfers, M. Albert, and H.-G. Lommen. Zur sinnvollen Anwendung ganzheitlicher Gebäudemodelle in der Tragwerksplanung von Hochbauten. *Beton- und Stahlbetonbau*, 104(10):657–663, 2009.

[64] S. J. Fenves, S. Foufou, C. Bock, and R. D. Sriram. CPM: A core model for product data. *Journal of Computing and Information Science in Engineering*, 8(1), 2008.

[65] A. R. Forrest. Interactive interpolation and approximation by Bézier polynomials. *The Computer Journal*, 15(1):71–79, 1972.

[66] R. W. Freund and N. M. Nachtigal. A new Krylov-subspace method for symmetric indefinite linear systems. In *Proceedings of the 14th IMACS World Congress on Computational and Applied Mathematics*, pages 1253–1256, 1994.

[67] M. P. Gallaher, A. C. O'Connor, and J. L. Dettbarn. Cost analysis of inadequate interoperability in the US capital facilities industry. *National Institute of Standards and Technology (NIST)*, 2004.

[68] A. George and J. W. H. Liu. *Computer solution of large sparse positive definite systems*. Prentice-Hall series in computational mathematics. Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[69] F. Gerold. *Konzepte zur interaktiven Entwurfsraum-Exploration im Tragwerksentwurf*. PhD thesis, Bauhaus-Universität Weimar, 2013.

[70] C. Giannelli, B. Jüttler, and H. Speleers. THB-splines: The truncated basis for hierarchical splines. *Computer Aided Geometric Design*, 29(7):485–498, 2012.

[71] G. H. Golub and C. F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996. ISBN 0-8018-5414-8.

[72] S. Gottschalk. *Collision queries using oriented bounding boxes*. PhD thesis, The University of North Carolina, Chapel Hill, NC, USA, 2000.

[73] T. Grätsch and K.-J. Bathe. A posteriori error estimation techniques in practical finite element analysis. *Computers & Structures*, 83(4-5):235 – 265, 2005.

[74] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[75] N. Gu and K. London. Understanding and facilitating BIM adoption in the AEC industry. *Automation in Construction*, 19(8):988–999, 2010. The role of VR and BIM to manage the construction and design processes.

[76] W. Günthner and A. Borrmann, editors. *Digitale Baustelle - innovativer Planen, effizienter Ausführen*. VDI-Buch. Springer Berlin / Heidelberg, 2011.

[77] M. F. Hardwick, R. L. Clay, P. T. Boggs, E. J. Walsh, A. R. Larzelere, and A. Altshuler. DART System analysis. Technical report, Sandia National Laboratories, Albuquerque, 2005.

[78] C. Hesch and P. Betsch. Isogeometric analysis and domain decomposition methods. *Computer Methods in Applied Mechanics and Engineering*, 213–216(0):104–112, 2012.

[79] T. Horger, S. Kollmannsberger, F. Frischmann, E. Rank, and B. Wohlmuth. A new mortar formulation for modeling elastomer bedded structures with modal-analysis in 3D. *Advanced Modeling and Simulation in Engineering Sciences*, 2(18), 2014.

[80] K. Hormann and A. Agathos. The point in polygon problem for arbitrary polygons. *Computational Geometry*, 20(3):131 – 144, 2001.

[81] T. Huang, Y. Zhang, and L. Li. Factorization for solving electromagnetic scattering problems. *Progress In Electromagnetics Research*, 13:41–58, 2009.

[82] T. J. R. Hughes. *The Finite Element Method*. Dover Publications, 2000. ISBN 978-0486411811.

[83] T. J. R. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: CAD, finite elements, NURBS, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194 (39-41):4135 – 4195, 2005.

[84] T. J. R. Hughes, A. Reali, and G. Sangalli. Duality and unified analysis of discrete approximations in structural dynamics and wave propagation: Comparison of p-method finite elements with k-method NURBS. *Computer methods in applied mechanics and engineering*, 197(49):4104–4124, 2008.

[85] T. J. R. Hughes, A. Reali, and G. Sangalli. Efficient quadrature for NURBS-based isogeometric analysis. *Computer Methods in Applied Mechanics and Engineering*, 199(5-8):301 – 313, 2010.

[86] ISO 16739. *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries*. International Organization for Standardization, 2013.

[87] Y.-S. Jeong, C. Eastman, R. Sacks, and I. Kaner. Benchmark tests for BIM data exchanges of precast concrete. *Automation in Construction*, 18(4):469 – 484, 2009.

[88] K. A. Johannessen, T. Kvamsdal, and T. Dokken. Isogeometric analysis using LR B-splines. *Computer Methods in Applied Mechanics and Engineering*, 269:471–514, 2014.

[89] K. A. Johannessen, F. Remonato, and T. Kvamsdal. On the similarities and differences between Classical Hierarchical, Truncated Hierarchical and LR B-splines. *Computer Methods in Applied Mechanics and Engineering*, 291:64–101, 2015.

[90] X. Juvigny and J. Ryan. Preconditioners for domain decomposition methods. *Computers & Mathematics with Applications*, 42(01):1143–1155, 2001.

[91] J. Kiendl, K. U. Bletzinger, J. Linhard, and R. Wüchner. Isogeometric shell analysis with Kirchhoff-Love elements. *Computer Methods in Applied Mechanics and Engineering*, 198(49-52):3902–3914, 2009.

[92] J. Kiendl, Y. Bazilevs, M. C. Hsu, R. Wüchner, and K. U. Bletzinger. The bending strip method for isogeometric analysis of Kirchhoff-Love shell structures comprised of multiple patches. *Computer Methods in Applied Mechanics and Engineering*, 199(37-40):2403–2416, 2010.

[93] J. Kiendl, R. Schmidt, R. Wüchner, and K.-U. Bletzinger. Isogeometric shape optimization of shells using semi-analytical sensitivity analysis and sensitivity weighting. *Computer Methods in Applied Mechanics and Engineering*, 274:148–167, 2014.

[94] E. G. Kirsch. Die Theorie der Elastizität und die Bedürfnisse der Festigkeitslehre. *Zeitschrift des Vereines deutscher Ingenieure*, 42:797–807, 1898.

[95] G. Kiss, C. Giannelli, and B. Jüttler. Algorithms and data structures for truncated hierarchical b–splines. In *Mathematical Methods for Curves and Surfaces*, pages 304–323. Springer, 2014.

[96] S. K. Kleiss and S. K. Tomar. Guaranteed and sharp a posteriori error estimates in isogeometric analysis. *Computers & Mathematics with Applications*, 2015.

[97] S. K. Kleiss, C. Pechstein, B. Jüttler, and S. Tomar. IETI - Isogeometric Tearing and Interconnecting. *Computer Methods in Applied Mechanics and Engineering*, 247-248:201–215, 2012.

[98] S. Kollmannsberger, A. Özcan, J. Baiges, M. Ruess, E. Rank, and A. Reali. Parameter-free, weak imposition of Dirichlet boundary conditions and coupling of trimmed and non-conforming patches. *International Journal for Numerical Methods in Engineering*, 101(9):670–699, 2015. doi: 10.1002/nme.4817.

[99] G. Kuru, C. V. Verhoosel, K. G. van der Zee, and E. H. van Brummelen. Goal-adaptive Isogeometric Analysis with hierarchical splines. *Computer Methods in Applied Mechanics and Engineering*, 270:270–292, 2014.

[100] M. Laakso and A. Kiviniemi. The IFC standard: A review of history, development, and standardization. *Journal of Information Technology in Construction (ITcon)*, 17(9):134–161, May 2012.

[101] W. M. Lai, D. Rubin, and E. Krempl. *Introduction to Continuum Mechanics*. Butterworth-Heinemann, Boston, 4th edition, 2010. ISBN 978-0-7506-8560-3.

[102] S. H. Lee and Y. S. Jeong. A system integration framework through development of ISO 10303-based product model for steel bridges. *Automation in Construction*, 15:212–228, 2006.

[103] X. S. Li. SuperLU : Sparse Direct Solver and Preconditioner. 13th DOE ACTS Collection Workshop, 2004.

[104] X. S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):302–325, 2005.

[105] X. S. Li and M. Shao. A supernodal approach to incomplete LU factorization with partial pivoting. *ACM Transactions on Mathematical Software*, 37:1–20, 2011.

[106] Z. Li and K. Ito. *The immersed interface method: numerical solutions of PDEs involving interfaces and irregular domains*, volume 33. Siam, 2006.

[107] S. Lipton, J. Evans, Y. Bazilevs, T. Elguedj, and T. Hughes. Robustness of isogeometric structural discretizations under severe mesh distortion. *Computer Methods in Applied Mechanics and Engineering*, 199 (5–8):357–373, 2010. Computational Geometry and Analysis.

[108] A. N. Lowan, N. Davids, and A. Levenson. Table of the zeros of the legendre polynomials of order 1-16 and the weight coefficients for gauss' mechanical quadrature formula. *Bull. Amer. Math. Soc.*, 48(10):739–743, 10 1942.

[109] Y. L. Ma and W. Hewitt. Point inversion and projection for nurbs curve and surface: Control polygon approach. *Computer Aided Geometric Design*, 20(2):79 – 99, 2003.

