Karlsruher Institut für Technologie

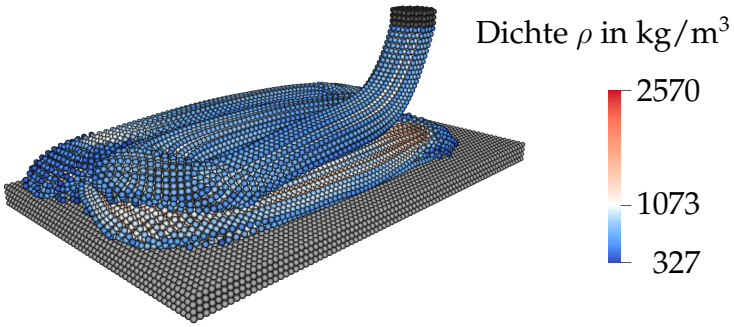
Institut für Fahrzeugsystemtechnik
Leichtbautechnologie
Leitung: Prof. Dr.-Ing. Luise Kärger

Rintheimer Querallee 2
76131 Karlsruhe

Masterarbeit

Entwicklung eines Simulationsmodells zur Strangablage in der Materialextrusion mit der Smoothed Particle Hydrodynamics Methode

Lukas Valentin Hof, B. Sc.



Projektleiter: Felix Frölich, M. Sc.
Leichtbau

Nr.: 22-L-0025

Karlsruhe, November 2022

Karlsruher Institut für Technologie

Institut für Fahrzeugsystemtechnik
Leichtbautechnologie
Leitung: Prof. Dr.-Ing. Luise Kärger

Rintheimer Querallee 2
76131 Karlsruhe

Entwicklung eines Simulationsmodells zur Strangablage in der Materialextrusion mit der Smoothed Particle Hydrodynamics Methode

Developement of a modell simulating the extrusion of a melt strand in material extrusion using smoothed particle hydrodynamics

Lukas Valentin Hof, B. Sc.
Matr.-Nr.: 1964029

Die Abschlussarbeit ist im engen Kontakt mit dem Institut auszuarbeiten. Alle dem Institut übergebenen Exemplare werden Eigentum des Instituts. Eine Einsichtnahme Dritter in diese Exemplare darf nur nach Rücksprache mit dem Institut erfolgen.

Ausgabetag: 02.05.2022

Abgabetag: 02.11.2022

Betreuerin:

Projektleiter:

(Prof. Dr.-Ing. Luise Kärger)

(Felix Frölich, M. Sc.)

Bearbeiter:

Anschrift:

Ort	Karlsruhe
Straße	Klauprechtstraße 52
Telefon	+49 171 186 0134

(Lukas Valentin Hof, B. Sc.)

Karlsruher Institut für Technologie

Institut für Fahrzeugsystemtechnik
Leichtbautechnologie
Leitung: Prof. Dr.-Ing. Luise Kärger

Rintheimer Querallee 2
76131 Karlsruhe

Aufgabenstellung

für

Herrn Lukas Valentin Hof, B. Sc.

(Matr.-Nr.: 1964029)

Entwicklung eines Simulationsmodells zur Strangablage in der Materialextrusion mit der Smoothed Particle Hydrodynamics Methode

Developement of a modell simulating the extrusion of a melt strand in material extrusion using smoothed particle hydrodynamics

Im Bereich der additiven Fertigung steht am Institutsteil Leichtbau des Instituts Fahrzeugsystemtechnik (FAST-LB) die Materialextrusion (MEX) von Thermoplasten im Fokus. Die hiermit hergestellten Bauteile unterliegen einer ausgeprägten Prozess-Material-Bauteil-Wechselbeziehung, weswegen ein simulatives Abbilden des Prozesses von großem Nutzen ist. Die Besonderheit des filament- und schichtweisen Auftragens des Polymers führt zu einem Multiskalenproblem. Daher werden in Ansätzen zur Prozesssimulation auf Bauteilebene homogenisierte Materialdaten verwendet, welche oft durch sehr aufwendige experimentelle Untersuchungen ermittelt werden müssen. Aus diesem Grund sollen Methoden entwickelt werden, um zum einen eine virtuelle Charakterisierung der MEX-Strukturen und zum anderen ein tieferes Verständnis über den Einfluss der einzelnen Prozessparameter auf die Strangablage zu ermöglichen.
Im Rahmen dieser Arbeit soll eine Grundlage geschaffen werden, um die Entwicklung eines vollumfänglichen Simulationsmodells der Strangablage mit Hilfe der „Smoothed Particle Hydrodynamics“ (SPH)-Methode zu ermöglichen. Hierbei sollen die am Institut bereits verwendeten Tools ABAQUS und PySPH untersucht werden. Es soll ein grundlegendes Verständnis dieser Tools bezüglich ihrer Eignung zum Abbilden der MEX geschaffen und in Zukunft notwendige Erweiterungen herausgearbeitet und teilweise umgesetzt werden.

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig und nur mit den im Literaturverzeichnis angegebenen Quellen und Hilfsmitteln angefertigt zu haben.

Karlsruhe, den 2. November 2022

Danksagung

Ich möchte mich bei allen bedanken, die zu dieser Arbeit beigetragen haben. Vielen Dank an meinen Betreuer M. Sc. Felix Frölich für die Unterstützung und Geduld. Ein besonderer Dank gilt meiner Freundin Hannah, die mich während dieser Zeit aufgemuntert und manchmal ausgehalten hat. Vielen Dank auch an Lasse für die Korrekturen.

Da diese Arbeit den Schlusspunkt meines Studiums markiert, möchte ich mich auch bei meinen Eltern Ingrid und Martin bedanken, die mich dabei unterstützt haben. Nicht zuletzt gilt mein Dank all den Freunden, ohne die das Studium vielleicht nicht möglich und sicher nicht so schön gewesen wäre.

Kurzfassung

Additive Fertigungsverfahren etablieren sich zusehends in vielfältigen Anwendungsfeldern. Eine weit verbreitete Gruppe ist die Materialextrusion (MEX). Die Eigenschaften durch MEX gefertigte Bauteile sind in hohem Maße von der während des Prozesses ausgebildeten Mesostruktur, also der Anordnung und Gestalt der einzelnen Polymerstränge, abhängig. Daraus ergibt sich die Notwendigkeit, den Prozess auf mehreren Größenskalen numerisch abzubilden. Es liegen bereits verschiedene Modellierungsansätze vor, mit denen die Entstehung der Mesostruktur abgebildet wird. Ein vielversprechender Ansatz ist die Verwendung der Smoothed Particle Hydrodynamics Methode (SPH). Bisher liegt kein SPH-Modell vor, das die Extrusion repräsentativer Volumenelemente der Mesostruktur unter Berücksichtigung thermomechanischer Effekte ermöglicht. In der vorliegenden Arbeit werden die Grundlagen des MEX-Verfahrens sowie der SPH-Methode erläutert sowie ein Überblick über den Stand der Modellierung des Verfahrens präsentiert. Darauf aufbauend wird mithilfe der Simulationswerkzeuge ABAQUS und PySPH jeweils ein Modell des MEX-Prozesses erstellt. PySPH wird zu diesem Zweck um ein thermisches Modell, das die Berücksichtigung von Konduktion, Konvektion und Wärmestrahlung ermöglicht, erweitert. Beide Modelle werden verwendet, um einzelne Stränge sowie einfache Mesostrukturen zu extrudieren, und die Ergebnisse diskutiert. Mit beiden Werkzeugen kann der MEX-Prozess nur eingeschränkt abgebildet werden. Das Fluid kann bei vertretbarem Rechenaufwand nicht als inkompressibel und hochviskos modelliert werden. Aufgrund seiner Erweiterbarkeit ist PySPH jedoch gut geeignet, um das Modell weiter auszubauen.

