Institut für Parallele und Verteilte Systeme
Abteilung Simulation großer Systeme
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2375

Partition of Unity Methode für dreidimensionale Probleme

Michael Lahnert

Studiengang:	Informatik
Prüfer:	Prof. Dr. rer. nat. Marc Alexander Schweitzer
Betreuer:	Dr. rer. nat. Stefan Zimmer
begonnen am:	02. Mai 2012
beendet am:	01. November 2012
CR-Klassifikation:	G.1.8, I.3.5

Kurzfassung

Ziel dieser Studienarbeit ist die Implementierung und Bewertung einer Erweiterung für die institutseigene Partition of Unity Method-Software, die es ermöglicht, die Methode sowohl für zwei- als auch für dreidimensionale Geometrien direkt auf der Computer Aided Design-Geometrie anzuwenden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

Abkürzungsverzeichnis

BRep	Boundary Representation
bspw.	beispielsweise
bzgl.	bezüglich
bzw.	beziehungsweise
CAD	Computer Aided Design
d. h.	das heißt
FEM	Finite Element Method
GFEM	Generalized Finite Element Method
OCC	Open CASCADE
OCE	Open CASCADE Community Edition
PU	Partition of Unity
PUM	Partition of Unity Method
STEP	Standard for The Exchange of Product model data
v. a.	vor allem
vgl.	vergleiche

1. Einleitung

In der heutigen Zeit haben sich Simulationen innerhalb der Produktentwicklung immer weiter etabliert. Das liegt zum einen an der besseren Verfügbarkeit von Rechenleistung, aber auch an der zunehmenden Verbreitung von Software, die Verfahren wie die Finite Elemente Methode (FEM) implementiert. Zum anderen können durch eine geeignete Simulation Kosten für die Herstellung von Prototypen eingespart werden, so dass auch finanzielle Überlegungen eine Rolle spielen.

Für die angesprochene FEM ist die Generierung eines geeigneten Gitters die größte Herausforderung. Da die Produktentwicklung ein iterativer Prozess ist, muss nach jedem Entwicklungsschritt für eine erneute Simulation ein neues Gitter generiert werden. Dazu ist meist menschlicher Eingriff nötig. Insbesondere in größeren Firmen sind Gittergenerierung und Konstruktion in getrennten Abteilungen untergebracht, wodurch sich der Prozess nochmals in die Länge zieht.

Eine deutliche Verbesserung in dieser Problematik brächte daher ein vollautomatischer Prozess, in dem nach jeder Geometrieänderung einfach die veränderte Geometrie in die Simulationsumgebung geladen werden kann.
Ermöglicht wird dies durch erweiterte Finite Elemente Methoden, wie bspw. die Partition of Unity Method (PUM). Der Funktionsumfang der in der betreuenden Abteilung vorhandenen Implementierung war bisher allerdings auf durch Polygone approximierte, zweidimensionale Geometrien und quaderförmige dreidimensionale Geometrien beschränkt.

Das Ziel dieser Arbeit besteht daher in der Erweiterung der Software um zwei Features:

1. Erweiterung der Funktionalität von zwei- auf dreidimensionale Geometrien.
2. Einsparung des Approximierungsschritts durch Rechnen auf der Geometrie.

Zur Präsentation der Ergebnisse ist die Arbeit in folgender Weise aufgebaut: In Kapitel 2 wird überblickhaft erläutert, wie die PUM funktioniert und deren Workflow dem der FEM gegenüber gestellt. Daran anschließend wird in Kapitel 3 erklärt, an welchen Stellen des Verfahrens eingegriffen und was dabei auf welche Weise implementiert wird. Diese Implementierung wird in Kapitel 4 mithilfe von Statistiken und Grafiken bewertet. Abschließend werden in Kapitel 5 die Ergebnisse der Arbeit zusammengefasst und Anknüpfungspunkte vorgestellt.

2. Grundlagen

2.1. Beschreibung Standardworkflow FEM anhand von DUNE

Eine Möglichkeit mit der in der Praxis beliebten FEM zu simulieren, besteht in der Benutzung des DUNE-Softwarepakets[1]. Den Workflow für die Behandlung von CAD-Objekten beschreibt [Bas12, Kapitel 4]:

Nach der Erzeugung der Geometrie wird diese mithilfe eines globalen Gitters diskretisiert. Dazu ist ein externes Programm nötig, DUNE unterstützt an dieser Stelle die Verwendung von Gmsh. Zusatzinformationen für Materialeigenschaften oder Randbedingungen können als Subdomains in Gmsh realisiert werden. Diese Informationen werden auf dem Gitter vermerkt und vom Dune::GmshReader erkannt. Diese Werte müssen anschließend in DUNE spezifiziert werden, da momentan neben der Geometrie nur ein Gitter vorhanden ist, auf dessen Knoten gespeichert ist, zu welcher Subdomain die Knoten gehören - nicht aber was innerhalb der Subdomain gelten soll. Flächen, die zu keiner Subdomain gehören, erhalten Neumann-0 Randbedingungen.

Nun wird mithilfe von lokalen Ansatzfunktionen auf den Gitterzellen sowie den Rand-, Anfangs- und Übergangsbedingungen ein Gleichungssystem definiert, das meist numerisch gelöst wird.

2.2. Beschreibung PUM

Die PUM ist eine Multilevel-Approximations-Methode zur numerischen Lösung partieller Differentialgleichungen, die eine Verallgemeinerung der FEM darstellt. Da die Methode als solche nicht Thema der vorliegenden Arbeit ist, beschränken sich die Ausführungen an dieser Stelle auf zum weiteren Verständnis notwendige Grundkonzepte und Definitionen. Eine vollständige Beschreibung des Verfahrens findet sich bspw. bei [Sch03], [Sch05] oder [Sch08].

Die in diesem Abschnitt folgenden Erläuterungen und Formeln stammen alle aus [Sch05, Kapitel 2 und 3].

[1] http://www.dune-project.org/

Ganz allgemein ist das Ziel der PUM die näherungsweise Lösung eines bspw. elliptischen Randwertproblems der folgenden Art:

$$\begin{aligned} Lu &= f \quad \text{auf } \Omega \subset \mathbb{R}^d \\ Bu &= g \quad \text{auf } \partial\Omega \end{aligned}$$

Dabei steht L für einen elliptischen Differentialoperator und B für geeignete Randbedingungen. Ω ist das Gebiet. Dieses wird mit einem Cover C_Ω überdeckt. Das Cover besteht aus rechteckigen Patches $\omega_i \subset \mathbb{R}^d$. Das Integrationsgebiet $\overline{\Omega}$ entspricht der Vereinigung des Gebiets Ω mit dessen Rand $\partial\Omega$. $\mathbb{R}^d$ steht für den Raum, in dem die Geometrie liegt und d ist dessen Dimension, konkret also $d = 2$ oder $d = 3$.

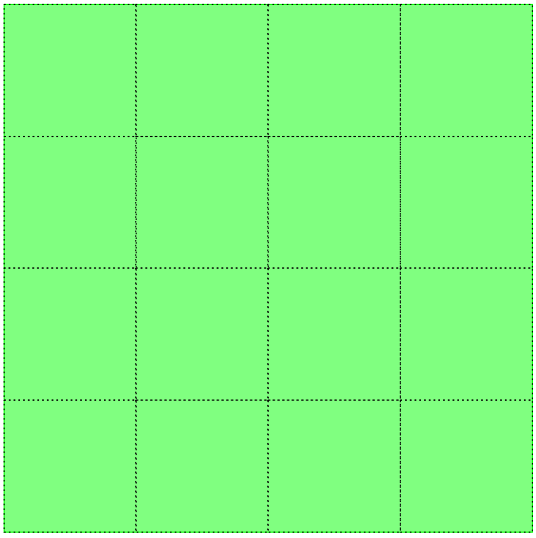

Abbildung 2.1.: Reguläre Zerlegung eines Quadrats, Level 2

Obige Abbildung zeigt exemplarisch eine reguläre Zerlegung auf Level 2 für ein Quadrat. Formal lässt sich der Aufbau des Covers wie folgt ausdrücken:

$$\begin{pmatrix} \{x_i\} \\ \mathcal{W} \\ \{p_i\} \end{pmatrix} \rightarrow \begin{pmatrix} \{\omega_i\} \\ \{W_i\} \\ \{V_i^{p_i} = \text{span}\langle \psi_i^n \rangle\} \end{pmatrix} \rightarrow \begin{pmatrix} \{\varphi_i\} \\ \{V_i^{p_i}\} \end{pmatrix} \rightarrow V^{\text{PU}} = \sum \varphi_i V_i^{p_i} \tag{2.1}$$

Das bedeutet, dass einer Punktmenge $\{x_i\}$, einer allgemeinen Gewichtsfunktion $\mathcal{W}$ und der Ordnung des lokalen Approximationsgebiets zunächst Patches ω_i, lokale Gewichtsfunktionen W_i und lokale Approximationsräume $V_i^{p_i}$ zugeordnet werden. Aus Patches und Gewichtsfunktionen errechnet man lokale PU-Funktionen $\{\varphi_i\}$. Aus PU-Funktionen und Approximationsräumen entsteht der „Trial- and Testspace“. Das ist der Raum, der in einer Galerkin-Formulierung verwendet wird, um die Diskretisierung durchzuführen.

$$V^{PU} := \sum_i \varphi_i V_i^{p_i} = \sum_i \varphi_i \,\text{span}\langle \{\psi_i^n\} \rangle = \text{span}\langle \{\varphi_i \psi_i^n\} \rangle$$

Zu beachten ist an dieser Stelle, dass die Patches nicht willkürlich und mit beliebiger Größe über die Geometrie verteilt werden. Vielmehr werden die Patches mithilfe eines regelmäßigen rechteckigen Gitters definiert[2], mit dessen Hilfe ein Cover $C_\Omega := \{\omega_i\}$ gebildet wird. Die Maschenweite des Gitters wird dabei zu $2h$ definiert. Damit ergibt sich für die Gitterzellen auf einem quadratischen oder kubischen Bereich im Intervall (-1, 1):

$$\mathcal{C}_i = \prod_{l=1}^{d} (c_i^l - h, c_i^l + h)$$

Damit werden die Patches ω_i definiert zu

$$\omega_i := \prod_{l=1}^{d} (c_i^l - \alpha h, c_i^l + \alpha h), \qquad \text{mit } \alpha > 1$$

Um daraus eine Partition der Eins auf dem Cover C_Ω mit $N := \text{card}(C_\Omega)$ zu erhalten, wird für jeden Patch ω_i eine Gewichtsfunktion $W_i : \Omega \mapsto \mathbb{R}$ mit $\text{supp}(W_i) = \omega_i$ wie folgt definiert:

$$W_i(x) = \begin{cases} \mathcal{W} \circ T_i(x) & x \in \omega_i \\ 0 & \text{sonst} \end{cases}$$

Dabei steht T_i für die affine Transformation $T_i : \overline{\omega_i} \mapsto [-1,1]^d$ und $\mathcal{W} : [-1,1]^d \mapsto \mathbb{R}$ für die allgemeine Gewichtsfunktion. Durch Mitteln über die Gewichtsfunktionen entstehen die PU-Funktionen:

$$\varphi_i := \frac{W_i(x)}{S(x)} \text{ mit } S(x) := \sum_{l=1}^{N} W_l(x) = \sum_{\{l:\omega_l \cap \omega_i \neq \emptyset\}} W_l(x)$$

Durch diese Konstruktion ist die Partition der Eins nicht-negativ, falls die Gewichtsfunktionen nicht-negativ sind. Außerdem erfüllen die PU-Funktionen φ_i die sogenannte Flat-Top-Eigenschaft für $\alpha \in (1,2)$. Durch diese Konstruktion kann die Überlappung der Patches durch den Parameter $\alpha \in (1,2)$ leicht kontrolliert werden. Diese Konstruktion ist ohne großen Aufwand auf größere und anisotrope Gebiete erweiterbar und bietet zudem den Vorteil, dass im Vergleich zu einer zufälligen Verteilung der Patches weniger Integrationszellen entstehen.

Die PUM gehört zur Klasse der gitterfreien Generalized Finite Element Methods (GFEM). Das bedeutet, dass eine partielle Differentialgleichung unter Benutzung der schwachen Formulierung der Ableitung und einem Galerkinansatz diskretisiert wird. Die schwache Formulierung, die in [Sch05, Gleichung (2.14)] verwendet wird, sieht dabei wie folgt aus:

$$\begin{aligned} a(u,v) &= l(v) \\ \text{mit } a(u,v) &= \int_\Omega \nabla u \nabla v + \int_{\Gamma_D} u(\beta v - v_\nu) - u_\nu v \\ \text{und } l(v) &= \int_\Omega f v + \int_{\Gamma_D} g_D(\beta v - v_\nu) + \int_{\Gamma_D} g_N v \end{aligned}$$

[2]Diese Darstellung dient nur der Vereinfachung. In [Sch05, Abschnitt 3.2] ist die Konstruktion als Quad-/Octtree definiert.