[110] A. Maceri. *Theory of Elasticity*. Springer, Berlin Heidelberg, 2010. ISBN 978-3-642-11391-8.

[111] Y. Maday, C. Mavriplis, and A. Patera. Nonconforming Mortar element methods: Application to spectral discretizations. In *Domain decomposition methods; Proceedings of the second international symposium, Los Angeles, CA; United States; 14-16 Jan. 1988*, pages 392 – 418, 1989.

[112] G. E. Mase and G. T. Mase. *Continuum mechanics for engineers*. CRC Press, Boca Raton, FL, 2nd edition, 1999. ISBN 0-8493-1855-6.

[113] A. Meister. *Numerik linearer Gleichungssysteme*. Vieweg + Teubner, 4^{th} edition, 2011.

[114] H. Mersch. Projekträume im Internet, Teil 1 – Anforderungen, Technik, Funktionalität für einen optimalen Einsatz. *Deutsches Architektenblatt*, Feb. 2006.

[115] H. Mersch. Projekträume im Internet, Teil 2 – Überlegungen vor Projektstart, Checkliste, Erfahrungen, Kosten. *Deutsches Architektenblatt*, Mar. 2006.

[116] M. Moumnassi, S. Belouettar, É. Béchet, S. P. Bordas, D. Quoirin, and M. Potier-Ferry. Finite element analysis on implicitly defined domains: An accurate representation based on arbitrary parametric surfaces. *Computer Methods in Applied Mechanics and Engineering*, 200(5):774–796, 2011.

[117] V. P. Nguyen, P. Kerfriden, M. Brino, S. P. a. Bordas, and E. Bonisoli. Nitsche's method for two and three dimensional NURBS patch coupling. *Computational Mechanics*, 53:1163–1182, 2014.

[118] N. Nguyen-Thanh, J. Kiendl, H. Nguyen-Xuan, R. Wüchner, K. Bletzinger, Y. Bazilevs, and T. Rabczuk. Rotation free isogeometric thin shell analysis using PHT-splines. *Computer Methods in Applied Mechanics and Engineering*, 200(47-48):3410 – 3424, 2011.

[119] N. Nguyen-Thanh, H. Nguyen-Xuan, S. Bordas, and T. Rabczuk. Isogeometric analysis using polynomial splines over hierarchical T-meshes for two-dimensional elastic solids. *Computer Methods in Applied Mechanics and Engineering*, 200(21-22):1892 – 1908, 2011.

[120] N. Nguyen-Thanh, J. Muthu, X. Zhuang, and T. Rabczuk. An adaptive three-dimensional RHT-splines formulation in linear elasto-statics and elasto-dynamics. *Computational Mechanics*, 53(2):369–385, 2014.

[121] A. K. Niggl. *Tragwerksanalyse am volumenorientierten Gesamtmodell*. PhD thesis, Technische Universität München, 2007.

[122] M. Nour and K. Beucke. Object Versioning as a basis for design change management within a BIM context. In W. Tizani, editor, *Proceedings of the International Conference on Computing in Civil and Building Engieering*, Nottingham, 2010. Nottingham University Press.

[123] C. Oblonsek and N. Guid. A fast surface-based procedure for object reconstruction from 3d scattered points. *Computer Vision and Image Understanding*, 69(2):185 – 195, 1998.

[124] R. Owen, R. Amor, M. Palmer, J. Dickinson, C. B. Tatum, A. S. Kazi, M. Prins, A. Kiviniemi, and B. East. Challenges for integrated design and delivery solutions. *Architectural Engineering and Design Management*, 6(4):232–240, 2010.

[125] C. C. Paige and M. A. Saunders. Solution of sparse indefinitie systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975.

[126] P. Papadopoulos. ME185 - Introduction to Continuum Mechanics. downloaded on 2015-04-21 from http://www.me.berkeley.edu/ME280B, 2008.

[127] J. Parvizian, A. Düster, and E. Rank. Finite cell method: h- and p-extension for embedded domain problems in solid mechanics. *Computational Mechanics*, 41(1):121–133, 2007.

[128] T. Pazlar and Z. Turk. Interoperability in practice: Geometric data exchance using the ifc standard. *ITcon Special Issue: Case studies of BIM use*, 13:362–380, 2008.

[129] L. A. Piegl and W. Tiller. *The NURBS Book*. Monographs in visual communication. Springer, 2nd edition, 1997.

[130] L. A. Piegl and W. Tiller. Parametrization for surface fitting in reverse engineering. *Computer-Aided Design*, 33(8):593 – 603, 2001.

[131] J. Plume and J. Mitchell. Collaborative design using a shared IFC building model – Learning from experience. *Automation in Construction*, 16(1):28–36, 2007. CAAD Futures, 2005.

[132] M. A. Puso. A 3D Mortar method for solid mechanics. *International Journal for Numerical Methods in Engineering*, 59(3):315–336, 2004.

[133] E. Rank, M. Ruess, S. Kollmannsberger, D. Schillinger, and A. Düster. Geometric modeling, isogeometric analysis and the finite cell method. *Computer Methods in Applied Mechanics and Engineering*, 249-252: 104–115, 2012.

[134] K. A. Reed. The role of the CIMSteel integration standards in automating the erection and surveying of structural steelwork. In *International Symposium on Automation and Robotics in Construction, 19th (ISARC)*, pages 15–20, Gaithersburg, Maryland, 2002. National Institute of Standards and Technology (NIST).

[135] D. F. Rogers. *An introduction to NURBS: with historical perspective*. Morgan Kaufmann Publishers Inc., 2001.

[136] R. Romberg. *Gebäudemodell-basierte Strukturanalyse im Bauwesen*. PhD thesis, Technische Universität München, 2005.

[137] M. Ruess, D. Schillinger, Y. Bazilevs, V. Varduhn, and E. Rank. Weakly enforced essential boundary conditions for NURBS-embedded and trimmed NURBS geometries on the basis of the finite cell method. *International Journal for Numerical Methods in Engineering*, 95(10):811–846, 2013.

[138] M. Ruess, D. Schillinger, A. I. Özcan, and E. Rank. Weak coupling for isogeometric analysis of non-matching and trimmed multi-patch geometries. *Computer Methods in Applied Mechanics and Engineering*, 269:46–71, 2014.

[139] Y. Saad. ILUT: A dual threshold incomplete LU factorization. *Numerical Linear Algebra with Applications*, 1(4):387–402, 1994.

[140] Y. Saad. *Preconditioned Krylov subspace methods for CFD applications*, pages 139–158. Computational Methods in Mechanics and Applied Sciences. Wiley, 1995.

[141] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[142] M. H. Sadd. *Elasticity*. Academic Press, Boston, 2nd edition, 2009. ISBN 978-0-12-374446-3.

[143] O. Samuelson and B.-C. Björk. Adoption processes for EDM, EDI and BIM technologies in the construction industry. *Journal of Civil Engineering and Management*, 19(sup1):S172–S187, 2013.

[144] P. Sanguinetti, S. Abdelmohsen, J. Lee, J. Lee, H. Sheward, and C. Eastman. General system architecture for BIM: An integrated approach for design and analysis. *Advanced Engineering Informatics*, 26(0):–, 2012. doi: 10.1016/j.aei.2011.12.001.

[145] D. Schillinger and E. Rank. An unfitted hp-adaptive finite element method based on hierarchical B-splines for interface problems of complex geometry. *Comp Meth in App Mech and Eng*, 200(47–48):3358–3380, 2011.

[146] D. Schillinger, L. Dedè, M. A. Scott, J. A. Evans, M. J. Borden, E. Rank, and T. J. R. Hughes. An isogeometric design-through-analysis methodology based on adaptive hierarchical refinement of NURBS, immersed boundary methods, and T-spline CAD surfaces. *Computer Methods in Applied Mechanics and Engineering*, 249-252(January):116–150, 2012.

[147] A. Schollmeyer and B. Froehlich. Direct isosurface ray casting of NURBS-based isogeometric analysis. *Visualization and Computer Graphics, IEEE Transactions on*, 20(9):1227–1240, 2014.

[148] K. Schrader. *Hybrid 3D simulation methods for the damage analysis of multiphase composites*. PhD thesis, Fakultät Bauingenieurwesen, Bauhaus-Universität Weimar, 2012.

[149] M. A. Scott, X. Li, T. W. Sederberg, and T. J. R. Hughes. Local refinement of analysis-suitable T-splines. *Computer Methods in Applied Mechanics and Engineering*, 213–216:206 – 222, 2011. doi: 10.1016/j.cma.2011.11.022. accepted manuscript.

[150] T. W. Sederberg, J. Zheng, A. Bakenov, and A. Nasri. T-splines and T-NURCCs. *ACM Transactions on Graphics*, 22:477–484, July 2003.

[151] Y.-D. Seo, H.-J. Kim, and S.-K. Youn. Shape optimization and its extension to topological design based on isogeometric analysis. *International Journal of Solids and Structures*, 47(11–12):1618 – 1640, 2010.

[152] J. R. Shewchuk. What is a good linear finite element? Interpolation, conditioning, anisotropy, and quality measures. Technical report, University of California at Berkeley, 2002.