Abstract

Additive manufacturing (AM) currently experiences rapid growth or adoption across many industries. One popular group of AM-processes is material extrusion (MEX). The properties of any part created using MEX are highly dependent on the mesostructure which is the specific arrangement and form of the layers and polymer strands that make up the part. It is therefore necessary to model the manufacturing process across multiple scales. There are various attempts to model the forming of the mesostructure. One promising approach is to use the smoothed particle hydrodynamics method (SPH). So far, there is no SPH-model that allows to extrude a representative elementary volume of a mesostructure while considering thermomechanical effects. This study introduces the fundamentals of the MEX-process and the SPH-Method. Furthermore, recent efforts modelling the process are summarized. Finally, a model of the MEX-process is implemented in the simulation tools ABAQUS and PySPH. To examine the temperature distribution, the PySPH-package is expanded to make heat conduction, convection, and radiation available. The models are used to extrude single melt strands as well as a simple mesostructure. Both tools are found to not be fully capable of modelling the process. The fluid can not be modelled as incompressible and highly viscous while also maintaining reasonable computational cost. However, PySPH is found to be more suitable due to its ability to be extended and adapted.

Inhaltsverzeichnis

Abkürzungen und Bezeichnungen

Abkürzung	**Beschreibung**
AHA	Schema nach Adami, Hu und Adams
AM	Additive Fertigung (engl. *additive manufacturing*)
CAD	Computer Aided Design
DEM	Diskrete-Elemente-Methode
EDAC	Schema der Entropie gedämpften künstlichen Kompressibilität (engl. *Entropically Damped Artificial Compressibility*)
EOS	Zustandsgleichung (engl. *equation of state*)
FAST(-LB)	(Institutsteil für Leichtbau am) Institut für Fahrzeugsystemtechnik
FDM	Fused Deposition Modeling
FEM	Finite-Elemente-Methode
FFF	Fused Filament Fabrication
FGM	Fused Granular Modeling
FLM	Fused Layer Modeling
FVK	Faserverstärkte Kunststoffe
FVM	Finite-Volumen-Methode
G-Code	Geometric Code
GD	Generative Design
KIT	Karlsruher Institut für Technologie
MEX	Materialextrusion
PLA	Polymilchsäure (engl. *polylactic acid*)
RVE	Repräsentatives Volumenelement
SPH	Smoothed Particle Hydrodynamics
STL	Stereolithografie bzw. Standard Triangle Language
TO	Topologieoptimierung

Abkürzung	Beschreibung
TVF	Transportgeschwindigkeits-Schema (engl. *Transport Velocity Formulation*)
VOF	Volume-of-Fluid-Methode
WCSPH	Klassisches Schema geringfügiger Kompressibilität (engl. *weakly compressible SPH*)

Skalar	Einheit	Beschreibung
A	m^2	Oberfläche
A_1	-	Koeffizient zur Berechnung der Null-Scherviskosität
A_2	K	Koeffizient zur Berechnung der Null-Scherviskosität
A_3	K	Koeffizient zur Berechnung von A_2
b_S	mm	Versatz zwischen den Strängen einer Ebene
c_s	m/s	Künstliche Schallgeschwindigkeit
c_p	J/kgK	Spezifische Wärmekapazität
d_D	mm	Düsendurchmesser
d_S	mm	Schichtdicke
D_1	Pa s	Koeffizient zur Berechnung der Null-Scherviskosität
D_2	K	Koeffizient zur Berechnung der Glasübergangstemperatur
D_3	K/Pa	Koeffizient zur Berechnung der Glasübergangstemperatur und von A_2
g	m/s^2	Gravitationskonstante
h	m	Glättungslänge
h_c	W/m^2K	Konvektiver Wärmetransportkoeffizient
h_in	mm	Anfänglicher Abstand zwischen Düse und Bauplattform
l_S	mm	Länge eines Stranges
m	kg	Masse
n	-	Potenzprofil bei hoher Schergeschwindigkeit
p	Pa	Druck
p_H	Pa	Hintergrunddruck

Skalar	Einheit	Beschreibung
q	-	Dimensionsloser Abstand
Q	J	Wärmemenge
Re	-	Reynoldszahl
t	s	Zeit
T	°C	Temperatur
T^*	°C	Durch Kurvenanpassung bestimmte Glasübergangstemperatur für den Cross-WLF-Ansatz
u	mm	Verschiebung in x-Richtung
v	mm	Verschiebung in y-Richtung
v_{Ex}	mm/s	Extrusionsgeschwindigkeit
v_{Vor}	mm/s	Vorschubgeschwindigkeit
V	m^3	Volumen
w	mm	Verschiebung in z-Richtung
W	$1/\mathrm{m}^n$	Glättungsfunktion im n-dimensionalen Raum
α	$1/\mathrm{m}^n$	Vorfaktor zur dimensionsabhängigen Normierung
γ	-	Exponent der Taitschen Zustandsgleichung
$\dot{\gamma}$	$1/\mathrm{s}$	Scherrate
δ	-	Dirac-Funktion
δt	s	Zeitschritt
Δt	s	Simulationszeit
Δx	mm	Partikelabstand (auch Partikeldurchmesser)
ε	-	Emissionskoeffizient
η	Pa s	Dynamische Viskosität
η_0	Pa s	Null-Scherviskosität
κ	W/mK	Wärmeleitfähigkeit
λ	-	Kleinster Eigenwert der invertierten Renormierungsmatrix
ν	m^2/s	Kinematische Viskosität
ρ	kg/m^3	Dichte
σ	$\mathrm{W}/\mathrm{m}^2\mathrm{K}^4$	Stefan-Boltzmann-Konstante
τ	Pa	Dichte
Φ	-	Beliebige differenzierbare Funktion

Vektor	Einheit	Beschreibung
$\boldsymbol{d}$	m	Verbindungsvektor
$\boldsymbol{u}$	m/s	Geschwindigkeit
$\tilde{\boldsymbol{u}}$	m/s	Advektionsgeschwindigkeit
$\boldsymbol{x}$	m	Ort

Tensor	Einheit	Beschreibung
$\boldsymbol{A}$	-	Renormierungsmatrix
$\boldsymbol{D}$	1/s	Deformationsgeschwindigkeitstensor
$\boldsymbol{L}$	1/s	Geschwindigkeitsgradiententensor

Index	Beschreibung
$()_0$	Referenzwert
$()_{1D}$	Im eindimensionalen Raum
$()_{2D}$	Im zweidimensionalen Raum
$()_{3D}$	Im dreidimensionalen Raum
$()_a$	Den Zentralpartikel betreffend
$()_{ab}$	Differenz der Werte von Zentral- und Nachbarpartikel
$()_b$	Den Nachbarpartikel betreffend
$()_{Diff}$	Differenz
$()_i$	Integrale Interpolation
$()_{max}$	Maximalwert
$()_{rel}$	Relativer Wert
$()_S$	Diskrete Summeninterpolation
$()_U$	Umgebung

Notation

Tensornotation

Zur Verdeutlichung verschiedener Tensor Stufen werden unterschiedliche Schriftarten verwendet. Bezeichner (zum Beispiel "max") werden nicht kursiv geschrieben und Variablen (zum Beispiel Summationsindices) werden kursiv geschrieben. Besondere Vektoren sind die Einheitsvektoren eines kartesischen Koordinatensystems $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, $\boldsymbol{e}_3$.

Typ	**Symbol**	**Bezeichner**	**Komponenten**
Skalare	a	$a_{\max}$	-
Vektoren	$\boldsymbol{a}$	$\boldsymbol{a}_{\max}$	a_k
Tensoren 2. Stufe	$\boldsymbol{A}$	$\boldsymbol{A}_{\max}$	A_{kl}
Tensoren 4. Stufe	$\mathbb{A}$	$\mathbb{A}_{\max}$	A_{klmn}

Zeitliche Ableitungen

Die materielle Zeitableitung wird als

$$\frac{\mathrm{D}(\cdot)}{\mathrm{D}t} = \frac{\partial(\cdot)}{\partial t} + \boldsymbol{v} \cdot \operatorname{grad}(\cdot) \tag{1}$$

eingeführt. Während $\frac{\mathrm{D}(\cdot)}{\mathrm{D}t}$ die Änderung einer Größe an einem materiellen Punkt beschreibt (Lagrange), beschreibt $\frac{\partial(\cdot)}{\partial t}$ die Änderung an einer fixen Position (Euler).