Zu berechnen sind die Einträge der Steifigkeitsmatrix und die Einträge des Vektors auf der rechten Seite.

$$\begin{aligned} A &= \left(A_{(i,n),(j,m)}\right) \text{ mit } A_{(i,n),(j,m)} = a\left(\varphi_j\psi_j^m, \varphi_i\psi_i^n\right) \\ \hat{f} &= \left(\hat{f}_{(i,n)}\right) \text{ mit } \hat{f}_{(i,n)} = l(\varphi_i, \psi_i^n) \end{aligned}$$

Die zugehörigen Ergebnisse finden sich als Gleichung (2.16) und (2.17) in [Sch05, Abschnitt 2.3]. Aufgrund der in Gleichung (2.1) skizzierten Konstruktion lässt sich der stückweise Charakter der Integranten auflösen.

Durch Unterteilung der Integrationsgebiete $\omega_{ij} := \omega_i \cap \omega_j \cap \Omega$ mithilfe der geometrischen Elemente des offenen Covers C_Ω erhält man für die GFEM stückweise glatte Integranten auf den Integrationszellen. Bisher existieren allerdings nur Patches und deren Gewichtsfunktionen, die gemeinsam die lokalen PU-Funktionen definieren. Eine geeignete Zerlegung $D_{\omega_{ij}}$ der Integrationsgebiete ω_{ij} in disjunkte Integrationszellen $D^s_{\omega_{ij}}$ kann durch Ausnutzung der Produktstruktur der Patches und Gewichtsfunktionen erreicht werden. Die entstehende Zerlegung $D_{\omega_{ij}} := \{D^s_{\omega_{ij}}\}$ ist dahingehend optimal, dass alle Stetigkeitsunterbrechungen in der Ableitung der PU-Funktionen mit minimaler Zellzahl aufgelöst sind. Somit kann eine Quadraturregel höherer Ordnung erfolgreich angewendet werden.

Bei konkreter Betrachtung einer ersten Zerlegung $E_{\omega_{ij}} := \{E^s_{\omega_{ij}}\}$ eines Integrationsgebiets ω_{ij} nach diesem Verfahren mit $\omega_{ij} = \omega_i \cap \omega_j \cap \Omega$ fällt auf, dass die zugehörigen Gewichtsfunktionen $W_i|_{E^s_{\omega_{ij}}}$ und $W_j|_{E^s_{\omega_{ij}}}$ zwar Polynome von Grad l sind, die Gewichtsfunktionen $W_k|_{E^s_{\omega_{ij}}}$ aller übrigen Nachbarn $\omega_k \in C_{ij} := C_i \cap C_j$ aber nach wie vor möglicherweise nur stückweise polynomiell sind. Folglich muss die Zerlegung mithilfe der ω_k^q Teilpatches aller Patches ω_k weiter verfeinert werden. Die resultierende Zerlegung $D_{\omega_{ij}} = \{D^s_{\omega_{ij}}\}$ besteht aus rechteckigen Zellen $D^s_{\omega_{ij}}$ auf denen alle Gewichtsfunktionen $W_k|_{D^s_{\omega_{ij}}}$ Polynome vom Grad l sind.

In Abbildung 2.2 sind die so entstandenen Integrationszellen für einen Überlappungsparameter $\alpha = 1.5$ dargestellt: Grün hinterlegt ist der Bereich Ω, ein Patch ω_i ist exemplarisch rot umrandet, ebenso exemplarisch in blau eine Coverzelle. Die schwarz umrandeten Vierecke stellen alle entstandenen Integrationszellen dar.

2.3. Workflow PUM und dessen Vorteile gegenüber FEM

Mit der im obigen Abschnitt beschriebenen PUM ergibt sich folgender Workflow:

Wie bei der FEM muss auch bei der PUM die Geometrie konstruiert oder eingelesen sowie zusätzliche Informationen wie bspw. Randbedingungen definiert werden. Im Gegensatz zur FEM wird nun allerdings kein externes Programm zur Gittergenerierung aufgerufen. Stattdessen werden alle inneren Integrationszellen durch die Zerlegung des von der Geometrie definierten Bereichs Ω in sich überlappende Patches ω_i erzeugt. Zur Vereinfachung der Implementierung wird lokal für Bereiche, in denen die Patches den Rand des Bereichs

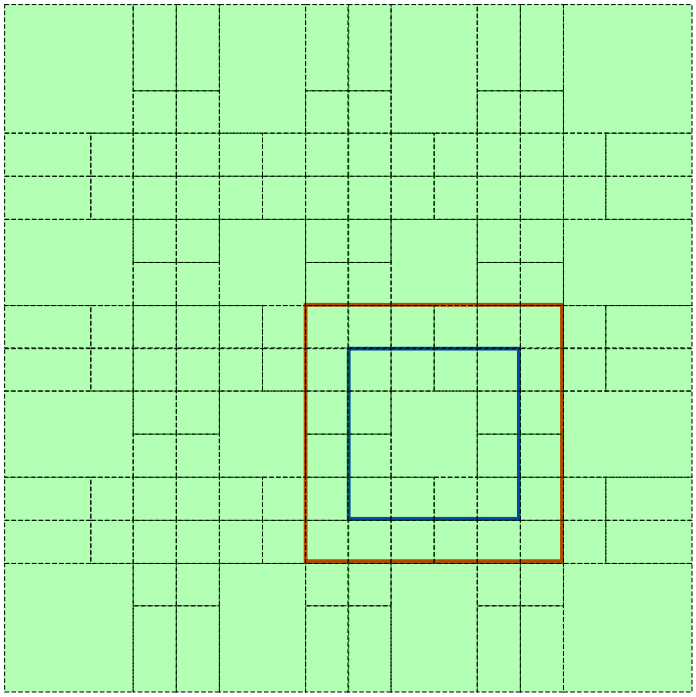

Abbildung 2.2.: Reguläres Cover eines Quadrats, Level 2

schneiden, in denen also gilt, dass $\partial\Omega \cap \omega_i \neq \emptyset$, ein lokaler Gittergenerator direkt aus der Simulationsumgebung heraus aufgerufen. Dabei entsteht kein global konsistentes Gitter.

Anschließend muss die Berechnung durchgeführt werden.

Positiv gegenüber der FEM ist bei diesem Ablauf v. a. zweierlei:
Erstens sind die Zerlegungsschritte ebenso wie die Aufrufe des Gittergenerators auf jedem Level vollständig unabhängig voneinander. Somit ist es ohne großen Aufwand möglich, die Integrationszellen parallel zu erzeugen.
Zweitens wird größtenteils direkt auf dem von der Geometrie definierten Bereich gerechnet und der Gittergenerator nur für lokal begrenzte Teilgebiete aufgerufen und dabei unmittelbar aus der Simulationsumgebung heraus kontrolliert. Das führt dazu, dass bei Geometrieänderungen nicht erst von außerhalb der Simulationsumgebung ein neues Gitter generiert werden muss, sondern stattdessen nach einer eventuellen Anpassung der Randbedingungen einfach die neue Geometrie geladen werden kann. Die vollständige Kontrolle des Gittergenerators aus der Simulationsumgebung in Verbindung mit dem Konzept der lokalen Aufrufe bringt außerdem den Vorteil, kritische Bereiche des Bauteils höher auflösen zu können.

3. Beschreibung der Implementierung

3.1. Implementierte Funktionalität

3.1.1. Allgemeines

Jegliche Funktionalität wird mithilfe der Programmiersprache C++ und des CMake Buildsystems[1] implementiert. Die Wahl begründet sich einerseits mit der Tatsache, dass alle verwendeten Frameworks in C++ implementiert sind, andererseits damit, dass CMake das Standardbuildsystem der betreuenden Abteilung ist. Ein weiteres Argument ist in der Einfachheit des CMake Systems zu sehen.

Der zeitliche Ablauf der Implementierung gestaltet sich wie folgt: Zunächst wird als Proof of concept eine Standalone Anwendung realisiert, deren Zweck darin besteht, herauszufinden, ob es überhaupt mit vertretbarem zeitlichen Aufwand möglich ist, direkt auf einer dreidimensionalen CAD-Geometrie eine Zerlegung nach dem in Kapitel 2.2 beschriebenen Verfahren zu berechnen.

Dabei entsteht der folgende Programmablauf:

1. Einlesen des Dateinamens der CAD-Datei und der gewünschten Iterationstiefe als Kommandozeilenparameter.
2. Auslesen der spezifizierten CAD-Datei. Die darin enthaltene Geometrie definiert das Gebiet Ω. Unterstützt wird das Einlesen von .brep und .stp-Dateien.
3. Berechnung der Bounding Box von Ω.
4. Zweidimensionale Geometrien, die nicht in der xy-Ebene liegen, werden zunächst in eine Parallelebene gedreht und anschließend in die xy-Ebene verschoben.
5. Konstruktion einer Vergleichsgeometrie R_L. Diese ist äquivalent zu einer Zerlegung des Gebiets mit Patches ohne Überlappung, d.h. $\alpha = 1$.
6. Test, wo die Vergleichsgeometrie bezüglich der CAD-Geometrie liegt: $\Omega \cap R_L$
7. Falls die Vergleichsgeometrie R_L den Rand des Bereichs schneidet, d.h. $\partial\Omega \cap R_L \neq \emptyset$, und der maximale Verfeinerungslevel noch nicht erreicht ist, wird das Gebiet $\Omega \cap R_L$ so zerlegt, dass alle Abmaße von R_L halbiert werden. Anschließend wird die Lage der so entstandenen Teilgebiete $\{R_{L+1}\}$ im Verhältnis zu Ω erneut überprüft.

[1] http://www.cmake.org/

8. Dabei entsteht implizit eine Baumstruktur, im zweidimensionalen Fall ein Quadtree, im dreidimensionalen Fall ein Octtree, wie man in den Abbildungen 3.1 und 3.2 sieht. Zur Erhöhung der Effizienz wird diese Baumstruktur implementiert und an jedem Knoten, der einem Teil des Rands zugeordnet ist, an dem also gilt, dass $R_L \cap \partial\Omega \neq \emptyset$, die entsprechende Teilgeometrie $\Omega \cap R_L$ gespeichert.

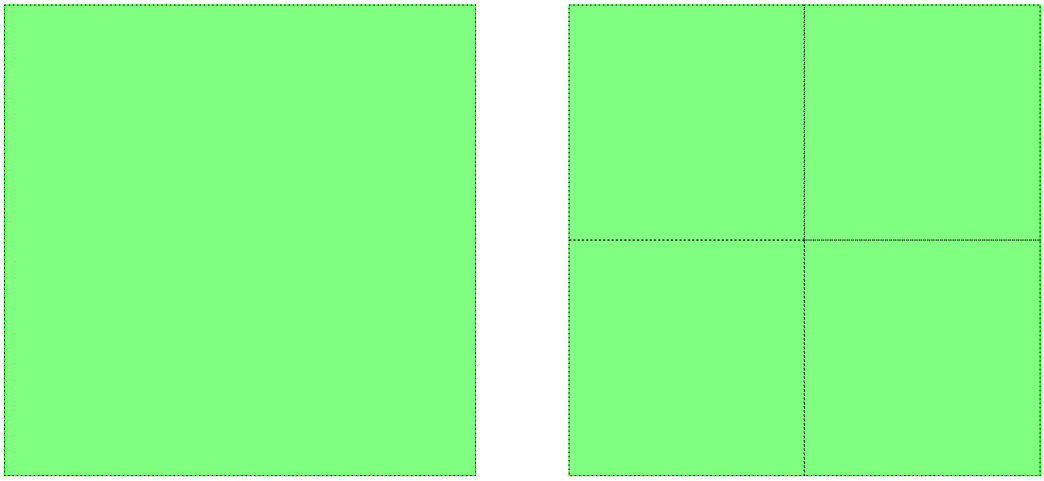

Abbildung 3.1.: Zerlegung eines 2D-Teilgebiets

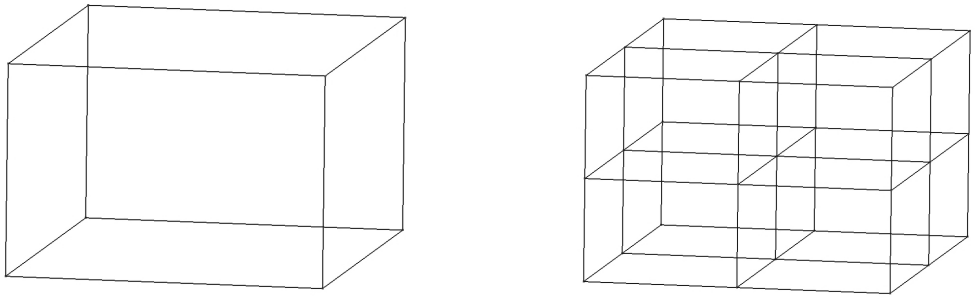

Abbildung 3.2.: Zerlegung eines 3D-Teilgebiets

Im Anschluss daran wird diese Funktionalität in einer Bibliothek gekapselt. Zusätzlich wird eine Zugriffsfunktion auf Baumelemente implementiert, um für Teile der im Baum gespeicherten Geometrien Integrationszellen generieren zu können. Dabei wird für einen Teilpatch $t \subset \omega_i$ bestimmt, ob dieser die Geometrie schneidet. Falls das der Fall ist, wird für $t \cap \overline{\Omega}$ ein externer Gittergenerator aktiviert und das erhaltene Gitter zurückgegeben. Dass es sich im zweidimensionalen Fall bezahlt macht, Teilgeometrien zu speichern um ähnlich große Geometrien miteinander vergleichen zu können, im dreidimensionalen jedoch nicht, zeigt Kapitel 4.2.1.