[153] J. Steel, R. Drogemuller, and B. Toth. Model interoperability in building information modelling. *Software & Systems Modeling*, 11(1):99–109, 2012. ISSN 1619-1366.

[154] P. Stein. *Procedurally generated models for Isogeometric Analysis*. PhD thesis, Bauhaus-Universität Weimar, Weimar, 2012.

[155] B. Succar. Building information modelling framework: A research and delivery foundation for industry stakeholders. *Automation in Construction*, 18(3):357 – 375, 2009.

[156] H. Sutter. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3):202–210, 2005. URL `http://www.gotw.ca/publications/concurrency-ddj.htm`.

[157] I. Temizer, P. Wriggers, and T. J. R. Hughes. Three-dimensional Mortar-based frictional contact treatment in isogeometric analysis with NURBS. *Computer Methods in Applied Mechanics and Engineering*, 209: 115–128, 2012.

[158] S. P. Timoshenko and J. N. Goodier. *Theory of Elasticity*. McGraw-Hill, 2nd edition, 1951.

[159] H. A. van der Vorst. *Iterative Krylov methods for large linear systems*, volume 13 of *Cambridge monographs on applied and computational mathematics*. Cambridge University Press, 2003. ISBN 9780511615115.

[160] R. Verfürth. A review of a posteriori error estimation techniques for elasticity problems. *Computer Methods in Applied Mechanics and Engineering*, 176(1-4):419–440, 1999.

[161] C. Verhoosel, M. Scott, R. de Borst, and T. Hughes. An Isogeometric Approach To Cohesive Zone Modeling. *International Journal for Numerical Methods in Engineering*, 87(1-5):336–360, 2011.

[162] K. J. Versprille. *Computer-aided Design Applications of the Rational B-spline Approximation Form*. PhD thesis, Syracuse, NY, USA, 1975.

[163] W. A. Wall, M. A. Frenzel, and C. Cyron. Isogeometric structural shape optimization. *Computer Methods in Applied Mechanics and Engineering*, 197(33-40):2976 – 2988, 2008. ISSN 0045-7825. doi: 10.1016/j.cma.2008.01.025.

[164] C. Wan, P.-H. Chen, and R. L. K. Tiong. Assessment of IFCs for structural analysis domain. *Information Technology in Construction*, 9:75–95, 2004.

[165] P. Wang, J. Xu, J. Deng, and F. Chen. Adaptive isogeometric analysis using rational PHT-splines. *Computer-Aided Design*, 43(11):1438 – 1448, 2011.

[166] B. Wassermann, T. Bog, S. Kollmannsberger, and E. Rank. A design-through-analysis approach using the finite cell method. In M. Papadrakakis, V. Papadopoulos, G. Stefanou, V. Plevris, editor, *ECCOMAS Congress 2016*, June 2016.

[167] M. Weise and P. Katranuschkov. Supporting State-based Transactions in Collaborative Product Modelling Environments. In R. J. Scherer, P. Katranuschkov, and S.-E. Schapke, editors, *CIB-W78 Conference on Information Technology in Construction*, Dresden, 2005.

[168] C. Willberg, S. Duczek, J. V. Perez, D. Schmicker, and U. Gabbert. Comparison of different higher order finite element schemes for the simulation of lamb waves. *Computer Methods in Applied Mechanics and Engineering*, 241:246–261, 2012.

[169] B. I. Wohlmuth. A Mortar finite element method using dual spaces for the lagrange multiplier. *SIAM Journal on Numerical Analysis*, 38(3):989 – 1012, 2001.

[170] N. Zander, T. Bog, M. Elhaddad, F. Frischmann, S. Kollmannsberger, and E. Rank. The multi-level hp-method for three-dimensional problems: Dynamically changing high-order mesh refinement with arbitrary hanging nodes. *Computer Methods in Applied Mechanics and Engineering*, 310:252–277, 2016.

[171] O. C. Zienkiewicz and J. Z. Zhu. A simple error estimator and adaptive procedure for practical engineerng analysis. *International Journal for Numerical Methods in Engineering*, 24(2):337–357, 1987.

[172] O. C. Zienkiewicz and J. Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. Part 1: The recovery technique. *International Journal for Numerical Methods in Engineering*, 33(7):1331–1364, 1992.

[173] O. C. Zienkiewicz and J. Z. Zhu. The superconvergent patch recovery and a posteriori error estimates. Part 2: Error estimates and adaptivity. *International Journal for Numerical Methods in Engineering*, 33(7): 1365–1382, 1992.

[174] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu. *The Finite Element Method: Its Basis and Fundamentals*. Butterworth-Heinemann, Burlington, 6th edition, 2000. ISBN 978-0750663205.

[175] M. Zlámal. Superconvergence and reduced integration in the finite element method. *Mathematics of computation*, 32(143):663–685, 1978.

Appendices

Appendix A IFC extension: NURBS solids

The subsequent code sections contain the EXPRESS schema specifications for the extension of IFC4 with NURBS solids. They base on existing specification for NURBS curves and surfaces and correspond to fig. 2.6 in sect. 2.4.

```
ENTITY IfcSolidModel
 ABSTRACT SUPERTYPE OF(ONEOF(IfcBoundedSolid, IfcCsgSolid, IfcManifoldSolidBrep,
     ↪ IfcSweptAreaSolid, IfcSweptDiskSolid))
 SUBTYPE OF IfcGeometricRepresentationItem;
 DERIVE
  Dim : IfcDimensionCount := 3;
END_ENTITY;
```

```
ENTITY IfcBoundedSolid
 ABSTRACT SUPERTYPE OF(IfcBSplineSolid)
 SUBTYPE OF IfcSolidModel;
END_ENTITY;
```

```
ENTITY IfcBSplineSolid
 ABSTRACT SUPERTYPE OF(IfcBSplineSolidWithKnots)
 SUBTYPE OF IfcBoundedSolid;
  UDegree           : INTEGER;
  VDegree           : INTEGER;
  WDegree           : INTEGER;
  ControlPointsList : LIST [2:?] OF LIST [2:?] OF LIST [2:?] OF IfcCartesianPoint;
  SolidForm         : IfcBSplineSolidForm;
  UClosed           : LOGICAL;
  VClosed           : LOGICAL;
  WClosed           : LOGICAL;
  SelfIntersect     : LOGICAL;
 DERIVE
  UUpper            : INTEGER := SIZEOF(ControlPointsList) - 1;
  VUpper            : INTEGER := SIZEOF(ControlPointsList[1]) - 1;
  WUpper            : INTEGER := SIZEOF(ControlPointsList[1][1]) - 1;
  ControlPoints     : ARRAY [0:UUpper] OF ARRAY [0:VUpper] OF ARRAY [0:WUpper] OF
     ↪ IfcCartesianPoint := IfcMakeArrayOfArrayOfArray(ControlPointsList,
     ↪ 0,UUpper,0,VUpper, 0,WUpper);
```

```
ENTITY IfcBSplineSolidWithKnots
 SUPERTYPE OF(IfcRationalBSplineSolidWithKnots)
 SUBTYPE OF IfcBSplineSolid;
  UMultiplicities : LIST [2:?] OF INTEGER;
  VMultiplicities : LIST [2:?] OF INTEGER;
  WMultiplicities : LIST [2:?] OF INTEGER;
  UKnots          : LIST [2:?] OF IfcParameterValue;
  VKnots          : LIST [2:?] OF IfcParameterValue;
  WKnots          : LIST [2:?] OF IfcParameterValue;
  KnotSpec        : IfcKnotType;
 DERIVE
  KnotUUpper      : INTEGER := SIZEOF(UKnots);
  KnotVUpper      : INTEGER := SIZEOF(VKnots);
  KnotWUpper      : INTEGER := SIZEOF(WKnots);
 WHERE
  UDirectionConstraints : IfcConstraintsParamBSpline ( SELF\IfcBSplineSolid.UDegree,
     ↪ KnotUUpper, SELF\IfcBSplineSolid.UUpper, UMultiplicities, UKnots);
```

```
  VDirectionConstraints : IfcConstraintsParamBSpline ( SELF\IfcBSplineSolid.VDegree,
      ↪ KnotVUpper, SELF\IfcBSplineSolid.VUpper, VMultiplicities, VKnots);
  WDirectionConstraints : IfcConstraintsParamBSpline ( SELF\IfcBSplineSolid.WDegree,
      ↪ KnotWUpper, SELF\IfcBSplineSolid.WUpper, WMultiplicities, WKnots);
  CorrespondingULists   : SIZEOF(UMultiplicities) = KnotUUpper;
  CorrespondingVLists   : SIZEOF(VMultiplicities) = KnotVUpper;
  CorrespondingWLists   : SIZEOF(WMultiplicities) = KnotWUpper;
END_ENTITY;
```

```
ENTITY IfcRationalBSplineSolidWithKnots
 SUBTYPE OF IfcBSplineSolidWithKnots;
  WeightsData : LIST [2:?] OF LIST [2:?] OF LIST [2:?] OF REAL;
 DERIVE
  Weights : ARRAY [0:UUpper] OF ARRAY [0:VUpper] OF ARRAY [0:WUpper] OF REAL :=
      ↪ IfcMakeArrayOfArrayOfArray(WeightsData,0,UUpper,0,VUpper,0,WUpper);
 WHERE
  CorrespondingWeightsDataLists : (SIZEOF(WeightsData) =
      ↪ SIZEOF(SELF\IfcBSplineSolid.ControlPointsList)) AND (SIZEOF(WeightsData[1])
      ↪ = SIZEOF(SELF\IfcBSplineSolid.ControlPointsList[1]) AND
      ↪ (SIZEOF(WeightsData[1][1]) =
      ↪ SIZEOF(SELF\IfcBSplineSolid.ControlPointsList[1][1]));
  WeightValuesGreaterZero       : IfcSolidWeightsPositive(SELF);
END_ENTITY;
```

```
TYPE IfcBSplineSolidForm = ENUMERATION OF (
 UNSPECIFIED);
END_TYPE;
```

```
FUNCTION IfcMakeArrayOfArrayofArray
 (Lis : LIST[1:?] OF LIST [1:?] OF LIST [1:?] OF GENERIC : T;
 Low1, U1, Low2, U2, Low3,U3 : INTEGER):
 ARRAY [Low1:U1] OF ARRAY [Low2:U2] OF ARRAY [Low3:U3] OF GENERIC : T;

  LOCAL
    Res : ARRAY[Low1:U1] OF ARRAY [Low2:U2] OF ARRAY [Low3:U3] OF GENERIC : T;
  END_LOCAL;

  (* Check input dimensions for consistency *)
  IF (U1-Low1+1) <> SIZEOF(Lis) THEN
    RETURN (?);
  END_IF;
  IF (U2 - Low2 + 1 ) <> SIZEOF(Lis[1]) THEN
    RETURN (?) ;
  END_IF;
  IF (U3 - Low3 + 1 ) <> SIZEOF(Lis[1][1]) THEN
    RETURN (?) ;
  END_IF;

  (* Initialise Res with values from Lis[1] *)
  Res := [IfcArrayofArrayArray(Lis[1], Low2, U2, Low3, U3) : (U1-Low1 + 1)];
  REPEAT i := 2 TO HIINDEX(Lis);
    IF (U2-Low2+1) <> SIZEOF(Lis[i]) THEN
      RETURN (?);
    END_IF;
    Res[Low1+i-1] := IfcArrayOfArray(Lis[i], Low2, U2, Low3, U3);
  END_REPEAT;
  RETURN (Res);
END_FUNCTION;
```

```
FUNCTION IfcSolidWeightsPositive
( B: IfcRationalBSplineSolidWithKnots)
: BOOLEAN;

  LOCAL
    Result : BOOLEAN := TRUE;
  END_LOCAL;

  REPEAT i := 0 TO B\IfcBSplineSurface.UUpper;
    REPEAT j := 0 TO B\IfcBSplineSurface.VUpper;
      REPEAT k := 0 TO B\IfcBSplineSurface.WUpper;
        IF (B.Weights[i][j][k] <= 0.0) THEN
          Result := FALSE;
          RETURN(Result);
        END_IF;
      END_REPEAT;
    END_REPEAT;
  END_REPEAT;
  RETURN(Result);
END_FUNCTION;
```