Produkte

Skalarprodukt: $\boldsymbol{a} \cdot \boldsymbol{b} = \sum_i a_i b_i$

Dyadisches Produkt: $\boldsymbol{a} \otimes \boldsymbol{b} = \underline{ab}^\top = \sum_{ij} a_i b_j \boldsymbol{e}_i \otimes \boldsymbol{e}_j$

Kreuzprodukt: $\boldsymbol{a} \times \boldsymbol{b} = \begin{pmatrix} a_2 b_3 - a_3 b_2 \\ a_3 b_1 - a_1 b_3 \\ a_1 b_2 - a_2 b_1 \end{pmatrix} \boldsymbol{e}_i$

Einfache Kontraktion: $\boldsymbol{Ab} = \sum_{ij} A_{ij} b_j \boldsymbol{e}_i$

Doppelte Kontraktion: $\mathbb{A}\boldsymbol{B} = \sum_{ijkl} A_{ijkl} B_{kl} \boldsymbol{e}_i \otimes \boldsymbol{e}_j$

Einfache Hintereinanderschaltung: $\boldsymbol{AB} = \sum_{ijk} A_{ik} B_{kj} \boldsymbol{e}_i \otimes \boldsymbol{e}_j$

Doppelte Hintereinanderschaltung: $\mathbb{AB} = \sum_{ijklmn} A_{ijmn} B_{mnkl} \boldsymbol{e}_i \otimes \boldsymbol{e}_j \otimes \boldsymbol{e}_k \otimes \boldsymbol{e}_l$

Räumliche Ableitungen

Gradient (Skalar): $\nabla a = \sum_i \frac{\partial a}{\partial x_i} \boldsymbol{e}_i$

Gradient (Vektor): $\nabla \boldsymbol{a} = \sum_i \frac{\partial a_i}{\partial x_j} \boldsymbol{e}_i \otimes \boldsymbol{e}_j$

Divergenz (Vektor): $\nabla \cdot \boldsymbol{a} = \sum_i \frac{\partial a_i}{\partial x_i}$

Divergenz (2. Stufe): $\nabla \cdot \boldsymbol{A} = \sum_j \frac{\partial A_{ij}}{\partial x_j} \boldsymbol{e}_i$

Rotation (Vektor): $\nabla \times \boldsymbol{a} = \begin{pmatrix} \frac{\partial a_3}{\partial x_2} - \frac{\partial a_2}{\partial x_3} \\ \frac{\partial a_1}{\partial x_3} - \frac{\partial a_3}{\partial x_1} \\ \frac{\partial a_2}{\partial x_1} - \frac{\partial a_1}{\partial x_2} \end{pmatrix} \boldsymbol{e}_i$

Laplace-Operator: $\nabla \cdot (\nabla a) = \nabla^2 a = \Delta a$

1 Einleitung

1.1 Motivation

Die additiven Fertigungsverfahren haben sich in den vergangenen Jahren in vielen Industrien und Anwendungen etablieren können. Es existieren Verfahren für verschiedenste Materialien und Bauteilgrößen. [1–3] In diesem Bereich sind sowohl ein starkes Marktwachstum [4] als auch umfangreiche Forschungsaktivitäten zu beobachten [5]. Eine weit verbreitete Gruppe von Verfahren ist die Materialextrusion (MEX). V. a. das Fused Filament Fabrication (FFF) Verfahren konnte sich für den Prototypenbau, im Hobbybereich und teilweise der Fertigung durchsetzen [5]. Wie die meisten additiven Verfahren erlaubt die MEX eine sehr hohe Gestaltungsfreiheit und Flexibilität. Bauteilspezifische Werkzeuge werden nicht benötigt und Bauteile können aus Metallen, Keramiken oder Kunststoffen hergestellt werden. Statt einfacher Polymere können auch faserverstärkte Kunststoffe (FVK) eingesetzt werden. Trotz der im Vergleich zu anderen Herstellungsverfahren verminderten und anisotropen mechanischen Eigenschaften, weisen additive Fertigungsverfahren durch diese Gestaltungsfreiheit ein exzellentes Leichtbaupotential auf. Um dieses Potential voll auszuschöpfen, ist es sinnvoll, MEX Fertigungsprozesse mit computergestützten Gestaltungsansätzen wie der *Topologieoptimierung* (TO) oder dem *Generative Design* (GD) zu kombinieren [6]. In jedem Fall ist der Ausgangspunkt eines additiv gefertigten Bauteils stets ein digitales Modell der Geometrie [7].

Der Fertigungsprozess führt zu Verzug und Eigenspannungen im fertigen Bauteil [1, 2]. Diese müssen möglichst präzise vorhergesagt werden, um spezifizierte Maßtoleranzen und mechanische Eigenschaften einhalten zu können sowie das Gelingen des Fertigungsprozesses sicherzustellen. Zur Vorhersage der thermomechanischen Effekte und für Struktursimulationen werden numerische Modelle eingesetzt. In der MEX sind die Abstände zwischen den einzelnen Strängen, aus denen das Bauteil aufgebaut wird, und die Schichtdicken im Vergleich zu den Bauteilabmessungen i. d. R. sehr klein. Die Modellierung des Prozesses für ein ganzes Bauteil mit einer räumlichen Auflösung, bei der der Querschnitt eines einzelnen Strangs durch mehrere Zellen bzw. Partikel abgebildet wird, ist deshalb mit sehr hohem Rechenaufwand verbunden. Es ist daher sinnvoll, den Prozess auf mehreren Ebenen zu betrachten, der *Makro-*, *Meso-* und *Mikroebene*.

Für Modelle auf Makroebene, also der Modellierung des Prozesses für in ganzes Bauteil, müssen geringere räumliche Auflösungen gewählt werden, sodass die einzelnen Stränge nicht abgebildet und stattdessen homogenisierte Materialkennwerte verwendet werden. [8–17] Für jede Kombination der Prozessparameter, wie Material, Temperatur von Schmelze, Bauplattform und Umgebung, Kühlung, Vorschubgeschwindigkeit, Düsendurchmesser, Schichtdicke, Abstand der Stränge usw., bildet sich eine spezielle *Mesostruktur* aus, deren Gestalt großen Einfluss auf den Prozess und das fertige Bauteil hat [18, 19]. Sie muss durch homogenisierte Materialkennwerte berücksichtigt werden. Die charakteristischen Eigenschaften der Mesostruktur können experimentell bestimmt werden [20–22], analytisch angenähert werden [12, 23] oder durch numerische Modelle mit höherer räumlicher Auflösung, die lediglich repräsentative Ausschnitte eines Bauteils umfassen, berechnet werden [8, 11]. Um homogenisierte Materialkennwerte mit numerischen Methoden zu bestimmen, muss allerdings die Gestalt der Mesostruktur möglichst genau bekannt sein. Dies umfasst die Gestalt und Anordnung der Polymerstränge. Numerische mesoskopische Modelle des Fertigungsprozesses können hierüber detaillierte Auskunft geben und sind weniger aufwändig als umfangreiche experimentelle Untersuchungen. Darüber hinaus lässt sich mit solchen Simulationen das allgemeine Prozessverständnis verbessern. [14, 24–36] Für Probleme, die auch auf Mesoebene nicht aufgelöst werden können, wie etwa die Betrachtung diskreter Fasern in FVK oder den Vorgängen in der Schmelzeinheit, können Mikromodelle eingesetzt werden [24–26, 33, 37–39].