Schließlich wird die Bibliothek in die abteilungseigene PUM-Software integriert, so dass diese mithilfe der Funktionalität aus der Bibliothek die PUM für zwei- und dreidimensionale

Geometrien direkt auf der CAD-Geometrie anwenden kann. Die dazu notwendigen Schritte werden in Kapitel 3.2.1 erläutert.

Nachfolgend werden die beiden zentralen Schritte, nämlich die Bestimmung der Position der Vergleichsgeometrie R_L bezüglich des Bereichs Ω und die lokale Gittergenerierung, näher beschrieben.

3.1.2. Erkennung von inneren Zellen und Randzellen

Für die Position einer PUM-Zelle C_i bezüglich der untersuchten Geometrie gibt es genau drei Möglichkeiten: Die betrachtete Zelle liegt entweder vollständig ($C_i \subset \overline{\Omega}$) oder teilweise innerhalb ($C_i \cap \overline{\Omega} \neq \emptyset$) oder vollständig außerhalb ($C_i \cap \overline{\Omega} = \emptyset$) des Gebiets. Offensichtlich trifft immer genau eine Eigenschaft zu, weshalb es genügt, zwei der drei Eigenschaften zu prüfen.

Wie in Abbildung 3.1 dargestellt, entstehen bei der Verfeinerung der Auflösung im zweidimensionalen Fall vier rechteckige Zellen R_{L+1} von gleicher Größe. Für dreidimensionale Geometrien gilt nach dem gleichen Gliederungsprinzip, dass aus einem mit einer Baumzelle verknüpften Quader acht Teilquader R_{L+1} mit gleichen Abmaßen entstehen, siehe Abbildung 3.2.

Daher kann die Zerlegung so implementiert werden, dass eine einzelne kleinere Zelle erzeugt wird, die dann drei bzw. sieben mal innerhalb der ursprünglichen Zelle verschoben wird.

Der Test als solches wird als Schnittmengenoperation zwischen ursprünglichem Gebiet Ω und der Zelle R_L realisiert.

Für innere Zellen wird der entsprechende Teil des Bereichs Ω aus der Vergleichsgeometrie durch Differenzmengenbildung herausgeschnitten. Falls das Ergebnis eine leere Geometrie ist, ist die Vergleichsgeometrie vollständig enthalten: $R_L - \overline{\Omega} \stackrel{?}{=} \emptyset$.

Listing 3.1 Funktion zur Erkennung innerer Zellen

```
bool isInnerCell ( TopoDS_Shape innerShape, TopoDS_Shape outerShape )
{
    TopoDS_Shape res = BRepAlgoAPI_Cut ( outerShape, innerShape );
    GModel mod;
    mod.importOCCShape ( ( void* ) &res );
    return mod.empty();
}
```

Zur Erkennung von Randzellen muss unter der Bedingung, dass die Information, ob eine Zelle eine innere Zelle ist oder nicht, bereits vorliegt, nur noch geprüft werden, ob das Ergebnis der Schnittmengenoperation $\Omega \cap R_L$ eine nicht-leere Geometrie ist.

Listing 3.2 Funktion zur Erkennung von Randzellen

```
bool isBoundaryCell ( GModel& mod )
{
    return !mod.empty();
}
```

3.1.3. Lokale Gittergenerierung

Die Funktion zur lokalen Gittergenerierung wird für Teilpatches t aufgerufen, die den Rand der Geometrie schneiden; $t \cap \partial\Omega \neq \emptyset$ mit $t \subset \omega_i$. Dies wird mit dem im vorangegangenen Abschnitt beschriebenen Ablauf zur Bestimmung von Randzellen entschieden. Falls t eine Randzelle bildet, extrahiert die Schnittmengenoperation $t \cap \Omega$ den innenliegenden Teil des Teilpatches t.

Die aus der Schnittmengenoperation entstandene Open CASCADE-Shape (OCC) wird dann in eine Gmsh-Geometrie importiert, für die ein Gitter erzeugt wird.

Anschließend müssen noch die Gitterpunkte gefunden werden, die den Rand der Originalgeometrie approximieren. Dafür gibt es zwei Möglichkeiten:

Die erste Möglichkeit besteht darin, den Rand $\partial\Omega \cap t$ vor der Gittergenerierung mit der Operation `BRepAlgoAPI_Section` zu identifizieren und dann sowohl ein Gitter für t als auch ein Gitter für $\partial\Omega \cap t$ zu generieren.

Alternativ kann man nur ein Gitter für den Teilpatch t erzeugen und in diesem die Gitterpunkte suchen, die den Rand des Bereichs $\partial\Omega$ approximieren. Diese müssen explizit gesucht werden, weil das Ergebnis der Schnittmengenoperation $t \cap \Omega$ zum Finden des innenliegenden Teils intern wie eine eigene Geometrie behandelt wird; nicht etwa als eine Teilshape der Originalgeometrie. Daher sind alle Kanten gleichermaßen „Rand“. Dieses Problem wird gelöst, indem der außerhalb von $\overline{\Omega}$ liegende Teilbereich des Teilpatches betrachtet wird, $t - (\Omega \cap t)$. Durch die Bestimmung des Abstands für jeden Gitterpunkt mittels der OCC-Funktion `BRepExtrema_DistShapeShape` zum außen liegenden Teilstück $t - (\Omega \cap t)$ lassen sich die Randpunkte leicht erkennen. Wenn der Gitterpunkt direkt auf bzw. innerhalb eines schmalen ε-Schlauchs liegt, wird er als Randpunkt erkannt. Dieser Ansatz ist wegen der im Umgang mit CAD-Modellen auftretenden Toleranzen nötig.

Zur Illustration des Prozesses dienen die Abbildungen 3.3 auf der nächsten Seite und 3.4 auf Seite 25.
Dargestellt ist in beiden Fällen oben links der durch die Originalgeometrie definierte Bereich Ω, gemeinsam mit einem Patch ω_i der den Rand schneidet; $\partial\Omega \cap \omega_i \neq \emptyset$. Im zweidimensionalen Fall wird die Situation oben rechts - um 90° gedreht - noch einmal detaillierter dargestellt.

Im nächsten Schritt, im dreidimensionalen Fall oben rechts dargestellt, wird der innenliegende Teil $\Omega \cap \omega_i$ erkannt.

Für diesen Teil des Patches ω_i wird mithilfe des Gittergenerators ein Gitter erstellt, unten links dargestellt. Das untere rechte Bild zeigt eine Detailansicht des erzeugten Gitters, erneut jeweils gedreht dargestellt.

Abbildung 3.3.: Lokale Gittergenerierung 2D: Oben links dargestellt sind die Eingangsdaten des Algorithmus: ω_i und Ω. Oben rechts eine detailliertere Darstellung, um 90° gedreht. Die anschließende Erkennung des innenliegenden Teils $\Omega \cap \omega_i$ mit Gitter befindet sich unten links, ebenfalls um 90° gedreht. Unten rechts ist das erzeugte Gitter als Detailansicht dargestellt.

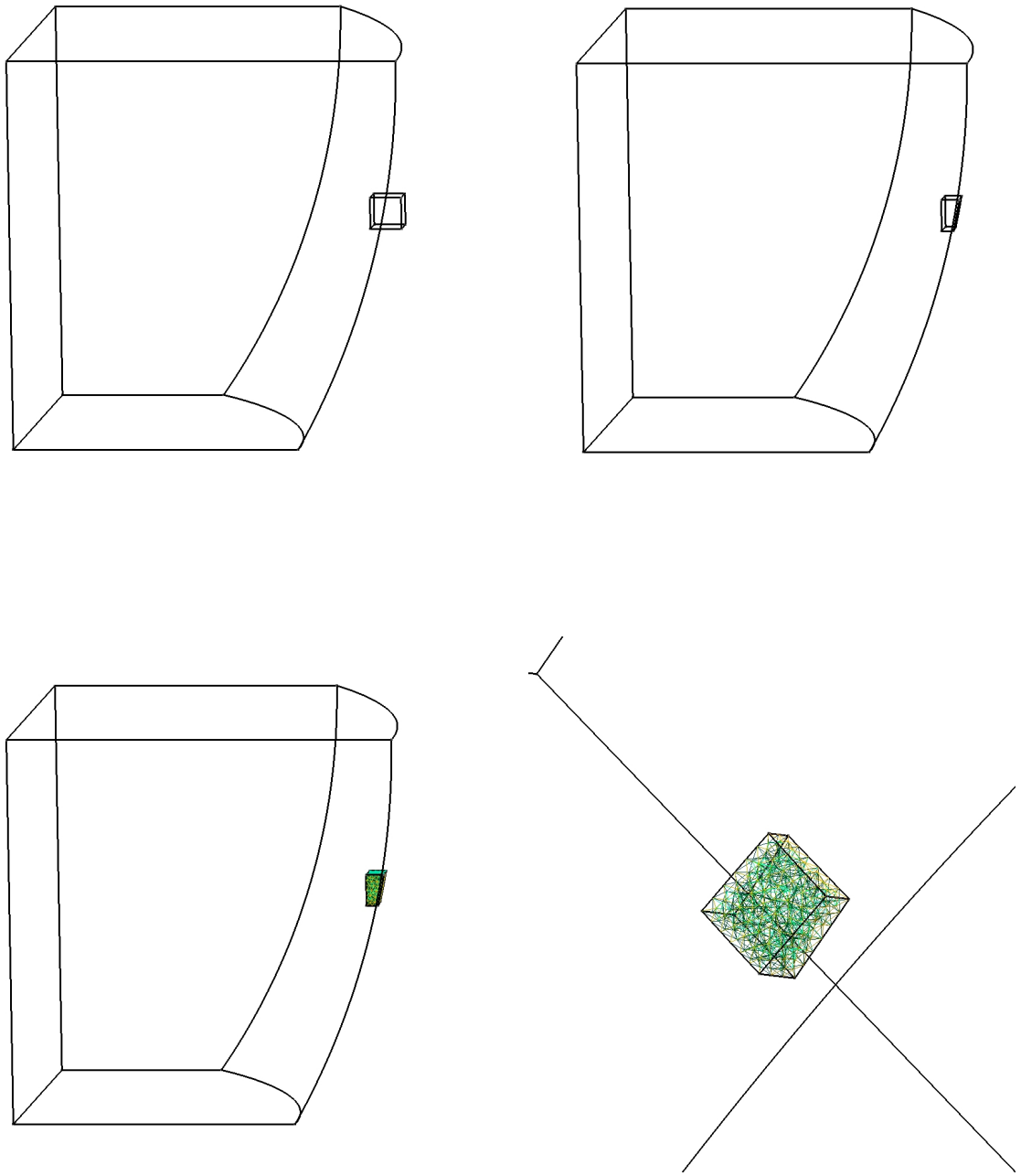

Abbildung 3.4.: Lokale Gittergenerierung 3d: Oben links dargestellt sind die Eingangsdaten des Algorithmus: ω_i und Ω. Oben rechts sieht man die Erkennung des innenliegenden Patchteils $\Omega \cap \omega_i$. Für diesen Teil wurde ein Gitter erstellt, unten links. Dieses ist unten rechts nochmals in einer Detailansicht dargestellt (gedreht).

3.2. Verwendete Frameworks

3.2.1. Abteilungseigene PUM-Software

Zur Integration in die abteilungseigene PUM-Software ist es zum einen nötig, die durch die Umstellung von 2D auf 3D notwendig gewordenen neuen Integrationszellen zu implementieren und zum anderen müssen bestehende Template-Funktionen für OCC-Geometrieobjekte spezifiziert werden.

Für dreidimensionale Geometrien entstehen neben Quadern als innere Volumenintegratonszellen durch den Einsatz von Gmsh Dreiecke als Oberflächenintegrationszellen und Tetraeder als Volumenintegrationszellen. Diese Objekte müssen mitsamt Quadraturregeln erzeugt werden.

Der Integrationsprozess ist noch nicht vollständig abgeschlossen. Konkret verbleibt noch folgendes zu tun:

- Vollständiges Implementieren und Testen der Integrationszellen für dreidimensionale Geometrien.
- Integration der Baumstruktur zur Speicherung von Teilgeometrien für zweidimensionale Geometrien.

Zu spezifizieren sind die folgenden drei Template-Funktionen:

1. **Schnittoperationen** $\omega_i \overset{?}{\subset} \Omega$ und $\Omega \cap \omega_i \overset{?}{\neq} \emptyset$

   Gegeben ist bei dieser Funktion das Gebiet Ω und ein Patch ω_i, dessen Lage zu prüfen ist. Dafür wird die interne Struktur zur Speicherung von Patches in eine OCC-Shape überführt. Anschließend wird die Schnittmengenoperation $\Omega \cap \omega_i$ durchgeführt und das Ergebnis anhand der in Abschnitt 3.1.2 beschriebenen Methode ermittelt.

2. **Schnittoperationen** $\Omega \cap \omega_i$ und $\partial\Omega \cap \omega_i$

   Gegeben ist bei dieser Funktion, die nur für die Patches aufgerufen wird, für die gilt $(\Omega \cap \omega_i \neq \emptyset) \wedge (\omega_i \nsubseteq \Omega)$, wie in der oben beschriebenen Funktion das Gebiet Ω und der zu untersuchende Patch ω_i. Auch hier muss der Patch zunächst in einer OCC-Shape überführt werden, um anschließend mittels der in [Tec12b, Abschnitt 5] spezifizierten Funktionen `BRepAlgoAPI_Common` für die Funktion $\Omega \cap \omega_i$ und `BRepAlgoAPI_Section` für die Funktion $\partial\Omega \cap \omega_i$ das gewünschte Ergebnis in Form einer OCC-Shape zurückzugeben.