Appendix B MultiStory example: NURBS geometry definition

For the multistory example of sect. 6.3.4, the geometry of the investigated structure is specified with the subsequent listing. It comprises the definition of 23 NURBS patches in their initial, unrefined state. The listing is written in XML. The associated document type definition describing the schema of the XML listing is included in the listing's header. NURBS data specified in this format is parsed and exported by the developed framework.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Structure [
<!ELEMENT Structure (NurbsPatch+)>
<!ELEMENT NurbsPatch (Dimension, Degree+, KnotVector+, ControlArray)>
<!ELEMENT Dimension (#PCDATA)>
<!ELEMENT Degree (#PCDATA)>
<!ELEMENT KnotVector (#PCDATA)>
<!ELEMENT ControlArray (Sequence)>
<!ELEMENT Sequence (Sequence|ControlPoint)+>
<!ELEMENT ControlPoint (Vertex, Weight, Displacement?, Rotation?, Temperature?, Strain?, Stress? )>
<!ELEMENT Vertex (#PCDATA)>
<!ELEMENT Weight (#PCDATA)>
<!ELEMENT Displacement (#PCDATA)>
<!ELEMENT Rotation (#PCDATA)>
<!ELEMENT Temperature (#PCDATA)>
<!ELEMENT Strain (#PCDATA)>
<!ELEMENT Stress (#PCDATA)>
<!ATTLIST Structure
id  ID  #IMPLIED
>
<!ATTLIST NurbsPatch
id  CDATA #REQUIRED
>
<!ATTLIST Dimension
size  CDATA \#REQUIRED
type  (real|int) #REQUIRED
>
<!ATTLIST Degree
dim   CDATA #REQUIRED
size  CDATA #REQUIRED
type  (real|int) #REQUIRED
>
<!ATTLIST KnotVector
dim   CDATA #REQUIRED
size  CDATA #REQUIRED
type  (real|int) #REQUIRED
>
<!ATTLIST ControlPoint
id  CDATA #REQUIRED
>
<!ATTLIST Vertex
size  CDATA #REQUIRED
type  (real|int) #REQUIRED
>
<!ATTLIST Weight
size  CDATA #REQUIRED
type  (real|int) #REQUIRED
>
<!ATTLIST Displacement
size  CDATA #REQUIRED
type  (real|int) #REQUIRED
>
<!ATTLIST Rotation
size  CDATA #REQUIRED
type  (real|int) #REQUIRED
>
<!ATTLIST Temperature
size  CDATA #REQUIRED
type  (real|int) #REQUIRED
>
<!ATTLIST Strain
size  CDATA #REQUIRED
type  (real|int) #REQUIRED
>
<!ATTLIST Stress
size  CDATA #REQUIRED
type  (real|int) #REQUIRED
>
]>
<Structure>
  <NurbsPatch id="1">
    <Dimension size="1" type="int">3</Dimension>
    <Degree dim ="1" size="1" type="int">1</Degree>
    <Degree dim ="2" size="1" type="int">1</Degree>
    <Degree dim ="3" size="1" type="int">1</Degree>
    <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
```

```
    <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <ControlArray>
      <Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="1">
              <Vertex size="3" type="real">0.00000000 0.00000000 0.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="2">
              <Vertex size="3" type="real">0.00000000 0.00000000 3.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="3">
              <Vertex size="3" type="real">0.00000000 5.00000000 0.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="4">
              <Vertex size="3" type="real">0.00000000 5.00000000 3.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="5">
              <Vertex size="3" type="real">0.40000000 0.00000000 0.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="6">
              <Vertex size="3" type="real">0.40000000 0.00000000 3.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="7">
              <Vertex size="3" type="real">0.40000000 5.00000000 0.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="8">
              <Vertex size="3" type="real">0.40000000 5.00000000 3.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
      </Sequence>
    </ControlArray>
  </NurbsPatch>
  <NurbsPatch id="2">
    <Dimension size="1" type="int">3</Dimension>
    <Degree dim ="1" size="1" type="int">1</Degree>
    <Degree dim ="2" size="1" type="int">1</Degree>
    <Degree dim ="3" size="1" type="int">1</Degree>
    <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <ControlArray>
      <Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="10">
              <Vertex size="3" type="real">0.00000000 0.00000000 3.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="11">
              <Vertex size="3" type="real">0.00000000 0.00000000 3.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="12">
              <Vertex size="3" type="real">0.00000000 5.00000000 3.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="13">
              <Vertex size="3" type="real">0.00000000 5.00000000 3.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="14">
              <Vertex size="3" type="real">15.00000000 0.00000000 3.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="15">
              <Vertex size="3" type="real">15.00000000 0.00000000 3.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="16">
              <Vertex size="3" type="real">15.00000000 5.00000000 3.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
```

```
          </ControlPoint>
          <ControlPoint id="17">
            <Vertex size="3" type="real">15.00000000 5.00000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="3">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="19">
            <Vertex size="3" type="real">14.60000000 0.00000000 0.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="20">
            <Vertex size="3" type="real">14.60000000 0.00000000 3.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="21">
            <Vertex size="3" type="real">14.60000000 5.00000000 0.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="22">
            <Vertex size="3" type="real">14.60000000 5.00000000 3.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="23">
            <Vertex size="3" type="real">15.00000000 0.00000000 0.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="24">
            <Vertex size="3" type="real">15.00000000 0.00000000 3.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="25">
            <Vertex size="3" type="real">15.00000000 5.00000000 0.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="26">
            <Vertex size="3" type="real">15.00000000 5.00000000 3.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="4">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="28">
            <Vertex size="3" type="real">0.40000000 0.00000000 0.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="29">
            <Vertex size="3" type="real">0.40000000 0.00000000 3.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="30">
            <Vertex size="3" type="real">0.40000000 0.40000000 0.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="31">
            <Vertex size="3" type="real">0.40000000 0.40000000 3.00000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
```