Während für Modelle auf Bauteilebene i. d. R. die Finite-Elemente-Methode (FEM) eingesetzt wird [15–17, 40–45], wird die Entstehung der Mesostruktur häufig mit der Finite-Volumen-Methode (FVM) [27–33] oder der Smoothed Particle Hydrodynamics (SPH) Methode abgebildet [26, 34–36]. Die SPH-Methode eignet sich besonders für die Modellierung der MEX auf Mesoebene aufgrund der folgenden Eigenschaften:

- Es ist keine Vernetzung des Simulationsgebietes notwendig. So können geometrische Parameter mit geringem Arbeitsaufwand variiert werden.
- Für Bereiche des Simulationsgebietes, in denen kein Material vorhanden ist, fällt kein Rechenaufwand an.
- Die Methode ist gut geeignet zur Abbildung großer Deformationen.
- Es sind keine zusätzlichen Algorithmen wie die Volume-of-Fluid-Methode (VOF) notwendig, um freie Oberflächen oder Phasengrenzen zu verfolgen. [46–49]

1.2 Zielsetzung

In dieser Arbeit soll ein erstes Modell der Strangablage in der MEX mithilfe der SPH Methode erstellt werden. Das Modell wird in zwei Simulationswerkzeugen aufgebaut: in der kommerziellen Simulationssoftware ABAQUS und dem Open Source Framework PySPH. ABAQUS wird am Institutsteil für Leichtbau am Institut für Fahrzeugsystemtechnik (FAST-LB) des Karlsruher Instituts für Technologie (KIT) bereits in großem Umfang zur Modellierung verschiedener Polymere genutzt, u. a. zur makroskopischen Modellierung des MEX Prozesses, weshalb auf eine große Zahl vorhandener Materialmodelle zurückgegriffen werden kann. PySPH ist durch den Open Source Ansatz uneingeschränkt anpassbar. Am FAST-LB wurde PySPH bereits von Meyer et al. [38] zur mikroskopischen Modellierung einzelner Fasern in einer flüssigen Polymermatrix eingesetzt. Mit beiden Werkzeugen soll jeweils ein einfaches Modell der Strangablage aufgebaut und die Ergebnisse verglichen werden. Anhand dieser Ergebnisse soll das für die Anwendung besser geeignete Werkzeug bestimmt werden, um den MEX Prozess auf Mesoebene detailliert abzubilden.

2 Grundlagen und Stand der Technik

2.1 Materialextrusion

Als additive Fertigung (engl. *additive manufacturing* kurz *AM*) werden Fertigungsverfahren bezeichnet, bei denen aus 3D-Modellen Bauteile durch das Verbinden des Materials erstellt werden. Üblicherweise werden die Bauteile schichtweise aufgebaut [7]. Die *Materialextrusion (MEX)* ist eine Gruppe additiver Fertigungsverfahren, bei denen das flüssige Material durch eine Düse aufgetragen und sich anschließend durch eine chemische Reaktion bzw. Erstarren der Schmelze verfestigt [7]. Im Folgenden sollen ausschließlich Verfahren dieser Kategorie betrachtet werden, bei denen thermoplastische Polymere in Form eines Drahts oder als Granulat zum Einsatz kommen und kontinuierlich aufgetragen werden. Das MEX Verfahren mit einem Thermoplast-Draht (auch *Filament* genannt) wird als *Fused Filament Fabrication* (FFF) bezeichnet. Auch die Bezeichnungen *Fused Layer Modeling* (FLM) und *Fused Deposition Modeling* (FDM) sind üblich, wobei letzteres ein Markenname der Firma STRATASYS ist [1, 50]. Liegt das Material als Granulat vor, wird das Verfahren als *Fused Granular Fabrication* (FGM) bezeichnet [51]. Darüber hinaus ist die Fertigung von faserverstärkten Kunststoffen (FVK), Metallen oder Keramiken möglich [50]. Im Folgenden wird ausschließlich die Fertigung mit unverstärkten Thermoplasten betrachtet. Da die Vorgänge stromauf des Düsenauslasses zunächst nicht berücksichtigt werden, sind die Ergebnisse sowohl für FFF als auch FGM relevant. Zudem wird das Verfahren im Folgenden als MEX bezeichnet, obwohl dieser Begriff Verfahren einschließt, die nicht abgebildet werden. In Abbildung 2.1 ist das FFF Verfahren schematisch dargestellt. In der Abbildung verfügt die Maschine über einen *Extruder* mit zwei Düsen und über zwei Filamentspulen.

2.1.1 Funktionsprinzip

In der hier betrachten MEX wird das Filament oder Granulat dem Extruder (in Abbildung 2.1 direkt über dem Bauteil) zugeführt und in dessen elektrisch beheizter Schmelzeinheit verflüssigt. Der flüssige Werkstoff wird dann kontinuierlich aus der unmittelbar darunterliegenden Düse auf eine Bauplattform aufgetragen. Abhängig von der verwendeten Maschine wird entweder

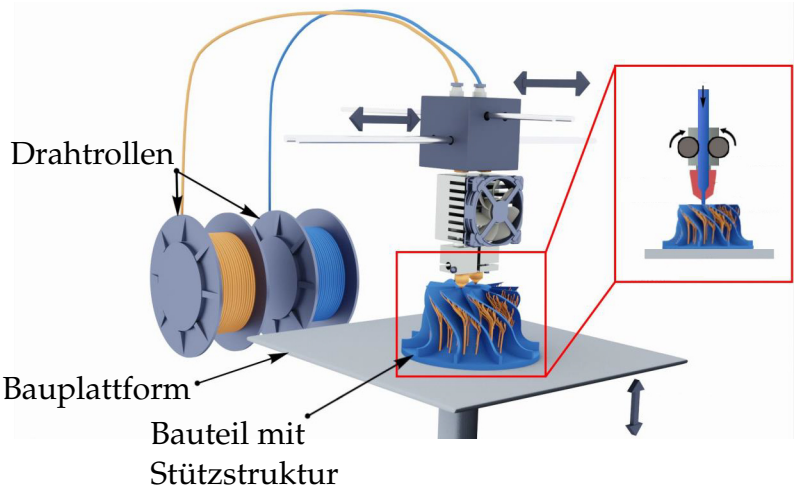


Abb. 2.1: Schematische Darstellung des FFF Verfahrens. Aus Elkaseer et al. [52].

der Extruder oder die Bauplattform in der horizontalen Ebene verschoben. Der Volumenstrom des Werkstoffs wird dabei ständig der Vorschubgeschwindigkeit angepasst, um eine möglichst konstanten Dicke des extrudierten Strangs zu erreichen. Die Stränge geben durch Konduktion Wärme an die Bauplattform bzw. zuvor abgelegte Stränge und durch Konvektion und Strahlung an die Umgebung ab, wodurch die Schmelze erstarrt. Ist das Material für eine ganze Ebene aufgetragen, verschiebt sich die Bauplattform oder die Düse entlang der Hochachse, sodass die nächste Ebene abgelegt werden kann. So wird Schicht für Schicht eine nahezu beliebige Geometrie aufgebaut. Eventuell benötigte Basis- und Stützstrukturen werden in derselben Weise zusammen mit dem Bauteil aufgebaut. Die in Abbildung 2.1 vorhandene zweite Düse erlaubt es, diese Strukturen aus einem anderen Material zu fertigen, das z. B. leichter abzulösen ist [53]. Ist das Bauteil vollständig *gedruckt*, muss es von der Bauplattform abgelöst werden. Ggf. müssen nicht mehr benötigte Stützstrukturen entfernt werden. Das fertige Teil kann anschließend weiterverarbeitet bzw. eingesetzt werden. [1, 50]

2.1.2 Slicing

Vor der eigentlichen Fertigung steht die Gestaltung des Bauteils und das *Slicing*. Das Modell des Bauteils wird aus einer beliebigen CAD-Software (*computer aided design)* exportiert, häufig im *STL*-Format (*Stereolithografie* bzw. *Standard Triangle Language*). Anschließend wird das Modell mithilfe einer *Slicing*-Software (auch *Slicer*) aufbereitet. Die Bauteilgeometrie wird in Schichten unterteilt und für jede Schicht der Pfad der MEX-Maschine bestimmt. Hier müssen relevante Druckparameter, wie Werkstoff, Düsendurchmesser, Schichtdicke, Position und Orientierung auf der Bauplattform usw., hinterlegt werden. Es ist aus Zeit- und Kostengründen oder zur