3. **Lokale Generation von Integrationszellen**

   Gegeben sind die Ergebnisse der in der vorangegangenen Funktion beschriebenen Schnittoperationsfunktionen. Die OCC-Shapes werden jeweils zunächst in eine Gmsh-Geometrie überführt und anschließend für diese Geometrien ein Gitter generiert. Anschließend wird das erzeugte Gitter elementweise exploriert, um daraus Integrationszellen zu bauen.

Dabei können 4 Fälle auftreten: Im zwei- wie im dreidimensionalen Fall können sowohl Rand- als auch innere Zellen erzeugt werden. Im zweidimensionalen Fall werden als innere Zellen Dreiecke und als Randzellen Liniensegmente erzeugt, vgl. Abbildung 3.3 unten rechts, während im dreidimensionalen Fall als innere Zellen Tetraeder und als Randzellen Dreiecke erzeugt werden, vgl. Abbildung 3.4.

Die Funktionalität der Integration ist für den zweidimensionalen Fall am Beispiel der Wärmeverteilung innerhalb eines Zahnrads, in Abbildung 3.5 auf der nächsten Seite dargestellt, dokumentiert. Für den dreidimensionalen Fall zeigt das Beispiel der Zerlegung einer Schraubenmutter, dargestellt in Abbildung 3.6 auf Seite 29, die erzeugten Integrationszellen. Weitere mit der abteilungseigenen PUM-Software erstellte Beispielabbildungen finden sich im Anhang.

3.2.2. Open CASCADE Technology

Open CASCADE[2] ist eine quelloffene CAD-Engine, die in der vorliegenden Arbeit für fast alle Geometrieoperationen verwendet wird. Neben der von Open CASCADE Technology vertriebenen „offiziellen“ Version existiert eine von Thomas Paviot begründete Version namens Open CASCADE Community Edition (OCE)[3], die in erster Linie dazu dienen soll, Bugs in OCC schneller zu fixen. Da OCE mit dem CMake Buildsystem arbeitet, wird zunächst mit OCE entwickelt, die Bibliothek dann zwecks Kompatibilität zur abteilungseigenen PUM-Software auf OCC umgestellt.

Die verwendeten Geometrieoperationen umfassen bspw. die Konstruktion und das Einlesen der notwendigen Geometrien. In [Tec12a] wird das Einlesen von CAD-Dateien im STEP-Format (Standard for the Exchange of Product model data) in eine interne Boundary Representation (BRep) Struktur dokumentiert. Der Aufbau der Vergleichsgeometrie wird vergleichbar mit [Tec12d] realisiert.

Ebenfalls mit OCC wird die Lage der Vergleichsgeometrien R_L bezüglich des von der Geometrie bestimmten Gebiets Ω bestimmt. Dazu sind die in [Tec12b] dokumentierten booleschen Algorithmen nötig. Zur „Auswertung“ von deren Ergebnissen wird dagegen Gmsh verwendet, weil die Auswertung nach Meinung des Autors mit Gmsh intuitiver funktioniert. In der Vermeidung der Übersetzung von einer OCC- in eine Gmsh-Geometrie liegt daher möglicherweise Optimierungspotential.

Darüber hinaus wird OCC für Geometrie-Transformationen benutzt, bspw. zur Positionierung von zweidimensionalen Geometrien außerhalb der xy-Ebene in diese hinein. Dies geschieht durch je eine Rotation um die x- und um die y-Achse mit anschließender Translation aus der Parallelebene in die xy-Ebene hinein. Zusätzlich wird auch die Verschiebung der Vergleichsgeometrie R_L, die im Ursprung des Koordinatensystems erstellt wird, zum

[2] http://www.opencascade.org/
[3] https://github.com/tpaviot/oce/

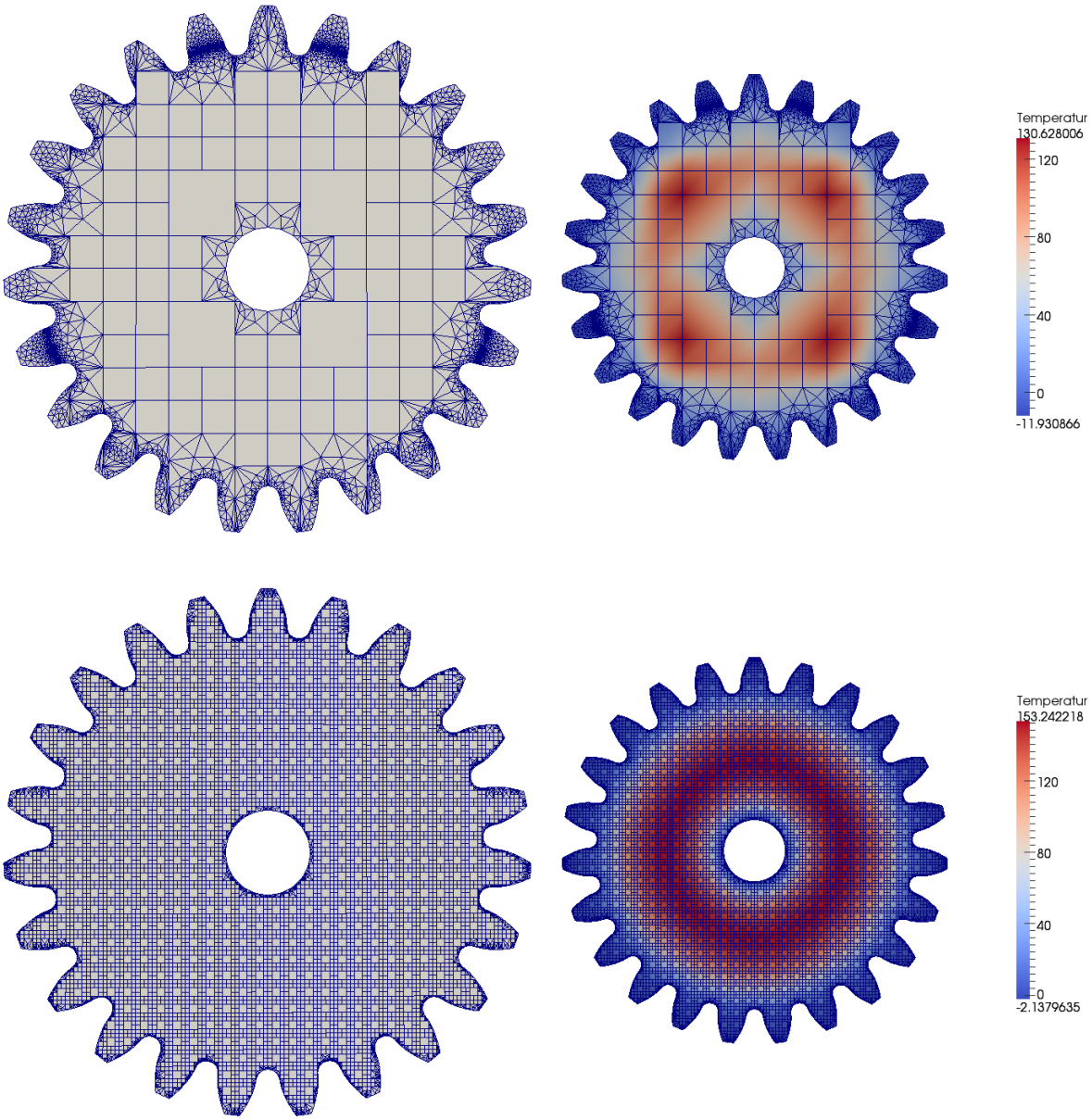

Abbildung 3.5.: Wärmeverteilung innerhalb eines Zahnrads bei 0° am Rand. Oben dargestellt ist das Ergebnis der Berechnung auf Level 2, unten das der Berechnung auf Level 5.

koordinatenmäßig-minimalen Punkt der Zelle und die Verschiebung der Vergleichsgeometrie durch die Zelle als Geometrietransformation realisiert. Dies ist ebenfalls in [Tec12b] dokumentiert.

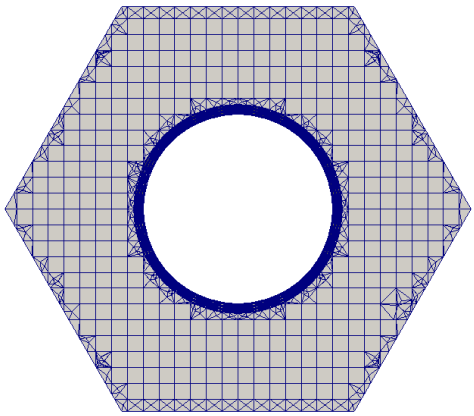

Abbildung 3.6.: Zerlegung einer Schraubenmutter, Level 3

Die Erkennung der Randgeometrie erfolgt mit der in [Tec12c] erläuterten Funktionalität zur Berechnung von Extremata.

3.2.3. Gmsh

Gmsh[4] ist ein quelloffener 3D-FEM-Gittergenerator mit zusätzlicher Funktionalität zum Pre- und Post-Processing. Gmsh ist funktional in vier Module - „Geometry“, „Mesh“, „Solver“ und „Post-Processing“ - gegliedert, von denen in der vorliegenden Arbeit v. a. das Mesh-Modul und in kleinen Teilen das Geometrie-Modul verwendet werden. Informationen zur Funktionalität von Gmsh findet man bei [GMR] oder im Reference Manual [GR12]. Der Einsatz von Gmsh in Verbindung mit OCC bietet sich besonders aus drei Gründen an:

1. Gmsh bringt die Funktionalität, OCC-Geometrien in das eigene Format zu überführen, von Haus aus mit.
2. Gmsh hat diverse Gittergenerierungsalgorithmen implementiert.
3. Gmsh kann als Bibliothek verwendet und damit komplett aus einer eigenen Anwendung heraus gesteuert werden.

Die Verwendung als Bibliothek ist allerdings nicht so verbreitet, wie eine Verwendung als eigenständiges Programm. Bspw. sieht der in Kapitel 2.1 beschriebene Workflow des dune-pdelabs den Einsatz als eigenständiges Programm vor und auch im Reference Manual [GR12] geht es in erster Linie um den Einsatz von Gmsh als geskriptetes Standalone Programm.

[4] http://geuz.org/gmsh/

Das Geometriemodul wird zur Erkennung leerer Zellen verwendet. Außerdem wird die Gittergenerierung auf mit OCC erzeugten Teilgeometrien angewendet und anschließend das erzeugte Gitter exploriert, um aus den Gitterknoten die Integrationszellen der abteilungseigenen PUM-Software zu konstruieren und - im Falle der erstellten Bibliothek - die den Rand approximierenden Knoten zu finden. Dieser Schritt entfällt durch die Trennung der Gittergenerierung für innere Zellen und Randzellen in der abteilungseigenen PUM-Software.

Die Möglichkeiten, die Gmsh bzgl. Parametrisierierung anbietet, werden nur sehr eingeschränkt verwendet, so dass auch hier möglicherweise Potential für weitere Optimierungsschritte liegt. Es wird nur ein einziger Parameter eingestellt, nämlich der in [GR12, Abschnitt 6.3.1] erläuterte Parameter `Mesh.CharacteristicLengthFromPoints`. Dieser wird für alle Geometrieknoten vom Standardwert nach dem Import einer OCC-Geometrie (10^{-22}) auf $0.05 * L_{\text{Bounding Box in x-Richtung}}$ gesetzt.