```
          </Sequence>
        </Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="32">
              <Vertex size="3" type="real">14.60000000 0.00000000 0.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="33">
              <Vertex size="3" type="real">14.60000000 0.00000000 3.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="34">
              <Vertex size="3" type="real">14.60000000 0.40000000 0.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="35">
              <Vertex size="3" type="real">14.60000000 0.40000000 3.00000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
      </Sequence>
    </ControlArray>
  </NurbsPatch>
  <NurbsPatch id="5">
    <Dimension size="1" type="int">3</Dimension>
    <Degree dim ="1" size="1" type="int">1</Degree>
    <Degree dim ="2" size="1" type="int">1</Degree>
    <Degree dim ="3" size="1" type="int">1</Degree>
    <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <ControlArray>
      <Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="37">
              <Vertex size="3" type="real">3.00000000 0.00000000 3.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="38">
              <Vertex size="3" type="real">3.00000000 0.00000000 7.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="39">
              <Vertex size="3" type="real">3.00000000 0.40000000 3.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="40">
              <Vertex size="3" type="real">3.00000000 0.40000000 7.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="41">
              <Vertex size="3" type="real">6.00000000 0.00000000 3.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="42">
              <Vertex size="3" type="real">6.00000000 0.00000000 7.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="43">
              <Vertex size="3" type="real">6.00000000 0.40000000 3.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="44">
              <Vertex size="3" type="real">6.00000000 0.40000000 7.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
      </Sequence>
    </ControlArray>
  </NurbsPatch>
  <NurbsPatch id="6">
    <Dimension size="1" type="int">3</Dimension>
    <Degree dim ="1" size="1" type="int">1</Degree>
    <Degree dim ="2" size="1" type="int">1</Degree>
    <Degree dim ="3" size="1" type="int">1</Degree>
    <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <ControlArray>
      <Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="46">
```

```
            <Vertex size="3" type="real">9.00000000 0.00000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="47">
            <Vertex size="3" type="real">9.00000000 0.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="48">
            <Vertex size="3" type="real">9.00000000 0.40000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="49">
            <Vertex size="3" type="real">9.00000000 0.40000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="50">
            <Vertex size="3" type="real">12.00000000 0.00000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="51">
            <Vertex size="3" type="real">12.00000000 0.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="52">
            <Vertex size="3" type="real">12.00000000 0.40000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="53">
            <Vertex size="3" type="real">12.00000000 0.40000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="7">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="55">
            <Vertex size="3" type="real">0.00000000 1.00000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="56">
            <Vertex size="3" type="real">0.00000000 1.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="57">
            <Vertex size="3" type="real">0.00000000 5.00000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="58">
            <Vertex size="3" type="real">0.00000000 5.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="59">
            <Vertex size="3" type="real">0.40000000 1.00000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="60">
            <Vertex size="3" type="real">0.40000000 1.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="61">
            <Vertex size="3" type="real">0.40000000 5.00000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="62">
            <Vertex size="3" type="real">0.40000000 5.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
```

```
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="8">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="64">
            <Vertex size="3" type="real">14.60000000 1.00000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="65">
            <Vertex size="3" type="real">14.60000000 1.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="66">
            <Vertex size="3" type="real">14.60000000 5.00000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="67">
            <Vertex size="3" type="real">14.60000000 5.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="68">
            <Vertex size="3" type="real">15.00000000 1.00000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="69">
            <Vertex size="3" type="real">15.00000000 1.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="70">
            <Vertex size="3" type="real">15.00000000 5.00000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="71">
            <Vertex size="3" type="real">15.00000000 5.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="9">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="73">
            <Vertex size="3" type="real">0.00000000 0.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="74">
            <Vertex size="3" type="real">0.00000000 0.00000000 7.90000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="75">
            <Vertex size="3" type="real">0.00000000 5.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="76">
            <Vertex size="3" type="real">0.00000000 5.00000000 7.90000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="77">
            <Vertex size="3" type="real">15.00000000 0.00000000 7.40000000</Vertex>
```

```
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="78">
            <Vertex size="3" type="real">15.00000000 0.00000000 7.90000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="79">
            <Vertex size="3" type="real">15.00000000 5.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="80">
            <Vertex size="3" type="real">15.00000000 5.00000000 7.90000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="10">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="82">
            <Vertex size="3" type="real">0.40000000 4.60000000 6.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="83">
            <Vertex size="3" type="real">0.40000000 4.60000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="84">
            <Vertex size="3" type="real">0.40000000 5.00000000 6.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="85">
            <Vertex size="3" type="real">0.40000000 5.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="86">
            <Vertex size="3" type="real">14.60000000 4.60000000 6.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="87">
            <Vertex size="3" type="real">14.60000000 4.60000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="88">
            <Vertex size="3" type="real">14.60000000 5.00000000 6.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="89">
            <Vertex size="3" type="real">14.60000000 5.00000000 7.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="11">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="91">
            <Vertex size="3" type="real">6.37500000 4.60000000 3.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="92">
            <Vertex size="3" type="real">6.37500000 4.60000000 6.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
```

```
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="93">
              <Vertex size="3" type="real">6.37500000 5.00000000 3.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="94">
              <Vertex size="3" type="real">6.37500000 5.00000000 6.80000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="95">
              <Vertex size="3" type="real">8.62500000 4.60000000 3.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="96">
              <Vertex size="3" type="real">8.62500000 4.60000000 6.80000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="97">
              <Vertex size="3" type="real">8.62500000 5.00000000 3.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="98">
              <Vertex size="3" type="real">8.62500000 5.00000000 6.80000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
      </Sequence>
    </ControlArray>
  </NurbsPatch>
  <NurbsPatch id="12">
    <Dimension size="1" type="int">3</Dimension>
    <Degree dim ="1" size="1" type="int">1</Degree>
    <Degree dim ="2" size="1" type="int">1</Degree>
    <Degree dim ="3" size="1" type="int">1</Degree>
    <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <ControlArray>
      <Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="100">
              <Vertex size="3" type="real">4.90000000 2.60000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="101">
              <Vertex size="3" type="real">4.90000000 2.60000000 8.95000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="102">
              <Vertex size="3" type="real">4.90000000 3.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="103">
              <Vertex size="3" type="real">4.90000000 3.00000000 8.95000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="104">
              <Vertex size="3" type="real">10.10000000 2.60000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="105">
              <Vertex size="3" type="real">10.10000000 2.60000000 8.95000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="106">
              <Vertex size="3" type="real">10.10000000 3.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="107">
              <Vertex size="3" type="real">10.10000000 3.00000000 8.95000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
      </Sequence>
    </ControlArray>
  </NurbsPatch>
  <NurbsPatch id="13">
    <Dimension size="1" type="int">3</Dimension>
```

```
    <Degree dim ="1" size="1" type="int">1</Degree>
    <Degree dim ="2" size="1" type="int">1</Degree>
    <Degree dim ="3" size="1" type="int">1</Degree>
    <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <ControlArray>
      <Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="109">
              <Vertex size="3" type="real">4.50000000 0.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="110">
              <Vertex size="3" type="real">4.50000000 0.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="111">
              <Vertex size="3" type="real">4.50000000 3.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="112">
              <Vertex size="3" type="real">4.50000000 3.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="113">
              <Vertex size="3" type="real">4.90000000 0.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="114">
              <Vertex size="3" type="real">4.90000000 0.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="115">
              <Vertex size="3" type="real">4.90000000 3.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="116">
              <Vertex size="3" type="real">4.90000000 3.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
      </Sequence>
    </ControlArray>
  </NurbsPatch>
  <NurbsPatch id="14">
    <Dimension size="1" type="int">3</Dimension>
    <Degree dim ="1" size="1" type="int">1</Degree>
    <Degree dim ="2" size="1" type="int">1</Degree>
    <Degree dim ="3" size="1" type="int">1</Degree>
    <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <ControlArray>
      <Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="118">
              <Vertex size="3" type="real">10.10000000 0.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="119">
              <Vertex size="3" type="real">10.10000000 0.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="120">
              <Vertex size="3" type="real">10.10000000 3.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="121">
              <Vertex size="3" type="real">10.10000000 3.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="122">
              <Vertex size="3" type="real">10.50000000 0.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="123">
              <Vertex size="3" type="real">10.50000000 0.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
```

```
          </Sequence>
          <Sequence>
            <ControlPoint id="124">
              <Vertex size="3" type="real">10.50000000 3.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="125">
              <Vertex size="3" type="real">10.50000000 3.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
      </Sequence>
    </ControlArray>
  </NurbsPatch>
  <NurbsPatch id="15">
    <Dimension size="1" type="int">3</Dimension>
    <Degree dim ="1" size="1" type="int">1</Degree>
    <Degree dim ="2" size="1" type="int">1</Degree>
    <Degree dim ="3" size="1" type="int">1</Degree>
    <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <ControlArray>
      <Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="127">
              <Vertex size="3" type="real">0.00000000 4.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="128">
              <Vertex size="3" type="real">0.00000000 4.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="129">
              <Vertex size="3" type="real">0.00000000 5.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="130">
              <Vertex size="3" type="real">0.00000000 5.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="131">
              <Vertex size="3" type="real">0.40000000 4.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="132">
              <Vertex size="3" type="real">0.40000000 4.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="133">
              <Vertex size="3" type="real">0.40000000 5.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="134">
              <Vertex size="3" type="real">0.40000000 5.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
      </Sequence>
    </ControlArray>
  </NurbsPatch>
  <NurbsPatch id="16">
    <Dimension size="1" type="int">3</Dimension>
    <Degree dim ="1" size="1" type="int">1</Degree>
    <Degree dim ="2" size="1" type="int">1</Degree>
    <Degree dim ="3" size="1" type="int">1</Degree>
    <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <ControlArray>
      <Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="136">
              <Vertex size="3" type="real">14.60000000 4.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="137">
              <Vertex size="3" type="real">14.60000000 4.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="138">
              <Vertex size="3" type="real">14.60000000 5.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
```