Gewichtsreduktion üblich, Bauteile nicht monolithisch herzustellen, sondern das Innere als Ausfachung (engl. *infill*) auszuführen. Des Weiteren benötigen viele Bauteile Stützstrukturen (engl. *support*) oder Basisstrukturen auf der Bauplattform. Sie sind notwendig, um Überhänge realisieren können. Die Winkel der Turbinenschaufeln in Abbildung 2.1 bspw. sind im oberen Bereich zu spitz, sodass das Material der jeweils nächsten Schicht ohne die Stützstrukturen nicht zuverlässig an der darunterliegenden haften könnte. Stützstrukturen können ebenfalls dazu verwendet werden, den Verzug des Bauteils während des Prozesses gering zu halten. Verzug kann unter anderem zu einem Scheitern des Prozesses führen, falls die Düse mit bereits verfestigten Bereichen kollidiert. Dies geschieht bspw., wenn sich das Bauteil von der Bauplattform ablösen sollte, was ebenfalls durch die Basis- und Stützstrukturen verhindert werden kann [53]. Die Art, Platzierung und Größe der Basis- und Stützstrukturen müssen ebenfalls im Slicer festgelegt werden. Der Pfad wird zusammen mit allen anderen für die Maschine relevanten Parametern als *G-Code*-Programm (*geometric Code*) ausgegeben und kann auf der MEX-Maschine ausgeführt werden [1, 50]. In Abbildung 2.2 ist als einfaches Beispiel ein Würfel nach dem Slicing gezeigt. In diesem Fall ist für eine bessere Anhaftung an die Bauplattform ein Rand (engl. *brim*) als Basisstruktur hinzugefügt (türkis). Die äußere Wand besteht aus zwei Lagen (rot und grün), das Innere wurde als gitterförmiges Infill ausgeführt (orange).

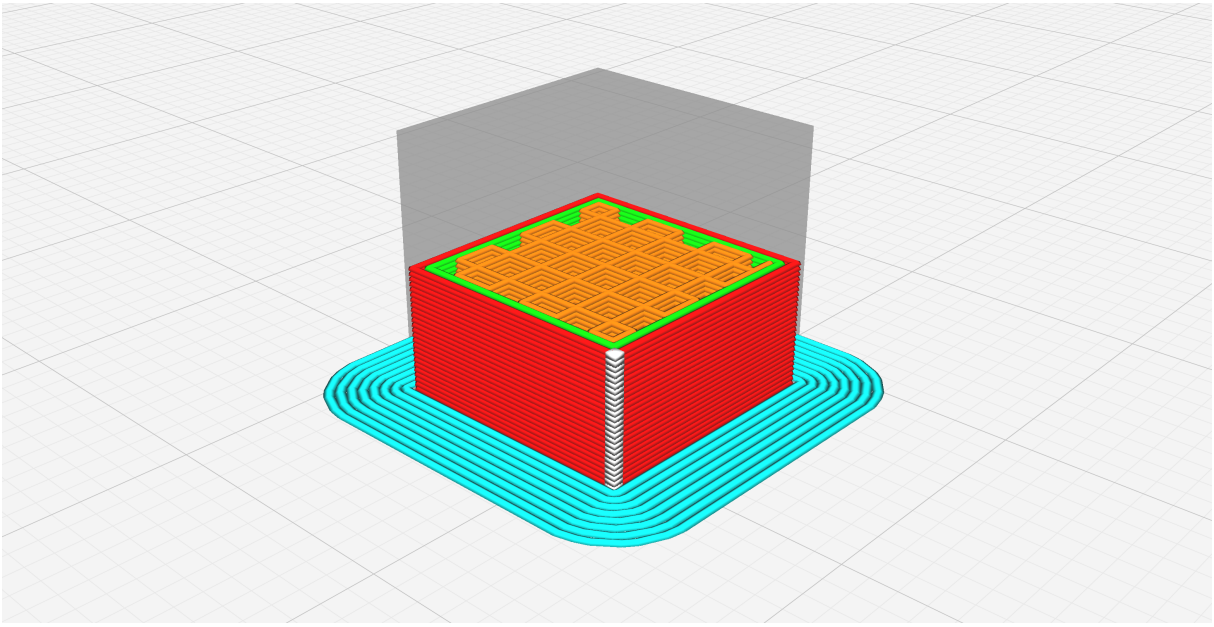
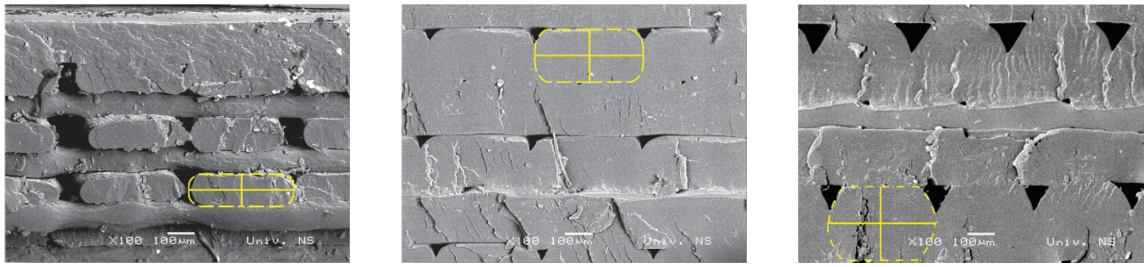


Abb. 2.2: Schnitt durch einen Würfel nach dem Slicing.

2.1.3 Eigenschaften des Verfahrens

Die MEX erlaubt im Vergleich zu anderen Fertigungsverfahren eine sehr hohe Gestaltungsfreiheit. Auch Kavitäten oder Hinterschneidungen sind grundsätzlich möglich, insofern Überhänge zur Bauplattform hin abgestützt werden können. Um diese Möglichkeiten bestmöglich ausnutzen zu können, bietet sich der Einsatz von Gestaltungsmethoden wie der Topologieoptimierung oder dem *Generative Design* besonders an. So können Bauteile hoher Komplexität gefertigt werden, die besonders gut an den jeweiligen Lastfall angepasst sind. Daraus leitet sich das hohe Leichtbaupotential und der geringe Materialverbrauch des Verfahrens ab [6]. Weiter werden für die Fertigung keinerlei bauteilspezifische Werkzeuge benötigt. Dadurch ist das Verfahren für kleine Losgrößen wirtschaftlich. Komplexe Geometrien mit hoher Funktionsintegration sind nicht aufwändiger zu fertigen. Die Gestalt der Bauteile kann ohne neue Werkzeuge angepasst werden, was schnelles Iterieren oder individuelle Anpassungen erlaubt [1, 2]. Nachteilig sind insbesondere die im Vergleich zu anderen Verfahren, wie etwa dem Spritzgießen, hohen Taktzeiten des Verfahrens. Diese sind insbesondere durch den Volumenstrom, den die Schmelzeinheit bereitstellen kann und die maximale Vorschubgeschwindigkeit begrenzt. Um die Auflösung des Verfahrens zu erhöhen, kann der Durchmesser der Düse verringert werden. Damit einher geht aber ein geringerer Volumenstrom, was wiederum die Taktzeiten weiter erhöht [2]. Auch für kleine Düsendurchmesser lässt sich oftmals keine ausreichende Oberflächenqualität erreichen, da die einzelnen Stränge bzw. Schichten an der Bauteiloberfläche deutlich zu erkennen bleiben. Insbesondere an Schrägen führt der schichtweise Aufbau zum *Treppeneffekt*, also sichtbaren Stufen anstatt einer ebenen Oberfläche. Die Oberflächenqualität kann ggf. durch Nachbearbeitung verbessert werden, dennoch ist die MEX dadurch für bestimmte Anwendungen, für die die Oberflächenqualität zur Funktionserfüllung oder aus optischen Gründen ausschlaggebend ist, weniger gut geeignet [18].