4. Bewertung der Implementierung

4.1. Darstellung der verwendeten Geometrien

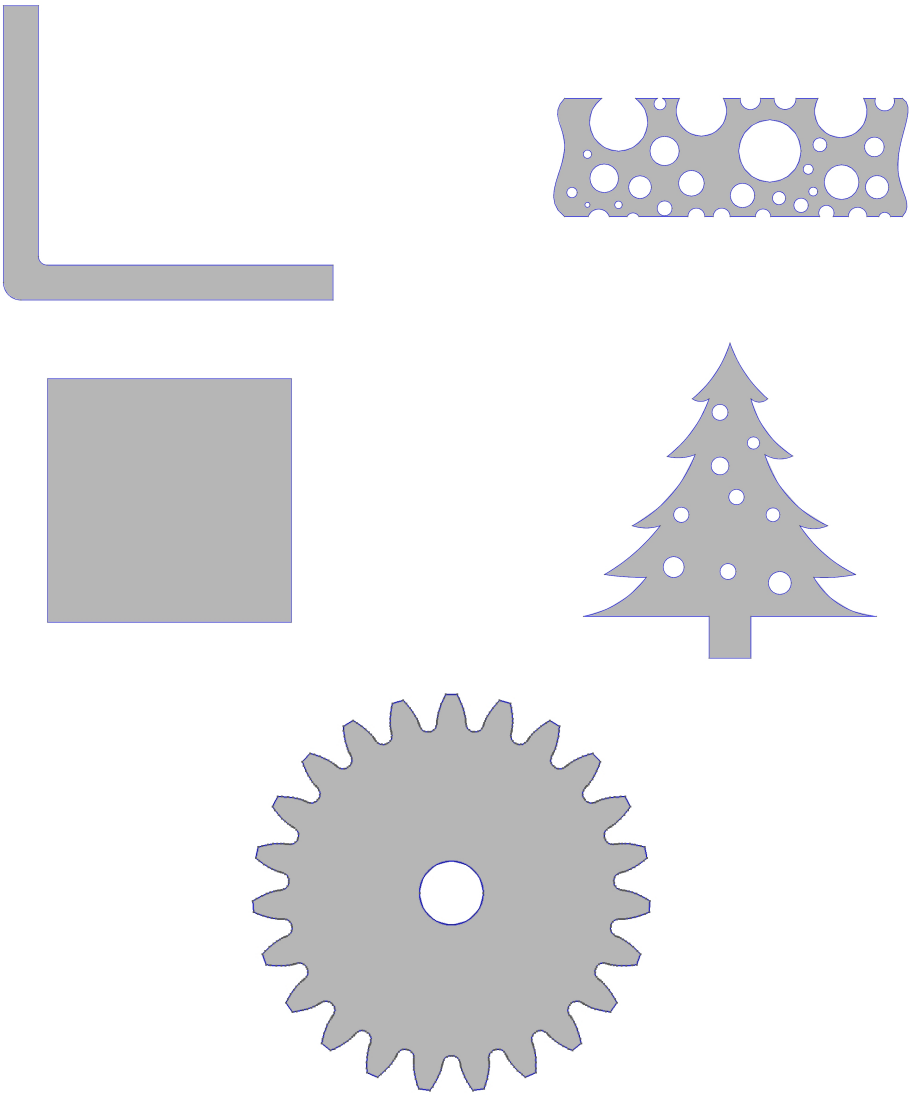

Abbildung 4.1.: Verwendete 2D-Geometrien

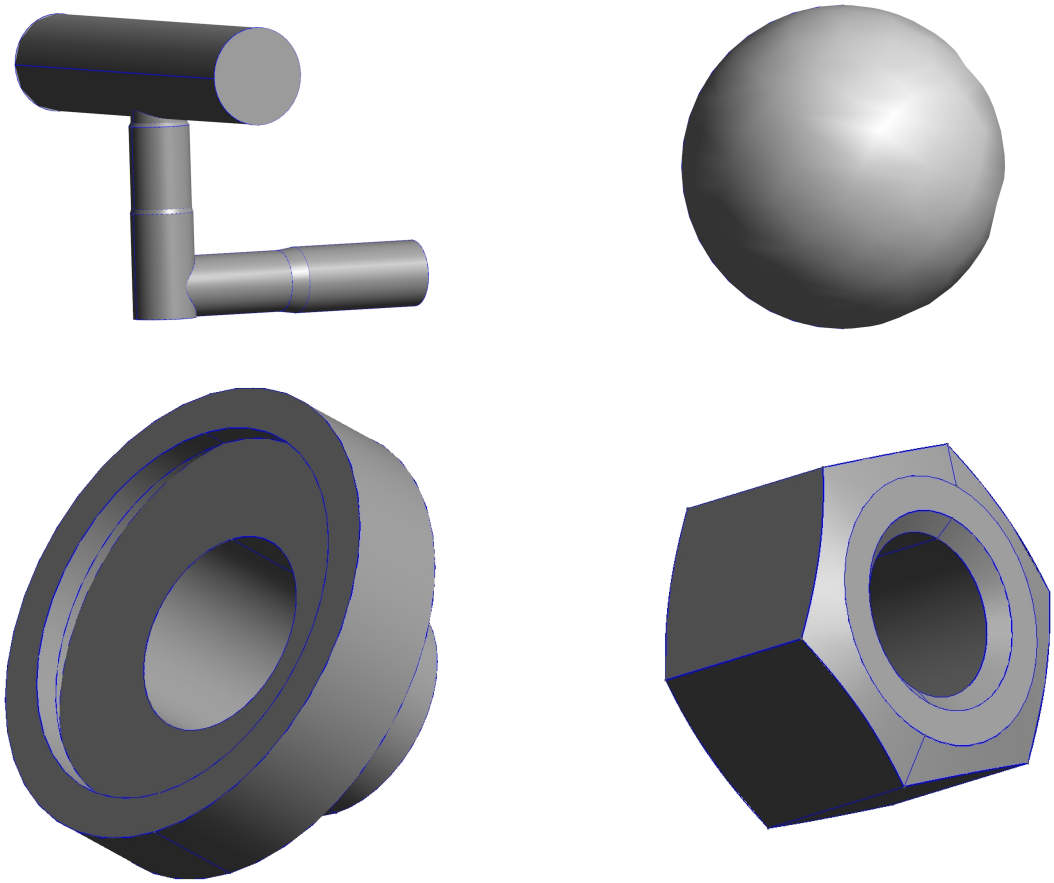

Abbildung 4.2.: Verwendete 3D-Geometrien

Das Modell der Rohre stammt aus dem Dolfyn-Forum[1].

Die Modelle des Dichtrings, der Schraubenmutter und des Zahnrads stammen aus der Traceparts-Bibliothek[2]. Der Dichtring ist das Modell DR19 der Firma Antrieb OOO-Russia, die Schraubenmutter ein Standardteil nach der EN-ISO Norm 4035 in der Größe M5 und das Zahnrad stammt von der Firma Winkel und hat die Artikelnummer 218.504.023. Das Zahnrad ist mit einer Ebene geschnitten, um eine zweidimensionale Geometrie zu erzeugen.

Die CAD-Modelle von Quadrat, Schwamm und Tannenbaum finden sich unter den Beispielgeometrien der abteilungseigenen PUM-Software. Die übrigen Modelle hat der Autor selbst erstellt.

[1] http://www.dolfyn.net/dolfyn/forum/viewtopic.php?f=5&t=441
[2] http://www.tracepartsonline.net/

4.2. Zeitkomplexität

Hinweis: Alle Zeitangaben sind nach folgendem Prinzip aufgebaut: hh:mm:ss.ss

4.2.1. Zeitmessung: Baumaufbau: Globaler oder lokaler Vergleich

In den folgenden beiden Abschnitten wird ein reguläres Cover C für einen durch ein CAD-Modell definierten Bereich Ω mit der in der Bibliothek bereitgestellten Funktionalität nach dem in Abbildung 3.1 auf Seite 20 bzw. in Abbildung 3.2 dargestellten Zerlegungsprinzip erstellt. Bei der Erstellung wird die Lage der so erzeugten Patches ω_i geprüft, da die Zerlegung nur dann vorgenommen wird, falls $\partial\Omega \cap \omega_i \neq \emptyset \wedge \omega_i \nsubseteq \Omega$, d.h. falls ein Patch ω_i den Rand des Gebiets schneidet. Diese Erkennung wird nach dem in Kapitel 3.1.2 beschriebenen Verfahren durchgeführt. Dabei werden zwei Operationen verwendet: `BRepAlgoAPI_Cut` und `BRepAlgoAPI_Common`.

Bei dieser Zeitmessung soll ermittelt werden, ob Schnitt- bzw. Differenzmengenoperation durch das Halten eines in Abschnitt 3.1.1, Aufzählungspunkt 8 erläuterten Quad- bzw. Octtree beschleunigt werden. Dadurch besteht die Möglichkeit, den Patch ω_i entweder mit der im Wurzelknoten des Baums gespeicherten Shape Ω oder mit der im Elternknoten gespeicherten Shape des Randbereichs $\Omega \cap \omega_j$ zu vergleichen.

Da die entsprechende Funktion zur Aufstellung des Baums in der Bibliothek rekursiv implementiert ist, ist zu beachten, dass in der gemessenen Zeit für Level (i) alle Operationen der Level $(i-1), (i-2), \ldots, (1), (0)$ ebenfalls enthalten sind.

1. Operation `BRepAlgoAPI_Cut`

   Begonnen wird mit der Differenzmengenoperation. Dabei werden alle Zeiten unter Verwendung der lokaler Schnittmengenoperation gemessen.

Tabelle 4.1.: Baumaufbau: lokale und globale Differenzmengenbildung (2D)

Geometrie	L		Weihnachtsbaum	
	lokal	global	lokal	global
Level 0	00:00:00.01	00:00:00.02	00:00:00.10	00:00:00.10
Level 1	00:00:00.04	00:00:00.03	00:00:00.20	00:00:00.21
Level 2	00:00:00.06	00:00:00.07	00:00:00.34	00:00:00.43
Level 3	00:00:00.14	00:00:00.18	00:00:00.62	00:00:00.91
Level 4	00:00:00.39	00:00:00.46	00:00:01.45	00:00:02.29
Level 5	00:00:01.22	00:00:01.43	00:00:02.82	00:00:05.36
Level 6	00:00:02.85	00:00:03.33	00:00:05.89	00:00:12.28
Level 7	00:00:05.90	00:00:07.05	00:00:12.36	00:00:27.78

Tabelle 4.2.: Baumaufbau: lokale und globale Differenzmengenbildung (3D)

Geometrie	Rohre		Kugel	
	lokal	global	lokal	global
Level 0	00:00:00.06	00:00:00.06	00:00:00.02	00:00:00.02
Level 1	00:00:00.48	00:00:00.53	00:00:00.29	00:00:00.23
Level 2	00:00:01.83	00:00:02.16	00:00:03.13	00:00:02.29
Level 3	00:00:07.63	00:00:08.47	00:00:20.52	00:00:15.22
Level 4	00:00:40.64	00:00:39.20	00:01:46.71	00:01:18.41
Level 5	00:03:12.82	00:02:55.58	00:07:59.62	00:05:49.61
Level 6	00:12:48.93	00:11:46.46	00:33:50.79	00:24:43.81
Level 7	00:54:53.30	00:48:35.02	02:20:06.77	01:42:07.84

Dabei ergibt sich anhand der in den Tabellen 4.1 und 4.2 gemessenen Werte, die in Abbildung 4.3 absolut und relativ zueinander visualisiert sind, dass es im zweidimensionalen Fall besser ist, den aktuellen Patch mit dem Patch aus dem Elternknoten zu vergleichen, während im dreidimensionalen Fall das Gegenteil der Fall ist.

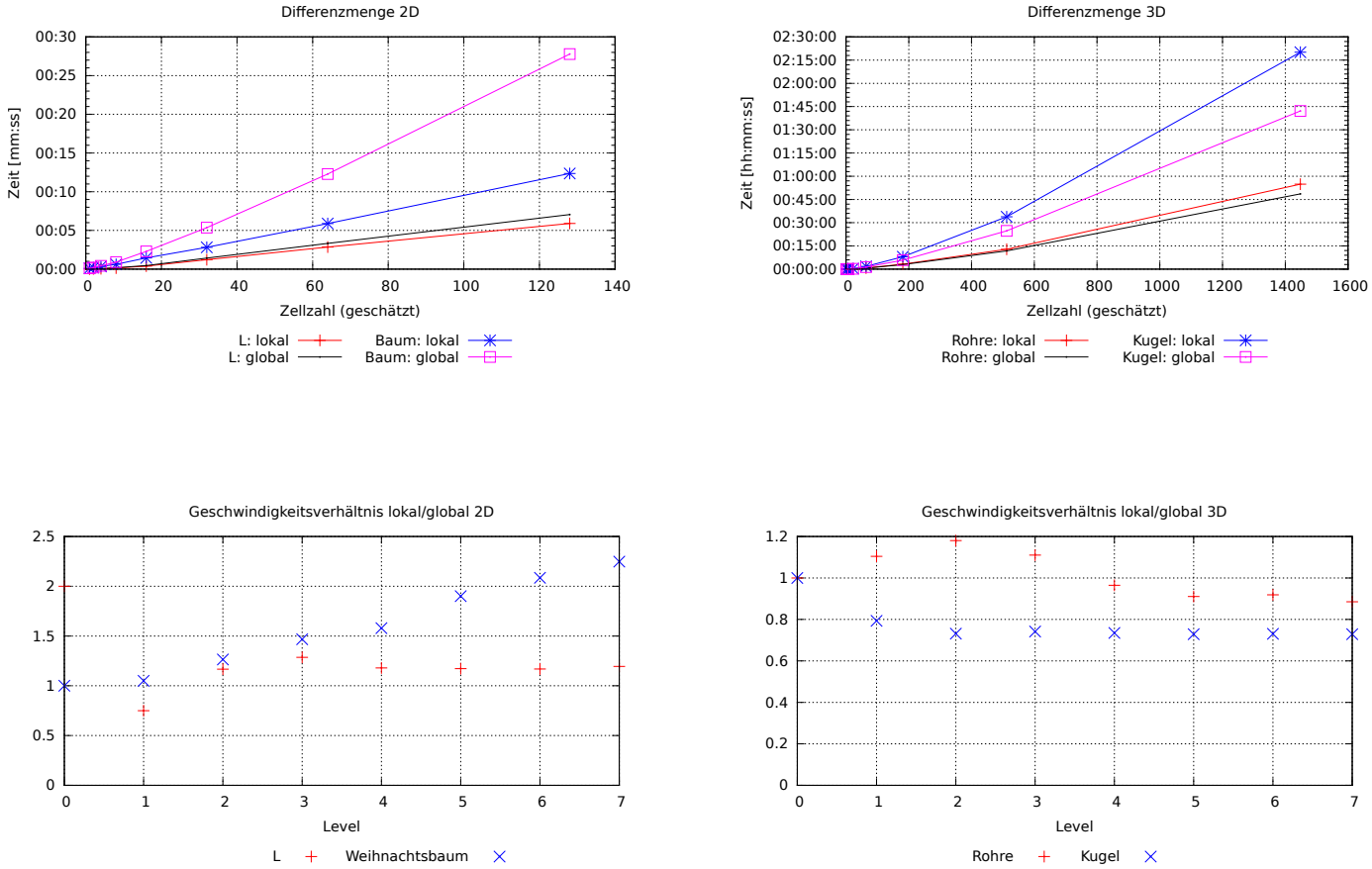

Abbildung 4.3.: Zeitmessungsplots zur Visualisierung der Dauer der Differenzmengenoperation: Auf der linken Seite befinden sich die Daten für zwei- und auf der rechten Seite für dreidimensionale Geometrien. Oben die absolute Zeit, unten das Verhältnis $t_{\text{lokal}}/t_{\text{global}}$.