```
          </ControlPoint>
          <ControlPoint id="139">
            <Vertex size="3" type="real">14.60000000 5.00000000 11.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="140">
            <Vertex size="3" type="real">15.00000000 4.00000000 7.90000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="141">
            <Vertex size="3" type="real">15.00000000 4.00000000 11.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="142">
            <Vertex size="3" type="real">15.00000000 5.00000000 7.90000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="143">
            <Vertex size="3" type="real">15.00000000 5.00000000 11.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="17">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="145">
            <Vertex size="3" type="real">0.00000000 0.00000000 7.90000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="146">
            <Vertex size="3" type="real">0.00000000 0.00000000 11.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="147">
            <Vertex size="3" type="real">0.00000000 1.00000000 7.90000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="148">
            <Vertex size="3" type="real">0.00000000 1.00000000 11.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="149">
            <Vertex size="3" type="real">0.40000000 0.00000000 7.90000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="150">
            <Vertex size="3" type="real">0.40000000 0.00000000 11.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="151">
            <Vertex size="3" type="real">0.40000000 1.00000000 7.90000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="152">
            <Vertex size="3" type="real">0.40000000 1.00000000 11.40000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="18">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
```

```
    <ControlArray>
      <Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="154">
              <Vertex size="3" type="real">14.60000000 0.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="155">
              <Vertex size="3" type="real">14.60000000 0.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="156">
              <Vertex size="3" type="real">14.60000000 1.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="157">
              <Vertex size="3" type="real">14.60000000 1.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="158">
              <Vertex size="3" type="real">15.00000000 0.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="159">
              <Vertex size="3" type="real">15.00000000 0.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="160">
              <Vertex size="3" type="real">15.00000000 1.00000000 7.90000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="161">
              <Vertex size="3" type="real">15.00000000 1.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
      </Sequence>
    </ControlArray>
  </NurbsPatch>
  <NurbsPatch id="19">
    <Dimension size="1" type="int">3</Dimension>
    <Degree dim ="1" size="1" type="int">1</Degree>
    <Degree dim ="2" size="1" type="int">1</Degree>
    <Degree dim ="3" size="1" type="int">1</Degree>
    <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
    <ControlArray>
      <Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="163">
              <Vertex size="3" type="real">0.00000000 0.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="164">
              <Vertex size="3" type="real">0.00000000 0.00000000 11.80000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="165">
              <Vertex size="3" type="real">0.00000000 5.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="166">
              <Vertex size="3" type="real">0.00000000 5.00000000 11.80000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
        </Sequence>
        <Sequence>
          <Sequence>
            <ControlPoint id="167">
              <Vertex size="3" type="real">15.00000000 0.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
            <ControlPoint id="168">
              <Vertex size="3" type="real">15.00000000 0.00000000 11.80000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
          </Sequence>
          <Sequence>
            <ControlPoint id="169">
              <Vertex size="3" type="real">15.00000000 5.00000000 11.40000000</Vertex>
              <Weight size="1" type="real">1.00000000</Weight>
            </ControlPoint>
```

```
          <ControlPoint id="170">
            <Vertex size="3" type="real">15.00000000 5.00000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="20">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="172">
            <Vertex size="3" type="real">0.40000000 0.00000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="173">
            <Vertex size="3" type="real">0.40000000 0.00000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="174">
            <Vertex size="3" type="real">0.40000000 0.40000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="175">
            <Vertex size="3" type="real">0.40000000 0.40000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="176">
            <Vertex size="3" type="real">14.60000000 0.00000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="177">
            <Vertex size="3" type="real">14.60000000 0.00000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="178">
            <Vertex size="3" type="real">14.60000000 0.40000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="179">
            <Vertex size="3" type="real">14.60000000 0.40000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="21">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="181">
            <Vertex size="3" type="real">0.40000000 4.60000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="182">
            <Vertex size="3" type="real">0.40000000 4.60000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="183">
            <Vertex size="3" type="real">0.40000000 5.00000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="184">
            <Vertex size="3" type="real">0.40000000 5.00000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
```

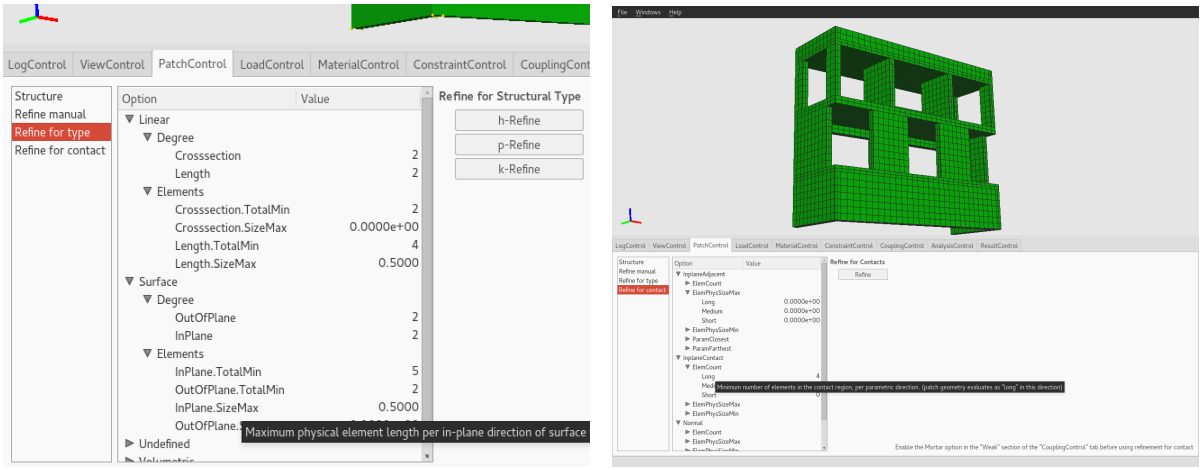
```
      </Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="185">
            <Vertex size="3" type="real">14.60000000 4.60000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="186">
            <Vertex size="3" type="real">14.60000000 4.60000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="187">
            <Vertex size="3" type="real">14.60000000 5.00000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="188">
            <Vertex size="3" type="real">14.60000000 5.00000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="22">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="190">
            <Vertex size="3" type="real">0.00000000 0.00000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="191">
            <Vertex size="3" type="real">0.00000000 0.00000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="192">
            <Vertex size="3" type="real">0.00000000 5.00000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="193">
            <Vertex size="3" type="real">0.00000000 5.00000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="194">
            <Vertex size="3" type="real">0.40000000 0.00000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="195">
            <Vertex size="3" type="real">0.40000000 0.00000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
        <Sequence>
          <ControlPoint id="196">
            <Vertex size="3" type="real">0.40000000 5.00000000 11.80000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
          <ControlPoint id="197">
            <Vertex size="3" type="real">0.40000000 5.00000000 12.60000000</Vertex>
            <Weight size="1" type="real">1.00000000</Weight>
          </ControlPoint>
        </Sequence>
      </Sequence>
    </Sequence>
  </ControlArray>
</NurbsPatch>
<NurbsPatch id="23">
  <Dimension size="1" type="int">3</Dimension>
  <Degree dim ="1" size="1" type="int">1</Degree>
  <Degree dim ="2" size="1" type="int">1</Degree>
  <Degree dim ="3" size="1" type="int">1</Degree>
  <KnotVector dim ="1" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="2" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <KnotVector dim ="3" size="4" type="real">0.00000 0.00000 1.00000 1.00000</KnotVector>
  <ControlArray>
    <Sequence>
      <Sequence>
        <Sequence>
          <ControlPoint id="199">
            <Vertex size="3" type="real">14.60000000 0.00000000 11.80000000</Vertex>
```