Die nebeneinander abgelegten Stränge werden zwar teilweise verschweißt, allerdings bleibt das entstehende Bauteil porös. Abhängig von den Prozessparametern, wie dem eingesetzten Material, Düsendurchmesser, Schichtdicke, Abstand der Stränge oder Temperatur, und dem Muster, nach dem die Schichten aufgebaut werden, bildet sich eine spezifische Mesostruktur aus. Die Beschaffenheit der Mesostruktur hängt von einer Vielzahl an Parametern ab, die teilweise gewählt werden können, teilweise durch die verwendete Maschine oder die Umgebung festgelegt sind [18, 19]. In Abbildung 2.3 sind Mikroskopaufnahmen dreier Mesostrukturen gezeigt, für die jeweils die Schichtdicke variiert wurde. Die Porosität und die Größe der miteinander verschweißten Flächen unterscheiden sich deutlich. In der Arbeit von Luzanin et al. [54] wird ein starker Zusammenhang zwischen der Schichtdicke und der Zugfestigkeit der so hergestellten Proben festgestellt. Die homogenisierten mechanischen Eigenschaften durch MEX gefertigter Strukturen sind anisotrop und im Vergleich zu anderen Fertigungsverfahren geringer. Die



(a) (b) (c)

Abb. 2.3: Mikroskopaufnahmen im FFF-Verfahren erzeugter Mesostrukturen für die Schichtdicken (a) $d_S = 0{,}1\,\mathrm{mm}$; (b) $d_S = 0{,}2\,\mathrm{mm}$; (c) $d_S = 0{,}3\,\mathrm{mm}$. In Gelb ist jeweils der Querschnitt eines Strangs markiert. Aus Luzanin et al. [54].

Steifigkeit und Festigkeit ist in Längsrichtung der Stränge deutlich höher als in Querrichtung. Diese Anisotropie lässt sich durch die Ausrichtung der Stränge in der Ebene reduzieren, entlang der Hochachse prinzipbedingt jedoch nicht. Auch andere Materialeigenschaften, wie z. B. die Wärmeleitfähigkeit, weißen diese Anisotropie auf. [2]

2.2 Modellierung der Materialextrusion

Numerische Methoden werden im Zusammenhang mit der MEX häufig in Struktursimulationen zur Auslegung von Bauteilen eingesetzt. Auch der Fertigungsprozess wird modelliert, um Verzug und Eigenspannungen, aber auch um Einflussfaktoren einzelner Druckparameter zu untersuchen und so Gestaltungsregeln abzuleiten bzw. das Prozessverständnis zu verbessern.

2.2.1 Struktursimulation

Struktursimulationen dienen dazu, Festigkeiten und Steifigkeiten der fertigen Bauteile zu bestimmen und so ihre Funktionserfüllung zu gewährleisten [8–14]. Die mechanischen und thermischen Eigenschaften des Bauteils werden maßgeblich von der jeweiligen Mesostruktur bestimmt. Um deren Einfluss zu berücksichtigen, kann die Geometrie der einzelnen Stränge durch mehrere über den Querschnitt verteilte Zellen vollständig aufgelöst werden. Diesen Ansatz verfolgen bspw. Garg und Bhattacharya [9] für die Struktursimulation von relativ kleinen Materialproben, wie sie in Zugversuchen eingesetzt werden. Die Modellierung größerer Bauteile auf diese Art ist aufgrund der sehr großen Zahl benötigter Zellen nicht praktikabel. Üblicherweise werden in Struktursimulationen die einzelnen Stränge deshalb nicht aufgelöst. Stattdessen wird die Mesostruktur homogenisiert und die Geometrie gröber vernetzt. [8, 10–14, 23, 55–57] Die homogenisierten Eigenschaften der Mesostruktur können durch Experimente, theoretische

Überlegungen oder multiskalige Simulationen bestimmt werden. Martínez et al. [8], Vidakis et al. [11] sowie Hossein Ehsani et al. [13] bestimmen die für Struktursimulationen benötigten Eigenschaften experimentell. Da die Menge möglicher Parameterkombinationen sehr groß ist und schon in einem Bauteil mehrere Kombinationen auftreten können, ist diese Art der Charakterisierung mit hohem Aufwand verbunden. Deutlich schneller und flexibler ist die Charakterisierung durch theoretische Überlegungen bzw. mithilfe empirischer Zusammenhänge, wie sie etwa von Li et al. [23] und Völkl et al. [12] durchgeführt wird. Voraussetzung dafür ist allerdings eine Vereinfachung der Mesostruktur, bzw. das Vorliegen gültiger empirischer Ansätze. Eine dritte Möglichkeit zur Homogenisierung der Mesostruktur sind mehrskalige Simulationsmodelle wie die von Somireddy und Czekanski [10] oder Tessarin et al. [14]. Hierzu wird für die Meso- und Makroebene jeweils ein Simulationsmodell erstellt. Mithilfe des Mesomodells können die homogenisierten Kennwerte bestimmt werden, die für das Makromodell benötigt werden. Die Geometrie der Mesostruktur muss auch hier bereits bekannt sein. Ggf. kann dieser mehrskalige Ansatz um die Mikroebene erweitert werden. Sollen bspw. FVK eingesetzt werden, können einzelne Fasern in repräsentativen Volumenelementen (RVE) abgebildet werden. [14]

2.2.2 Prozesssimulation auf Makroebene

Durch den MEX-Prozess treten, insbesondere aufgrund thermischer Schwindung, Eigenspannungen und Verzug im Bauteil auf [1, 2]. Diese können dazu führen, dass Lage- und Formtoleranzen nicht eingehalten werden oder der Prozess scheitert, etwa durch Ablösen des Bauteils von der Bauplattform [58, 59]. Neben den Struktursimulationen wird daher auch der Fertigungsprozess modelliert, um thermomechanische Effekte vorherzusagen. Soll der Prozess für ein ganzes Bauteil abgebildet werden, muss auch hier die räumliche Auflösung so gewählt werden, dass die einzelnen Stränge nicht mehr abgebildet werden. Der Prozess dauert zudem meist mehrere Stunden, was den Rechenaufwand weiter erhöht. Das Fließen der Polymerschmelze aus der Düse auf die Bauplattform wird ebenfalls nicht modelliert. Stattdessen wird aus der Bauteilgeometrie ein Netz erstellt, dessen Zellen nach und nach, dem Werkzeugpfad folgend, aktiviert werden. [15–17, 40–45, 60] Der Einfluss der Mesostruktur kann analog zur Struktursimulation durch experimentelle Charakterisierung, bekannte analytische Zusammenhänge oder mehrskalige numerische Modelle berücksichtigt werden.

2.2.3 Prozesssimulation auf Mesoebene durch FEM und FVM

Die Bestimmung homogenisierter thermomechanischer Eigenschaften durch Simulationsmodelle ist für einen großen Parameterraum weniger aufwändig als die Durchführung von Experimenten und kann eingesetzt werden, wenn keine gesicherten analytischen Zusammenhänge