2. Operation `BRepAlgoAPI_Common`

   Bei dieser Messung werden die Ergebnisse des vorherigen Abschnitts berücksichtigt, d.h. für den zweidimensionalen Fall wird `BRepAlgoAPI_Cut` mit Teilgeometrien aufgerufen, im dreidimensionalen Fall erfolgt der Aufruf mit dem gesamten Bereich Ω.

Tabelle 4.3.: Baumaufbau: lokale und globale Schnittmengenbildung (2D)

Geometrie	L		Weihnachtsbaum	
	lokal	global	lokal	global
Level 0	00:00:00.01	00:00:00.01	00:00:00.10	00:00:00.10
Level 1	00:00:00.04	00:00:00.03	00:00:00.20	00:00:00.19
Level 2	00:00:00.06	00:00:00.07	00:00:00.34	00:00:00.39
Level 3	00:00:00.14	00:00:00.17	00:00:00.62	00:00:00.84
Level 4	00:00:00.39	00:00:00.43	00:00:01.45	00:00:02.07
Level 5	00:00:01.22	00:00:01.33	00:00:02.82	00:00:05.04
Level 6	00:00:02.85	00:00:03.18	00:00:05.89	00:00:11.65
Level 7	00:00:05.90	00:00:06.66	00:00:12.36	00:00:26.42

Tabelle 4.4.: Baumaufbau: lokale und globale Schnittmengenbildung (3D)

Geometrie	Rohre		Kugel	
	lokal	global	lokal	global
Level 0	00:00:00.06	00:00:00.05	00:00:00.02	00:00:00.04
Level 1	00:00:00.53	00:00:00.54	00:00:00.23	00:00:00.23
Level 2	00:00:02.16	00:00:02.42	00:00:02.29	00:00:01.85
Level 3	00:00:08.47	00:00:09.50	00:00:15.22	00:00:11.32
Level 4	00:00:39.20	00:00:40.16	00:01:18.41	00:00:55.33
Level 5	00:02:55.58	00:02:51.48	00:05:49.61	00:04:00.38
Level 6	00:11:46.46	00:11:28.11	00:24:43.81	00:16:39.07
Level 7	00:48:35.02	00:45:42.73	01:42:07.84	01:07:38.22

Für die Schnittmengenoperation setzt sich die Tendenz analog zur Differenzmengenoperation fort, wie die Tabellen 4.3 und 4.4 zeigen. Die Ergebnisse sind in Abbildung 4.4 ebenso grafisch aufbereitet wie die der Differenzmengenoperation.

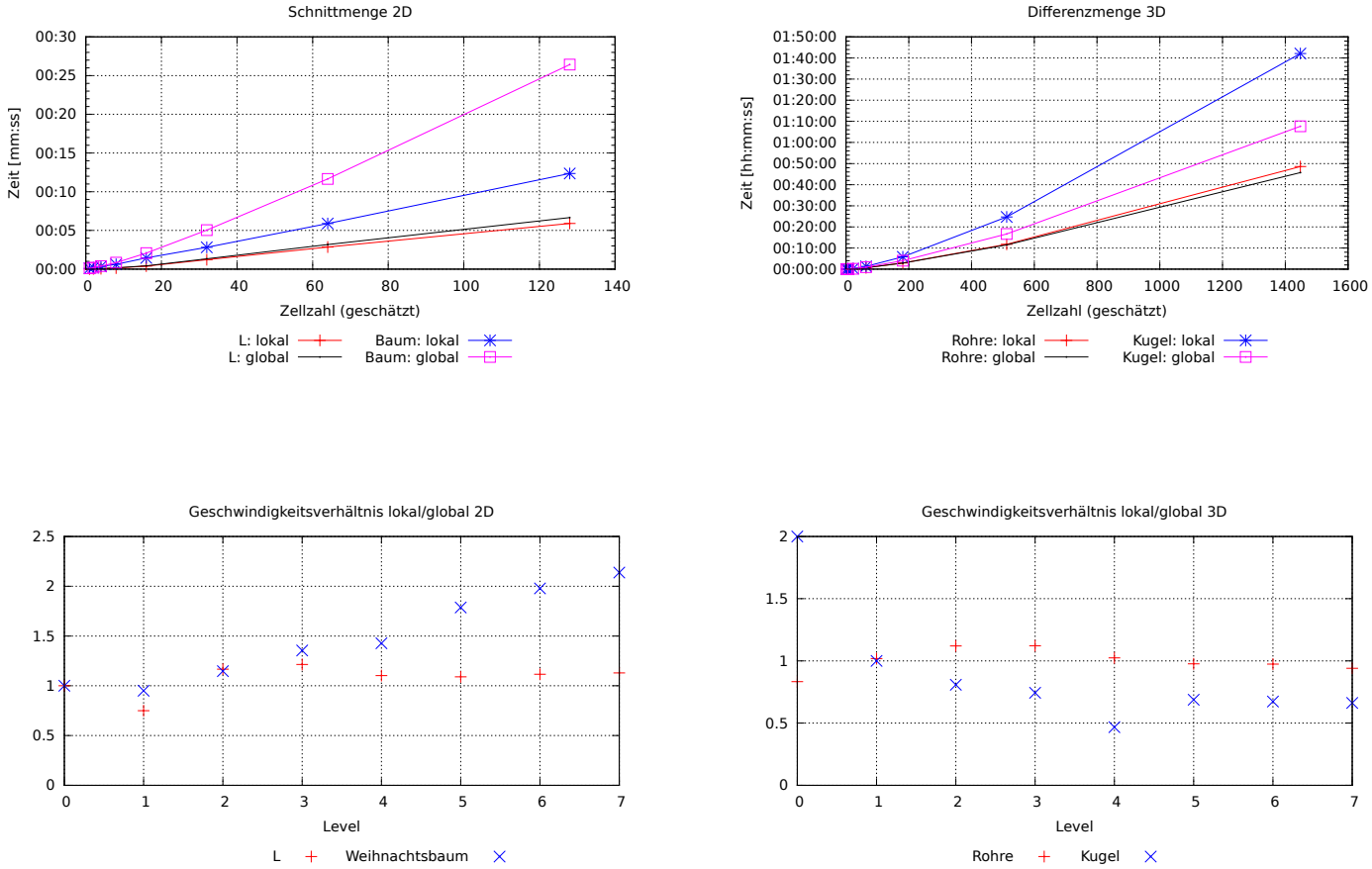

Abbildung 4.4.: Zeitmessungsplots zur Visualisierung der Dauer der Schnittmengenoperation: Auf der linken Seite befinden sich die Daten für zwei- und auf der rechten Seite für dreidimensionale Geometrien. Oben die absolute Zeit, unten das Verhältnis $t_{\text{lokal}}/t_{\text{global}}$.

4.2.2. Zeitmessung: Auswirkung der Parametrisierung von Gmsh

Die Verwendung des Parameters `Mesh.CharacteristicLengthFromPoints`, der auf allen Knoten der Gmsh-Geometrie lebt, wurde bereits in Kapitel 3.2.3 erläutert. Mit ihm lässt sich die Maschenweite des erzeugten Gitters kontrollieren und damit die Qualität der Geometrieapproximation bestimmen.

In diesem Abschnitt wird untersucht, wie sich eine Veränderung des Parameters von den Standardeinstellungen nach dem Import einer OCC-Shape von 10^{-22} auf $0.05 * L_{\text{BBox in x-Richtung}}$ an allen Knoten auf die Laufzeit auswirkt.

Im Zweidimensionalen wird einmal das Quadrat für die Level 1 bis 7 betrachtet und einmal der Schwamm für die Level 3 bis 7. Beide Geometrien sind in Abbildung 4.1 dargestellt. Dabei wird mit der institutseigenen PUM-Software folgendes Problem berechnet: An allen Rändern liegt eine Dirichlet-Randbedingung vor, deren Wert an allen Rändern der Geometrie auf 0 gesetzt ist. Damit wird die Wärmeverteilung im Objekt unter Verwendung eines Multilevel-Lösers berechnet sowie die Lösung grafisch als vtk-File ausgegeben.

Für dreidimensionale Geometrien wird nur die Zeit gemessen, die für die Erzeugung der Integrationszellen von Level 1 bis Level 4 und deren Ausgabe als vtk-File benötigt wird.

Zu beachten ist hierbei zweierlei: Erstens benötigen selbst große vtk-Files kaum Zeit für die Ausgabe und zweitens ist der Aufbau des Shape-Tree für zweidimensionale Geometrien innerhalb der PUM-Software noch nicht realisiert, so dass die Positionsbestimmung von Patches insbesondere auf hohen Levels ineffizient ist. Dies wird im folgenden Kapitel genauer untersucht.

Tabelle 4.5.: Auswirkungen der Parametrisierung von Gmsh auf die Laufzeit der institutseigenen PUM-Simulationssoftware für zweidimensionale Geometrien.

Geometrie	Quadrat		Schwamm	
Parameter	$0.05 * \triangle x$	10^{-22}	$0.05 * \triangle x$	10^{-22}
Laufzeit	00:05:17.25	00:07:04.67	01:28:27.60	01:36:11.42

Tabelle 4.6.: Auswirkungen der Parametrisierung von Gmsh auf die Laufzeit der institutseigenen PUM-Simulationssoftware für dreidimensionale Geometrien.

Geometrie	Würfel		Schraubenmutter	
Parameter	$0.05 * \triangle x$	10^{-22}	$0.05 * \triangle x$	10^{-22}
Laufzeit	02:33:33.48	03:55:03.40	05:20:00.00	06:58:28.00

Bei den in den Tabellen 4.5 und 4.6 aufgeführten Ergebnissen zeigt sich, dass die Laufzeit des Programms durch die durchgeführte Parametrisierung von Gmsh deutlich, d. h. im zweistelligen Prozentbereich, reduziert werden kann.

4.2.3. Zeitmessung: Dauer des Aufrufs der neuen Funktionen

In diesem Experiment wird untersucht, wie lange der Aufruf der neuen Funktionen auf jedem Level dauert. Dabei wird zwischen der Positionsbestimmung der Patches und der Gittergenerierung für Patches, die den Rand schneiden, unterschieden.

Zu beachten ist hierbei, dass durch die Überlappung der Patches die Lokalisierungsfunktionen häufiger aufgerufen werden als in Kapitel 4.2.1, wo keine Überlappung existiert. Diese Überlappung ist für die Funktionsweise der PUM essentiell, wie in Kapitel 2.2 erläutert wird. Die durch die Überlappung entstehenden Integrationszellen zeigt Abbildung 2.2 auf Seite 17.

Tabelle 4.7.: Dauer des Aufrufs der neuen Funktionen für zweidimensionale Geometrien

Geometrie	Quadrat		Zahnrad	
	Ortsbestimmung	Mesh-Gen	Ortsbestimmung	Mesh-Gen
Level 1	00:00:00.25	00:00:00.11	00:00:30.52	00:00:15.60
Level 2	00:00:00.68	00:00:00.20	00:02:42.49	00:00:08.13
Level 3	00:00:01.73	00:00:00.40	00:05:35.26	00:00:05.14
Level 4	00:00:04.11	00:00:00.82	00:14:40.02	00:00:06.38
Level 5	00:00:08.26	00:00:01.82	00:35.21.45	00:00:09.07

Tabelle 4.8.: Dauer des Aufrufs der neuen Funktionen für dreidimensionale Geometrien

Geometrie	Würfel		Schraubenmutter	
	Ortsbestimmung	Mesh-Gen	Ortsbestimmung	Mesh-Gen
Level 1	00:00:16.95	00:00:14.44	00:00:41.06	00:00:06.44
Level 2	00:02:06.01	00:01:11.04	00:05:49.88	00:00:56.75
Level 3	00:10:21.41	00:04:52.25	00:29:26.25	00:04:10.91
Level 4	00:31:01.45	00:13:21.32	02:16:09.15	00:17:31.57

Die in den Tabellen 4.7 und 4.8 dargestellten Ergebnisse der Untersuchung zeigen, dass für die Gittergenerierung im Vergleich zur Ortsbestimmung nur sehr wenig Zeit benötigt wird. Berücksichtigt man zusätzlich die Ergebnisse aus Kapitel 4.2.2, so ergibt sich, dass die Parametrisierung von Gmsh ein extrem wirkungsvolles Instrument ist, denn der Laufzeitunterschied aus der vorangegangenen Untersuchung entsteht komplett bei der Gittergenerierung. Diese nimmt durch die Parametrisierung - gemessen an der Gesamtlaufzeit - nur eine relativ geringe Zeit in Anspruch, wie sich in der obigen Untersuchung herausgestellt hat.