```
1339                <Weight size="1" type="real">1.00000000</Weight>
1340              </ControlPoint>
1341              <ControlPoint id="200">
1342                <Vertex size="3" type="real">14.60000000 0.00000000 12.60000000</Vertex>
1343                <Weight size="1" type="real">1.00000000</Weight>
1344              </ControlPoint>
1345            </Sequence>
1346            <Sequence>
1347              <ControlPoint id="201">
1348                <Vertex size="3" type="real">14.60000000 5.00000000 11.80000000</Vertex>
1349                <Weight size="1" type="real">1.00000000</Weight>
1350              </ControlPoint>
1351              <ControlPoint id="202">
1352                <Vertex size="3" type="real">14.60000000 5.00000000 12.60000000</Vertex>
1353                <Weight size="1" type="real">1.00000000</Weight>
1354              </ControlPoint>
1355            </Sequence>
1356          </Sequence>
1357          <Sequence>
1358            <Sequence>
1359              <ControlPoint id="203">
1360                <Vertex size="3" type="real">15.00000000 0.00000000 11.80000000</Vertex>
1361                <Weight size="1" type="real">1.00000000</Weight>
1362              </ControlPoint>
1363              <ControlPoint id="204">
1364                <Vertex size="3" type="real">15.00000000 0.00000000 12.60000000</Vertex>
1365                <Weight size="1" type="real">1.00000000</Weight>
1366              </ControlPoint>
1367            </Sequence>
1368            <Sequence>
1369              <ControlPoint id="205">
1370                <Vertex size="3" type="real">15.00000000 5.00000000 11.80000000</Vertex>
1371                <Weight size="1" type="real">1.00000000</Weight>
1372              </ControlPoint>
1373              <ControlPoint id="206">
1374                <Vertex size="3" type="real">15.00000000 5.00000000 12.60000000</Vertex>
1375                <Weight size="1" type="real">1.00000000</Weight>
1376              </ControlPoint>
1377            </Sequence>
1378          </Sequence>
1379        </Sequence>
1380      </ControlArray>
1381    </NurbsPatch>
1382  </Structure>
```

Appendix C Prototypical implementation of the integrated analysis framework

The concepts and algorithms outlined in the main section were, in the course of preparing this work, combined and complemented to form an autonomous, NURBS-based finite element software. Supplemented by a graphical user interface (GUI), it constitutes the prototypical implementation of an integrated analysis framework as understood by this work.

The entire code is written in C++; it is platform-independent to run on Linux and Windows[1] systems alike. The open-source library Qt,[2] which provides a framework to create platform-independent user interfaces, is used to implement the GUI. The rendering of the structure is realized with the help of the OpenSceneGraph[3] library, which serves as middleware for the visualization of graphical objects with OpenGL.



(a) Settings **(b)** Overview

Figure C.1: Patch tab Provides access to various refinement settings, i.e. manual h-, p-, and k-refinement for individual patches and parametric directions, geometrical type refinement for which the settings are shown in (a), and contact refinement.