vorliegen. Voraussetzung für dieses Vorgehen ist allerdings, dass die Gestalt der Mesostruktur bekannt ist. Anstatt lediglich idealisierte Geometrien anzunehmen, kann der Prozess für ein relativ kleines RVE numerisch modelliert werden. Die daraus erhaltenen Strukturen können die Genauigkeit der Charakterisierung ohne Experimente erhöhen. Darüber hinaus können die Modelle selbst zum allgemeinen Prozessverständnis beitragen. An ihnen kann der Einfluss verschiedener Fertigungsparameter auf den Prozess und die entstehende Struktur untersucht werden. Zur Modellierung werden verschiedene numerische Methoden eingesetzt: Ähnlich zur Prozesssimulation auf Makroebene verwenden Brenken et al. [61] die FEM Methode zur Untersuchung der Kristallisation und dem erneuten Aufschmelzen des Polymers. Die Stränge werden als ideale Zylinder angenommen. Die Zellen werden schrittweise aktiviert, das Fließen des Polymers wird nicht abgebildet. Andere Modelle basieren auf der FVM [27–33]. Mishra et al. [33] modellieren das Aufschmelzen des Filaments und die Strömung durch die Schmelzeinheit und Düse. Das so berechnete Geschwindigkeitsprofil wird als Einlass-Randbedingung an die Simulation der Ablage eines einzelnen Stranges weitergegeben. Die Viskosität wird nach dem Cross-WLF-Ansatz [62] unter Berücksichtigung von Temperatur und Scherrate berechnet. Die verwendete Zustandsgleichung ist druck- und temperaturabhängig. Hesse et al. [32] betrachten ebenfalls die Ablage eines einzelnen Strangs. Das Simulationsgebiet beginnt unmittelbar stromauf des Düsenaustritts, die Schmelzeinheit wird also nicht betrachtet. Die Viskosität wird ebenfalls mit dem Cross-WLF-Ansatz berechnet. Die taitsche Gleichung gibt den Zusammenhang zwischen Temperatur, Druck und Volumen an [63–65]. Die Temperaturverteilung des Strangs wird mit Infrarotaufnahmen aus einem Experiment verglichen. In den Experimenten kühlt der Strang deutlich langsamer ab als vom Modell vorhergesagt. Die Geometrie des Strangs wird hingegen sehr genau vorhergesagt. Mehrere Stränge werden von Serdeczny et al. [31] simuliert. Die Vorgänge in der Schmelzeinheit werden auch hier nicht betrachtet. Nachdem ein Strang abgelegt ist, wird aus dem Querschnitt eine in Vorschubrichtung konstante Geometrie für eine Randbedingung abgeleitet. Bei der Ablage des jeweils nächsten Strangs werden die zuvor abgelegten also nur als Randbedingung berücksichtigt. Viskosität und Dichte werden als konstant angenommen. Ein erneutes Aufschmelzen der Stränge ist dadurch nicht möglich, das Vorgehen spart jedoch Rechenzeit. Die Ablage erfolgt stets in dieselbe Richtung und in mehreren Schichten. Der Querschnitt weißt eine hohe Übereinstimmung mit aus Experimenten gewonnenen Schliffen auf. Flexibler ist das Modell von Xia et al. [27–30]: Hier werden mehrere Stränge nacheinander abgelegt und bleiben über den gesamten Prozess Teil der Simulation. Dadurch kann das Modell das erneute Aufschmelzen des Polymers abbilden. Das Simulationsgebiet beginnt auch hier kurz vor dem Düsenausgang. Die Berechnung der Viskosität erfolgt durch das Cross-WLF-Modell und die Dichte ist temperaturabhängig. Es werden einfache Muster aus über- und nebeneinanderliegenden Strängen sowie Spiralen erstellt, für die u. a. die Temperaturverteilung und Eigenspannungen berechnet werden. In allen hier genannten FVM-basierten

Modellen wird die Strömung als Zweiphasenströmung aus dem Polymer und Luft mithilfe der VOF-Methode berechnet.

2.2.4 Prozesssimulation auf Mesoebene durch SPH

Wie die FVM eignet sich auch die SPH Methode, um das Fließen des Polymers auf die Bauplattform zu modellieren. Die Methode wird in Abschnitt 2.3 erläutert. Makino et al. [34] beschreiben ein dreidimensionales Modell, mit dem mehrere Stränge übereinander abgelegt werden. Die Integration erfolgt nach einem impliziten Schema, das Inkompressibilität gewährleistet. Die Temperaturverteilung wird nicht betrachtet. Viskosität und Dichte werden als konstant angenommen. Von den hier vorgestellten SPH-Modellen ist dies das einzige dreidimensionale. In den Folgenden wird das Problem zweidimensional betrachtet. Das Modell von Yang et al. [26] beschreibt die Schmelze mit der SPH Methode. Dazu werden diskrete Fasern mit der Diskrete-Elemente-Methode (DEM) modelliert und mit der Schmelze gekoppelt. Da das Verhalten diskreter Fasern untersucht wird, kann die Arbeit auch den Mikromodellen zugeordnet werden. Für eine Zuordnung zu den Mesomodellen spricht die Größe des Simulationsgebietes, das wenige Millimeter über dem Düsenaustritt beginnt. Es wird keine Einlassrandbedingung, sondern eine festgelegte Menge Partikel, die durch eine Wand aus der Düse geschoben wird, verwendet. Nur diese relativ kleine Zahl an Partikeln wird als kurzer Strang auf die Bauplattform aufgetragen. Viskosität, Temperatur und Dichte werden als konstant angenommen. Bertevas et al. [35] beschreiben die Fasern nicht diskret, sondern durch einen Faserorientierungstensor. Das Simulationsgebiet beginnt auch hier kurz vor dem Düsenausgang und es werden kurze Stränge abgelegt. Ouyang et al. [36] erweitern das Modell um die Temperatur sowie Scherraten- und Temperaturabhängigkeit der Viskosität. Die Wärmeleitfähigkeit des Materials wird von der Faserorientierung abhängig anisotrop modelliert. Außerdem betrachten sie die Ablage einer zweiten Schicht. Die Dichte wird als konstant angenommen. Bisher liegt noch kein SPH-Modell der MEX vor, mit dem nicht nur einzelne Stränge, sondern Strukturen aus mehreren schichtweise angeordneten Strängen erstellt werden und gleichzeitig der Temperaturverlauf und temperaturabhängige Materialwerte betrachtet werden können.

2.3 Smoothed Particle Hydrodynamics

Die Smoothed Particle Hydrodynamics (SPH) Methode ist ein unabhängig von Lucy [47] sowie von Gingold und Monaghan [46] entwickeltes gitterfreies Verfahren zur Lösung partieller Differentialgleichungen. Die Integrationspunkte, an denen die Simulationsgrößen ausgewertet werden, sind also nicht verbunden und bilden keine Zellen, wie es etwa in der Finite-

Elemente-Methode (FEM) oder Finite-Volumen-Methode (FVM) der Fall ist. Das Vernetzen des Simulationsgebietes entfällt. Zunächst für Probleme aus der Astrophysik angewandt, wird die Methode bis heute weiterentwickelt und für eine Vielzahl unterschiedlichster Anwendungen eingesetzt [66]. Die Erhaltungsgleichungen liegen in ihrer lagrangeschen Formulierung vor. Die Bestimmung der Simulationsgrößen an einer der beweglichen Stützstellen, die auch als Partikel bezeichnet werden, erfolgt durch Summation über benachbarte Partikel. Der Partikel, für den die Simulationsgrößen ausgewertet werden, wird *Zentralpartikel* genannt. Die jeweilige Gewichtung der Nachbarn ist durch eine sogenannte Glättungsfunktion festgelegt. In Abbildung 2.4 ist eine mögliche Glättungsfunktion über einem Partikelfeld dargestellt. Die Werte von Partikeln mit geringerem Abstand zum Zentralpartikel gehen mit größerem Gewicht in die Interpolation des Wertes des Zentralpartikels ein.

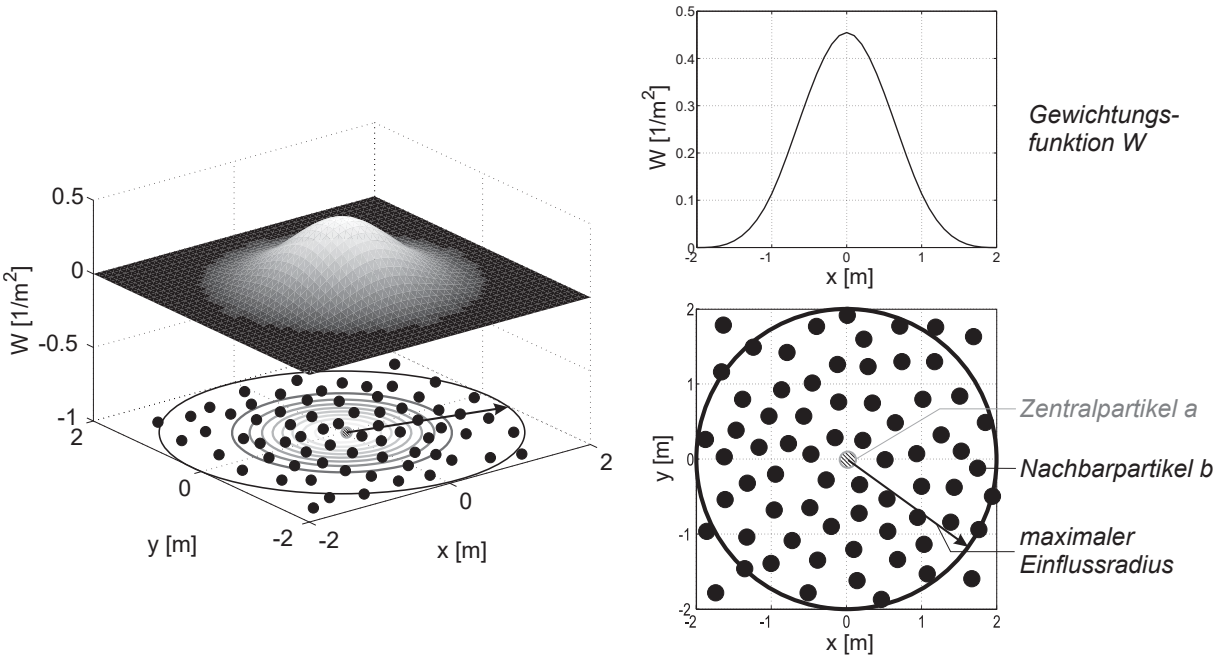


Abb. 2.4: Interpolation mittels Glättungsfunktion. Links: Die Glättungsfunktion über einem zweidimensionalen Partikelfeld. Rechts oben: Ein Schnitt durch die Glättungsfunktion. Rechts unten: Draufsicht auf das Partikelfeld. Aus Höfler [67].