Zusätzlich fällt bei Betrachtung der in Abbildung 4.5 grafisch aufbereiteten Ergebnisse auf, dass sich für dreidimensionale Geometrien je nach Komplexität zwar ein unterschiedliches Niveau für das Zeitverhältnis zwischen $t_{Lokalisierung}$ und $t_{Gittergenerierung}$ einstellt, dieses aber - zumindest im relativ begrenzten Beobachtungsbereich - etwa konstant bleibt. Im Ergebnis für zweidimensionale Geometrien zeigt sich - insbesondere für kompliziertere Geometrien wie das Zahnrad - ein gänzlich anderes Verhalten: Hier explodiert der Aufwand für die

Lokalisierung. Durch die Implementierung des Shape-Tree sollte sich ein ähnliches Verhalten einstellen, wie das für dreidimensionale Geometrien der Fall ist.

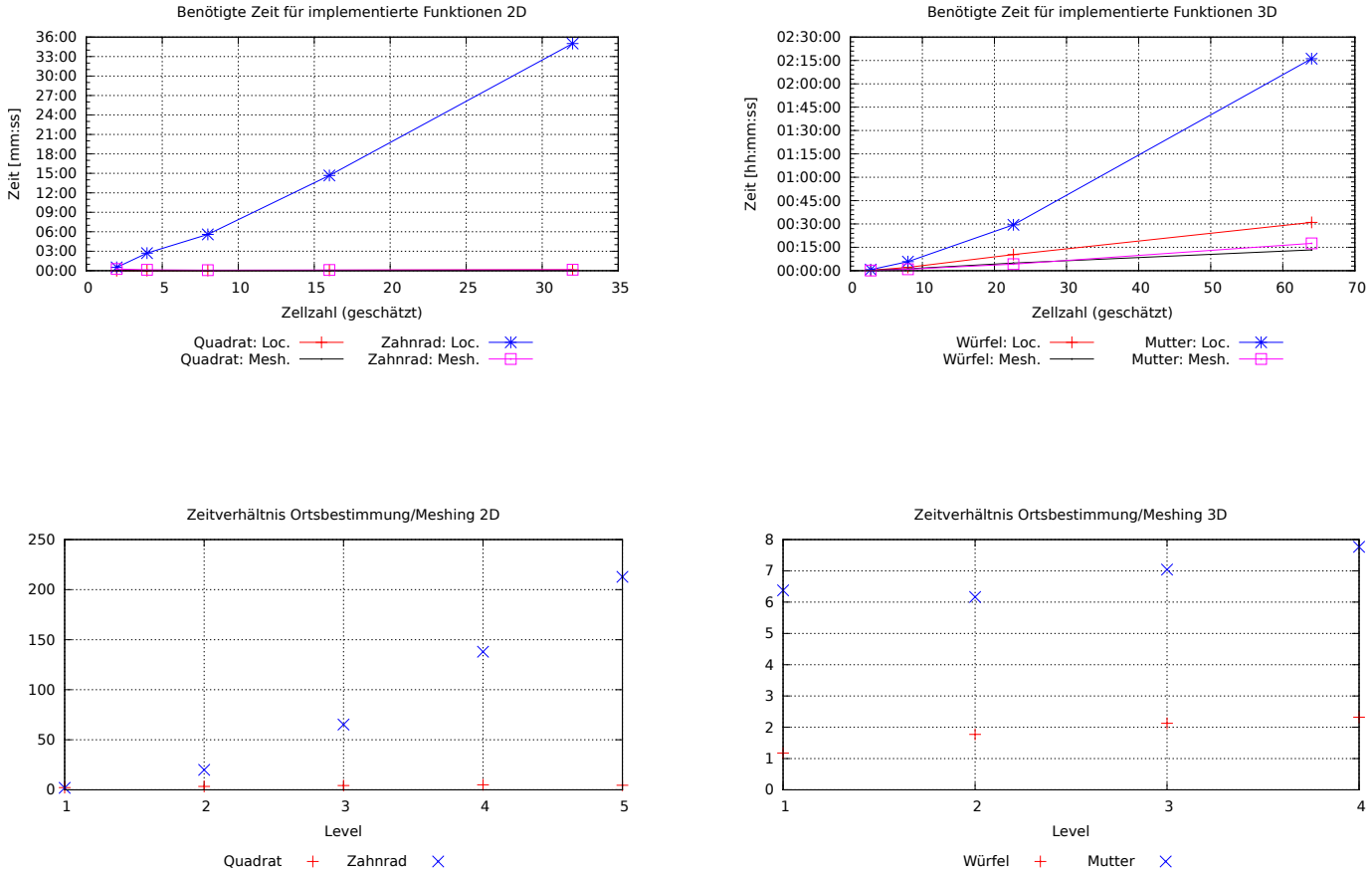

Abbildung 4.5.: Zeitmessungsplots zur Visualisierung der Aufrufdauer der spezifizierten Template-Funktionen: Auf der linken Seite befinden sich die Daten für zwei- und auf der rechten Seite für dreidimensionale Geometrien. Oben die absolute Zeit, unten das Verhältnis $t_{\text{Lokalisierung}}/t_{\text{Gittergenerierung}}$.

4.3. Zellvergleich ohne und mit neuer Funktionalität

In diesem Abschnitt sollen die Ergebnisse der Gittergenerierung der bestehenden Implementierung mit den Ergebnissen der vorliegenden Arbeit verglichen werden.

Dazu wird je eine einfache und eine komplexere Geometrie in die beiden Versionen der Simulationsumgebung geladen und mit beiden Versionen wurden Integrationszellen erzeugt.

Die erhaltenen Zerlegungen sind in Abbildung 4.6 für das Quadrat und in Abbildung 4.7 für das Zahnrad dargestellt.

Allgemein gilt, dass die Geometrie auf höheren Levels zu einem großen Teil durch innere Zellen approximiert wird, so dass der Anteil der zu triangulierenden Geometrie von Level zu Level verringert wird.

Für das Quadrat ergibt sich wegen der Gmsh Parametrisierung insbesondere auf höheren Levels kein nennenswerter Unterschied zur bestehenden Implementierung außer der Tatsache, dass Gmsh die jeweiligen Randrechtecke trianguliert, während die bisherige Implementierung Rechtecke als Randzellen akzeptiert und diese deshalb nicht trianguliert.

Für das Zahnrad ergibt sich wegen der zunehmend kleiner werdenden Fläche, die zu triangulieren ist, nach einem extremen Unterschied bei der Approximationsqualität auf Level 1 eine zunehmend geringere Abweichung zwischen bestehender Implementierung und der vorliegenden Arbeit.

Dennoch liefert die in der vorliegenden Arbeit von Gmsh erzeugte Delaunay-Triangulierung „bessere“ Integrationszellen als der bisherige Ansatz, weil dadurch der minimale innere Winkel maximiert wird, so dass die lokale Approximationsqualität steigt, vgl. [FM12, Abschnit 1.1.2].

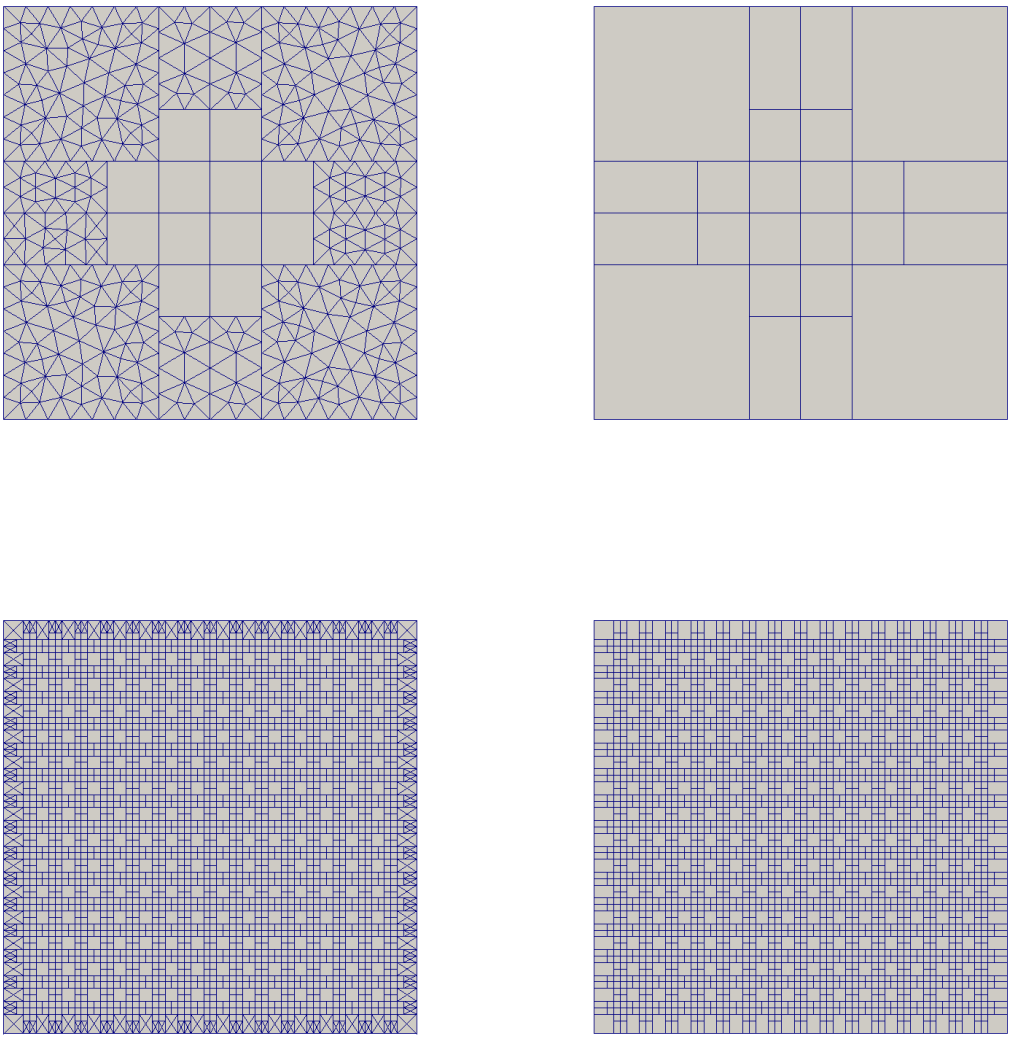

Abbildung 4.6.: Vergleich der erzeugten Integrationszellen links mit der neu implementierten Funktionalität und rechts der bisherige Stand am Beispiel eines Quadrats. Oben dargestellt ist Level 1, unten Level 4.

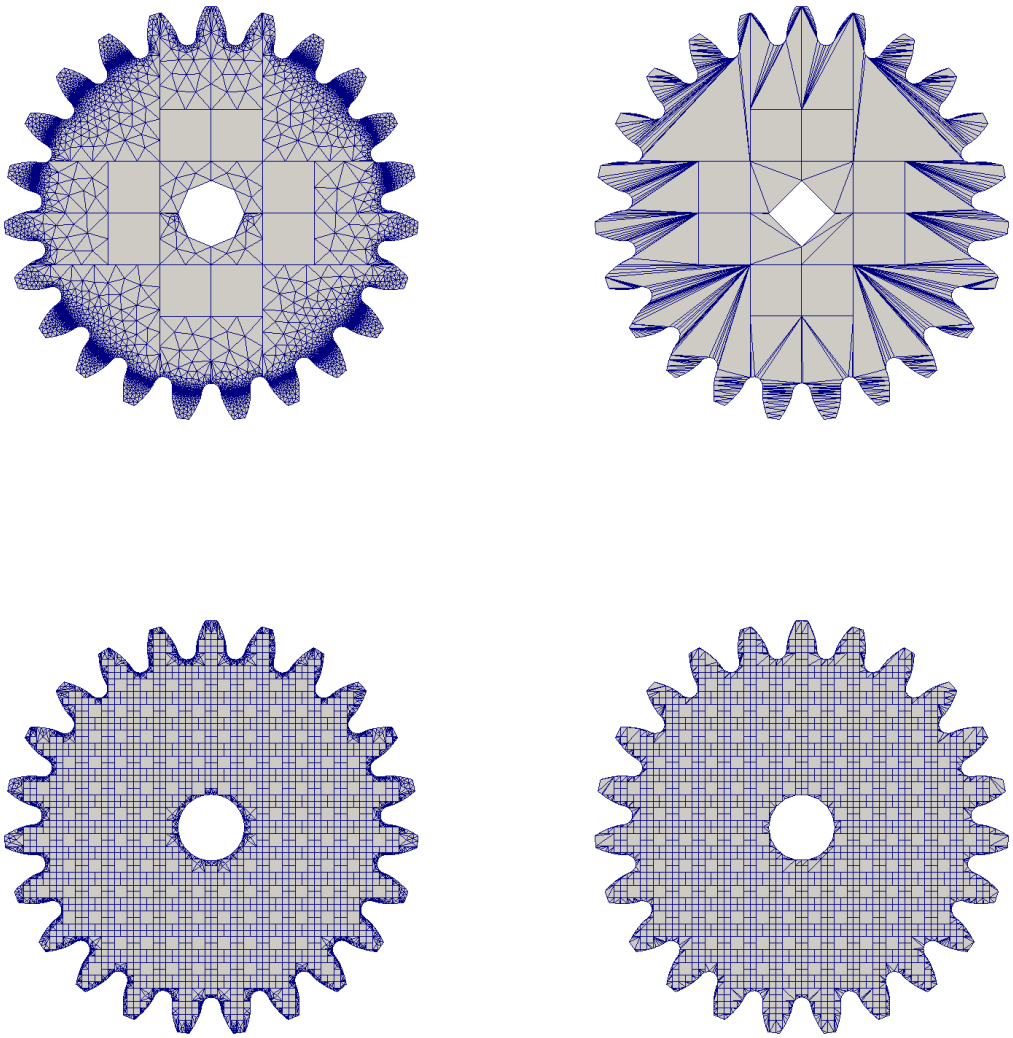

Abbildung 4.7.: Vergleich der erzeugten Integrationszellen links mit der neu implementierten Funktionalität und rechts der bisherige Stand am Beispiel eines Zahnrads. Oben dargestellt ist Level 1, unten Level 4.