The application is centered around a graphic window that provides a 3D rendering of the current structure. The view on that structure is intuitively manipulated with a computer mouse. A structure is solely defined by the NURBS patches representing its geometry. It is loaded from file, which must be given in the XML format defined in appendix B. The GUI application is not meant for the manipulation of the structure's geometric shape, but it does allow to modify the parametrization of the patches. Apart from refining the structure manually or applying geometric type and contact state refinement, the software can be used to apply loads, constraints, material properties, to control the analysis process, to run the solver, and to eventually evaluate the results. Users interact with the software mainly through the various widgets that are organized in tabs and located at the bottom of the main window. Each widget is devoted to a special analysis functionality.

[1] Microsoft Windows 10, Microsoft, www.microsoft.com/de-de/windows
[2] Qt 5.4, The Qt Company Ltd., www.qt.io/qt-5
[3] OpenSceneGraph 3.2.1, www.openscenegraph.org

The functionality of the application is, at least in part, demonstrated by the figures in this appendix.

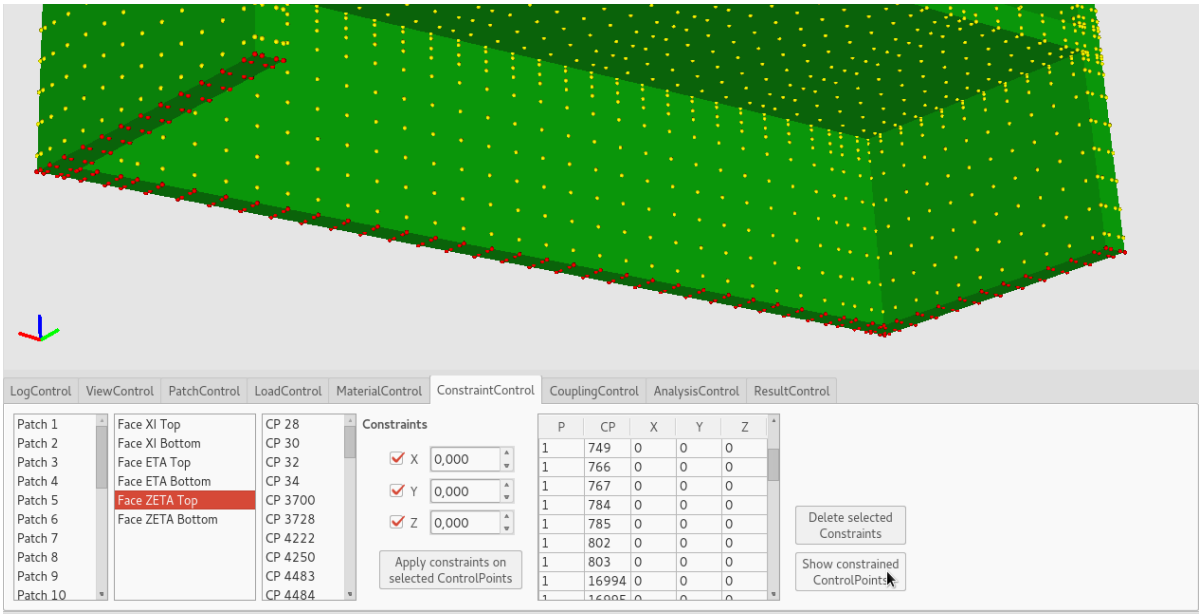
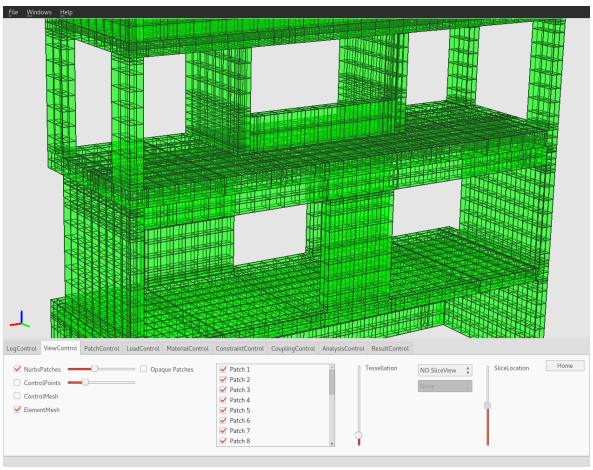
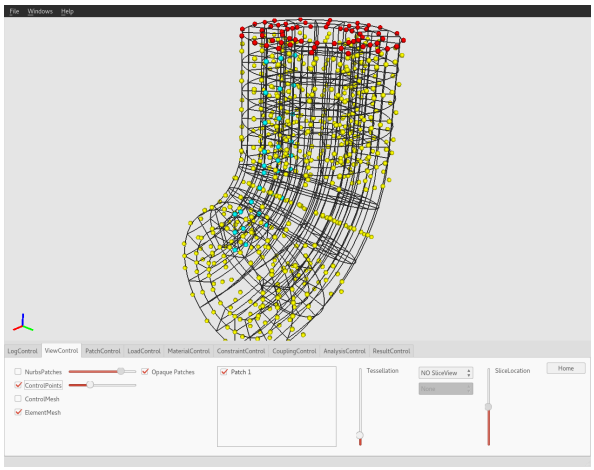


Figure C.2: Constraints tab Allows the application of Dirichlet boundary conditions.



(a) Rendering of patches and element edges, patch surfaces are transparent



(b) Rendering of control points and element edges; active control points are yellow, control points with Dirichlet boundary conditions red, and control points with master-slave constraints cyan

Figure C.3: View tab Provides various options for the visualization, e.g. enabling or disabling the rendering of patches, control points, element edges, and control meshes, complete deactivation of individual patches, control on patch transparency, or the visualization of an isoparametric surface within the volume.

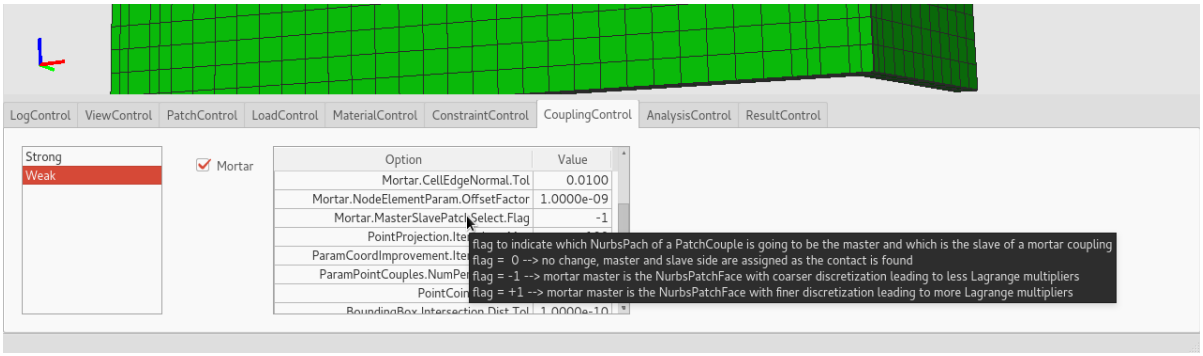
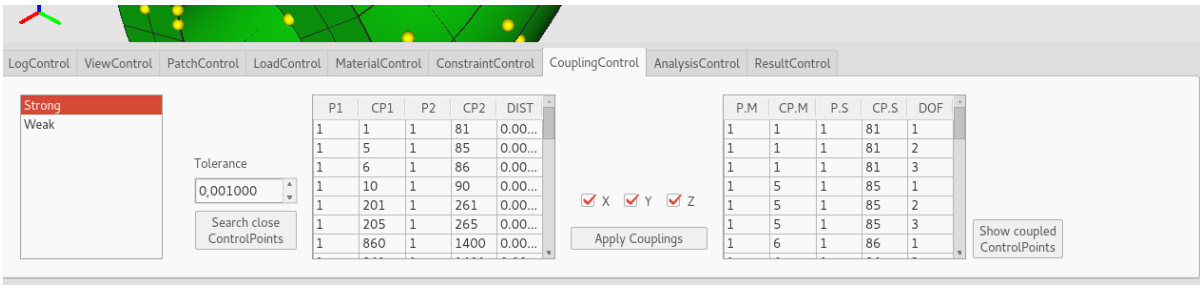
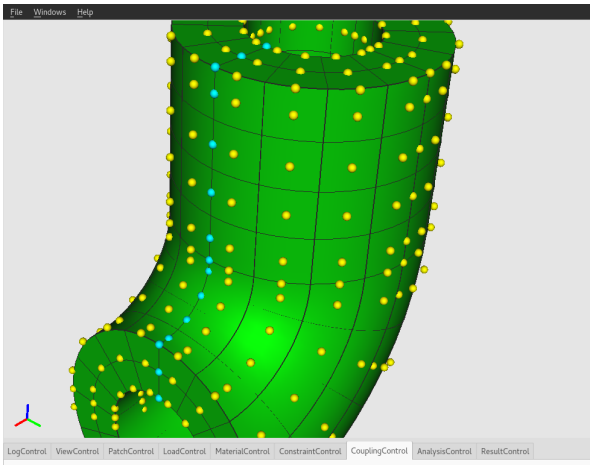


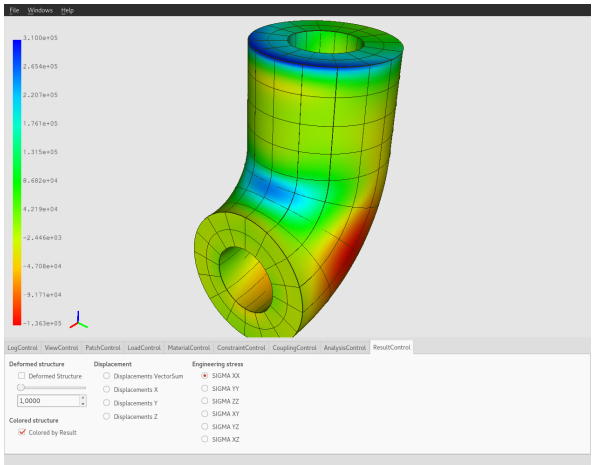
Figure C.4: Coupling tab – weak Activates the weak coupling with the mortar method and provides access to parameters that control the coupling process, e.g. the type of Lagrange multiplier interpolation, the selection of the non-mortar side, the minimum distance for points to be considered coincident, etc.



(a) Strong coupling settings



(b) Control points with master-slave constraints indicated by cyan coloring.



(c) Stress result for a strongly coupled patch. Though the displacement interpolation is only C^0 continuous at the coupling interface, there are no visible stress jumps at that interface.

Figure C.5: Coupling tab – strong Strong coupling is implemented for patches that match geometrically and parametrically at their coupling interface via master-slave constraints. Control points that are to be coupled are identified by their mutual distance. The specification of a respective tolerance value allows to find the control point pairs and to apply the constraints automatically.

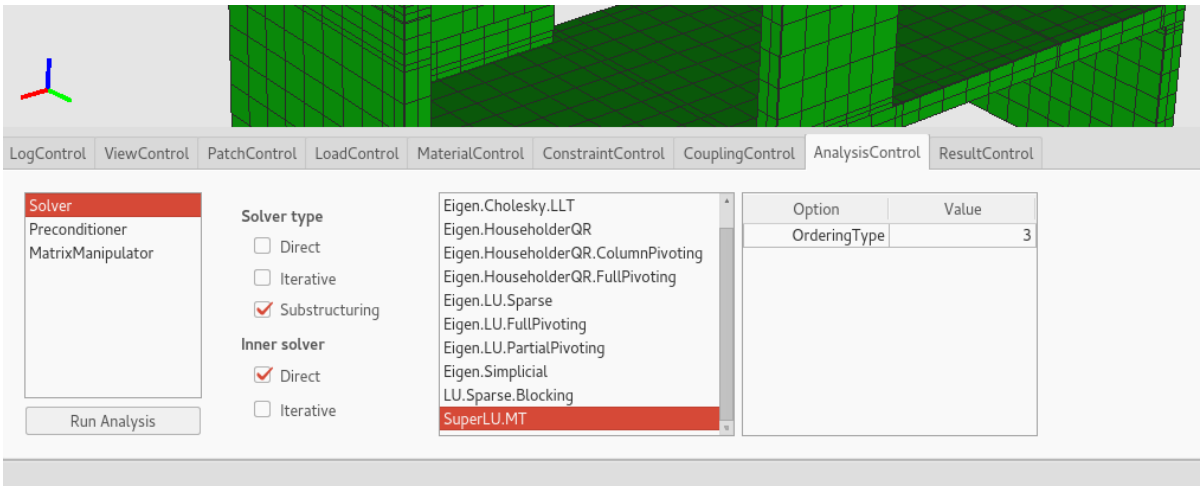
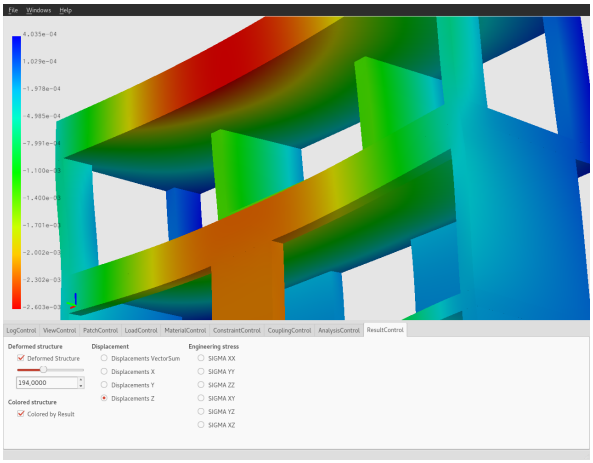
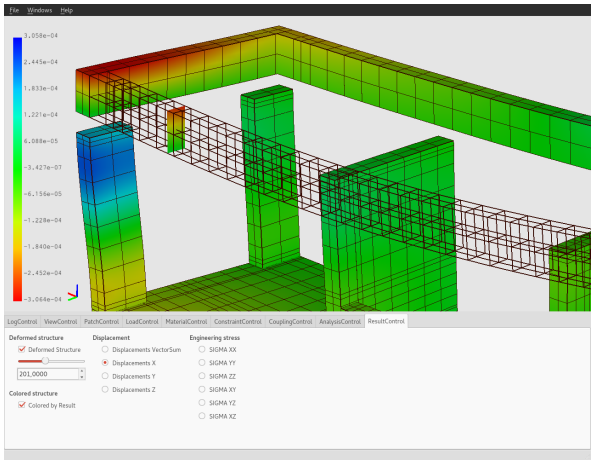


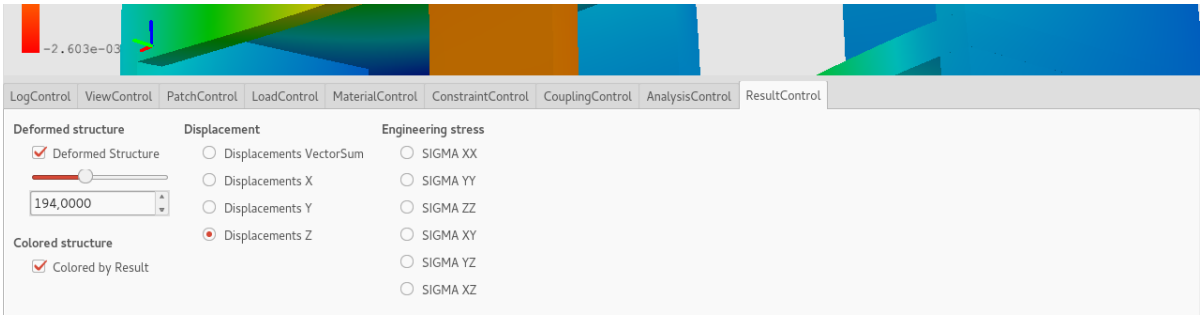
Figure C.6: Analysis tab Provides control over the solution process, i.e. the selection of the solver, the preconditioner, and the matrix manipulator, each with its respective options set.



(a) Displacements - full view



(b) View of an isosurface in the upper front beam while showing all element edges. Rendering of the top slab is disabled.



(c) Settings

Figure C.7: Results tab Allows the visualization of displacement and stress results via color coding as well as the view of the deformed structure at various scales.