5. Zusammenfassung und Ausblick

In der vorliegenden Arbeit konnte nachgewiesen werden, dass eine Erweiterung von zwei- auf dreidimensionale Geometrien ebenso möglich ist, wie die direkte Anwendung der Methode auf der Geometrie ohne den Zwischenschritt der Approximation.

Die Integration in die abteilungseigene PUM-Software war dabei relativ leicht, weil die Software von Beginn an darauf ausgelegt war, in diese Richtung erweitert zu werden.

Außerdem wurden einige Parameter gefunden, um die Laufzeit des Programms zu verbessern. Bspw. die charakteristische Maschenweite bei der Gittergenerierung in Gmsh oder die Vergleichsebene zur Ortsbestimmung der Patches in OCC, deren Wirksamkeit sowohl für einfache als auch für kompliziertere Geometrien nachgewiesen wurde.

Dennoch verbleibt weiteres Optimierungspotential. Hier ist in erster Linie die Einführung eines Shape-Trees für zweidimensionale Geometrien zu nennen, der idealerweise mit der bestehenden Baumstruktur zusammengefügt und dynamisch während der Methodenanwendung auf den einzelnen Levels aufgebaut wird. Ebenfalls ein Bereich mit großem Potential besteht in der Parallelisierung. Da die Erzeugung der Integrationszellen lokal vollständig unabhängig durchgeführt werden kann, ist der für die Parallelisierung zu betreibende Aufwand bei der Gittergenerierung relativ klein. Ein weiteres Feld besteht in der Benutzbarkeit des Programms, insbesondere wäre ein Mechanismus zur effizienten Festlegung von Randbedingungen wünschenswert.

A. Weitere Beispielgeometrien

A.1. 2D

Quadrat

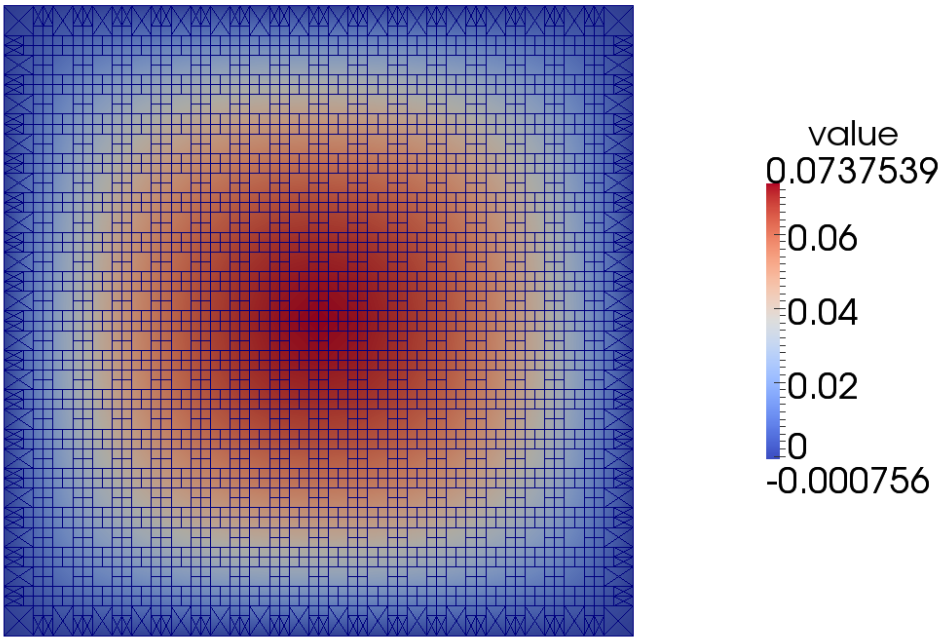

Abbildung A.1.: Wärmeverteilung innerhalb einer quadratischen Geometrie, Level 4

Schwamm V1

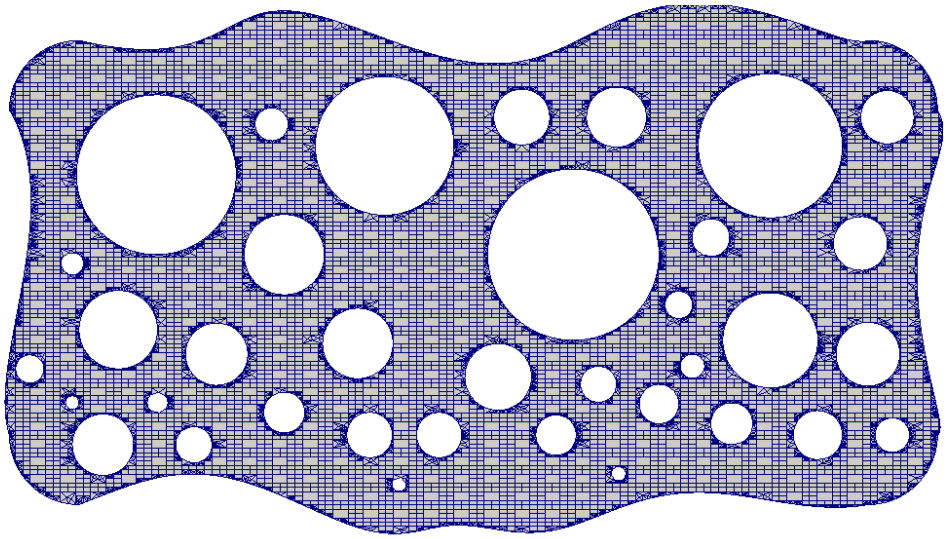

Abbildung A.2.: Zerlegung eines Schwamms, Level 6

Schwamm V2

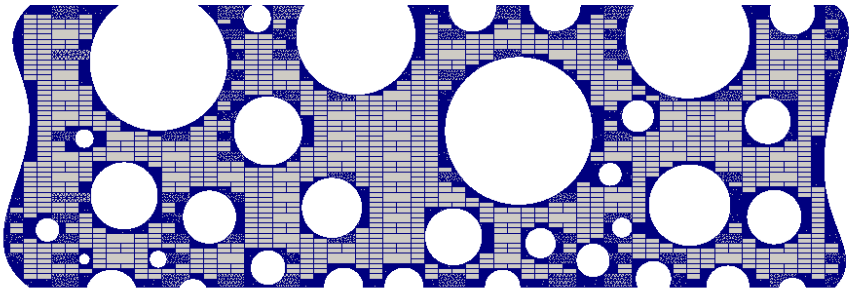

Abbildung A.3.: Zerlegung eines weiteren Schwamms, Level 4

Weihnachtsbaum

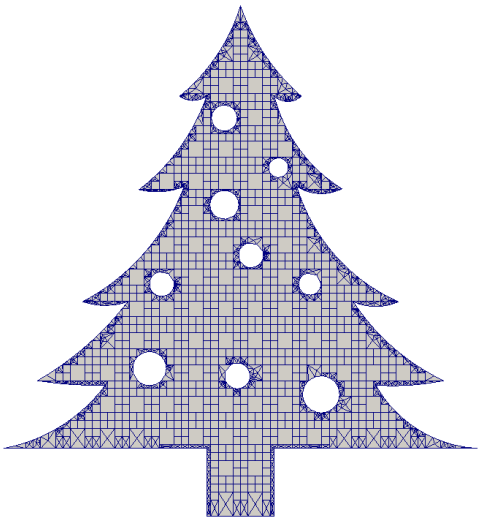

Abbildung A.4.: Zerlegung der Geometrie eines Tannenbaums, Level 4

A.2. 3D

Würfel

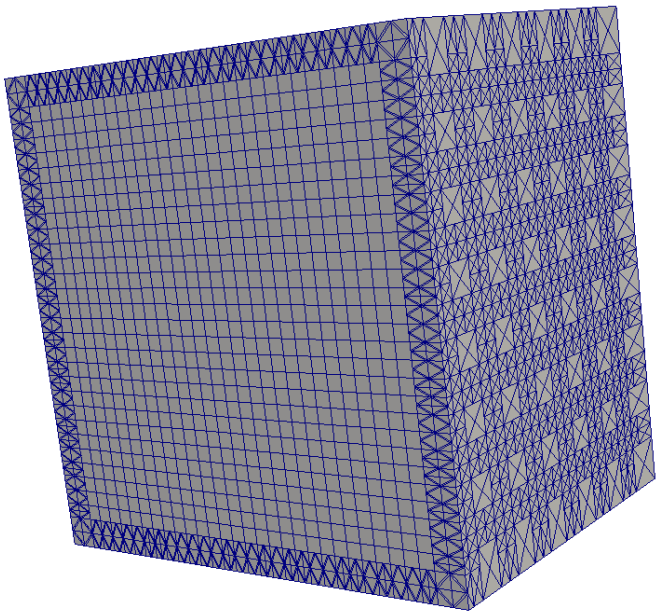

Abbildung A.5.: Zerlegung eines Würfels, Level 3

Kugel

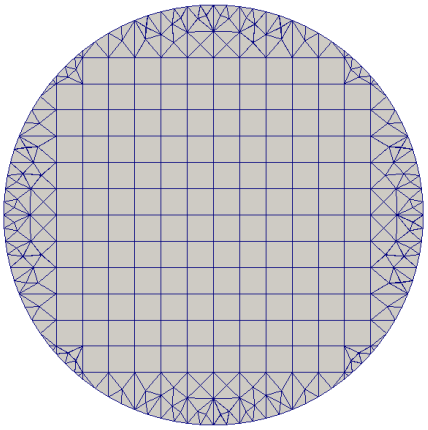

Abbildung A.6.: Schnittdarstellung der Zerlegung einer Kugel, Level 2

Gelenkwellendichtring

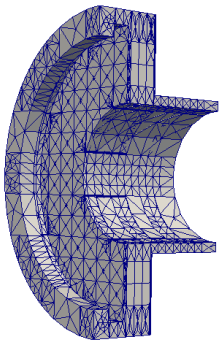

Abbildung A.7.: Schnittdarstellung der Zerlegung eines Gelenkwellendichtrings, Level 2

Literaturverzeichnis

[Bas12] F. Bastian. dune-pdelab Howto, 2012. URL http://www.dune-project.org/pdelab/pdelab-howto-1.0.1.pdf. (Zitiert auf Seite 13)

[FM12] S. Funke, N. Milosavljevic. (Draft) Notes for Computational Geometry, 2012. URL http://www.fmi.uni-stuttgart.de/fileadmin/user_upload/fmi/alg/lehre/ss12/COMPGEOM/scribe_notes.pdf. (Zitiert auf Seite 42)

[GMR] C. Geuzaine, E. Marchandise, J.-F. Remacle. An introduction to Geometric Modelling and Mesh Generation The Gmsh Companion. URL http://perso.uclouvain.be/vincent.legat/teaching/data/meca2170-GmshCompanion.pdf. (Zitiert auf Seite 29)

[GR12] C. Geuzaine, J.-F. Remacle. Gmsh Reference Manual, 2012. URL http://geuz.org/gmsh/doc/texinfo/gmsh.pdf. (Zitiert auf den Seiten 29 und 30)

[Sch03] M. A. Schweitzer. *A Parallel Multilevel Partition Of Unity Method For Elliptic Partial Differential Equations*. Springer, 2003. (Zitiert auf Seite 13)

[Sch05] M. A. Schweitzer. Efficient Implementation and Parallelization of Meshfree and Particle Methods—The Parallel Multilevel Partition of Unity Method. In J. F. Blowey, A. W. Craig, Herausgeber, *Frontiers of Numerical Analysis*, Universitext, S. 195–262. Springer Berlin Heidelberg, 2005. URL http://dx.doi.org/10.1007/3-540-28884-8_4. 10.1007/3-540-28884-8_4. (Zitiert auf den Seiten 13, 15 und 16)

[Sch08] M. A. Schweitzer. *Meshfree and Generalized Finite Element Methods*. Habilitationsschrift, Institut für Numerische Simulation, Universität Bonn, 2008. (Zitiert auf Seite 13)

[Tec12a] O. Technology. Data Exchange - STEP Import/Export User's Guide, 2012. URL http://www.opencascade.org/getocc/download/loadocc/. (Zitiert auf Seite 27)

[Tec12b] O. Technology. Object Libraries - Modeling Algorithms User's Guide, 2012. URL http://www.opencascade.org/getocc/download/loadocc/. (Zitiert auf den Seiten 26, 27 und 28)

[Tec12c] O. Technology. Object Libraries - Modeling Data User's Guide, 2012. URL http://www.opencascade.org/getocc/download/loadocc/. (Zitiert auf Seite 29)

[Tec12d] O. Technology. Tutorial - My First Application, 2012. URL http://www.opencascade.org/getocc/download/loadocc/. (Zitiert auf Seite 27)

Alle URLs wurden zuletzt am 29. Oktober 2012 geprüft.

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(Michael Lahnert)