2.3.1 SPH-Formalismus

Alle Simulationsgrößen wie Geschwindigkeit, Dichte, Druck, Temperatur usw. sind ortsabhängige Funktionen. Eine beliebige Funktion im Raum $f(\boldsymbol{x})$ kann als Faltung mit der Dirac-Funktion δ dargestellt werden:

$$f(\boldsymbol{x}) = \int f\left(\boldsymbol{x}'\right) \delta\left(\boldsymbol{x} - \boldsymbol{x}'\right) \mathrm{d}\boldsymbol{x}'. \tag{2.1}$$

Die Dirac Funktion kann näherungsweise durch eine *Glättungsfunktion* $W(\boldsymbol{x} - \boldsymbol{x}', h)$ ersetzt werden, die sich für $h \to 0$ der Dirac-Funktion annähert. Die *Glättungslänge* h beschreibt die Breite der Funktion und das Produkt $q_{\max} \cdot h$ ihren maximalen Einflussradius. Die Glättungslänge wird häufig als Vielfaches des *Partikelabstands* Δx (auch *Partikeldurchmesser*) definiert. Die Konstante $q_{\max}$ ist von der gewählten Glättungsfunktion abhängig. Die Anforderungen an eine solche Funktion werden in Abschnitt 2.3.2 beschrieben. Durch das Ersetzen der Dirac-Funktion ergibt sich die integrale Interpolation von f über benachbarte Partikel:

$$f_{\mathrm{i}}(\boldsymbol{x}) \approx \int f\left(\boldsymbol{x}'\right) W\left(\boldsymbol{x} - \boldsymbol{x}', h\right) \mathrm{d}\boldsymbol{x}'. \tag{2.2}$$

Zur numerischen Berechnung der Funktionswerte an den Stützstellen wird die integral interpolierte Funktion in eine diskrete Summeninterpolation überführt. Größen, die dem Zentralpartikel zugeordnet sind, werden mit dem Index a versehen. Größen der benachbarten Partikel erhalten den Index b. Die Summeninterpolation f_{S} ergibt sich zu

$$\begin{aligned} f_{\mathrm{s}}\left(\boldsymbol{x}_a\right) &\approx \sum_b V_b f\left(\boldsymbol{x}_b\right) W\left(\boldsymbol{x}_a - \boldsymbol{x}_b, h\right) \\ &= \sum_b \frac{m_b}{\rho_b} f\left(\boldsymbol{x}_b\right) W\left(\boldsymbol{x}_a - \boldsymbol{x}_b, h\right). \end{aligned} \tag{2.3}$$

V bezeichnet das Volumen, das auch durch die Masse m zusammen mit der Dichte ρ ausgedrückt werden kann.

2.3.2 Glättungsfunktionen

Für eine übersichtlichere Notation wird der Verbindungsvektor $\boldsymbol{d} = \boldsymbol{x} - \boldsymbol{x}'$ eingeführt. Damit Gleichung 2.2 mathematisch exakt ist, muss die Glättungsfunktion folgende Bedingungen erfüllen [67]:

Normierung: $\int W\left(\boldsymbol{x} - \boldsymbol{x}', h\right) \mathrm{d}\boldsymbol{x}' = 1$

Kompaktheit: $W\left(\boldsymbol{d}, h\right) = 0 \qquad \text{für } |\boldsymbol{d}| \geq q_{\max} h$

Grenzwertbetrachtung: $\lim\limits_{\mathrm{h} \to 0} W\left(\boldsymbol{d}, h\right) = \delta\left(\boldsymbol{d}\right).$

Diese Bedingungen alleine stellen allerdings nicht sicher, dass das physikalische Verhalten des Kontinuums korrekt abgebildet wird. Liu et al. [68] und Höfler [67] nennen dazu zusätzlich die folgenden Bedingungen:

Positivität: $\int W\left(\boldsymbol{x} - \boldsymbol{x}', h\right) \mathrm{d}\boldsymbol{x}' \geq 0 \qquad \text{für } |\boldsymbol{x} - \boldsymbol{x}'| \leq q_{\max} h$

Streng fallende Monotonie: $W_1\left(\boldsymbol{d}_1, h\right) > W_2\left(\boldsymbol{d}_2, h\right) \qquad \text{für } |\boldsymbol{d}_1| < |\boldsymbol{d}_2|$

Symmetrie: $$W\left(\boldsymbol{x}-\boldsymbol{x}',h\right)=W\left(\boldsymbol{x}'-\boldsymbol{x},h\right)$$

Stetigkeit: $$\lim_{\boldsymbol{d}\to\boldsymbol{d}_0} W\left(\boldsymbol{d},h\right)=\delta\left(\boldsymbol{d}_0\right).$$

Außerdem muss eine Glättungsfunktion mit einem dimensionsabhängigen Vorfaktor α_D normiert werden, um unabhängig von der Dimension des Problems eingesetzt werden zu können. Die Funktionen werden in Abhängigkeit des dimensionslosen Abstands $q=\frac{|\boldsymbol{d}|}{h}$ definiert. Ein Beispiel für eine Glättungsfunktion ist die durch Monaghan [69] formulierte kubische Funktion:

für $0\leq q\leq 1$:

$$W(q,h)=\alpha\left(\frac{2}{3}-q^2+\frac{1}{2}q^3\right) \tag{2.4}$$

$$\nabla W(q,h)=\alpha\left(-2q+\frac{3}{2}q^2\right)\frac{\boldsymbol{d}}{|\boldsymbol{d}|\,h} \tag{2.5}$$

für $1<q\leq 2$:

$$W(q,h)=\alpha\frac{1}{6}\left(2-q\right)^3 \tag{2.6}$$

$$\nabla W(q,h)=-\frac{1}{2}\alpha\left(2-q\right)^2\frac{\boldsymbol{d}}{|\boldsymbol{d}|\,h} \tag{2.7}$$

für $2<q$:

$$W(q,h)=0 \tag{2.8}$$

$$\nabla W(q,h)=0. \tag{2.9}$$

Der maximale Einflussradius der kubischen Glättungsfunktion ist also $q_\mathrm{max}=2h$. Der Vorfaktor α muss je nach Dimensionalität des Problems zu

$$\alpha_\mathrm{1D}=\frac{1}{h},\quad \alpha_\mathrm{2D}=\frac{15}{7\pi h^2}\quad\text{oder}\quad \alpha_\mathrm{3D}=\frac{3}{2\pi h^3} \tag{2.10}$$

gewählt werden. Ein relativ zum Partikelabstand größerer Einflussradius reduziert den Fehler der Approximation. Allerdings werden dadurch auch mehr Partikel in die Berechnung miteinbezogen, wodurch der Rechenaufwand steigt. [67]

Je nach Problemstellung kann aus einer Vielzahl von Glättungsfunktionen ausgewählt werden. Eine Darstellung möglicher Funktionen sowie eine Diskussion über die Auswirkungen der Wahl einer Funktion finden sich z. B. in der Arbeit von Höfler [67].

Zur besseren Lesbarkeit werden im Folgenden interpolierte Größen nur noch mit dem Variablennamen und dem Partikelindex benannt, für interpolierte Größen wird $=$ anstelle von $\approx$ geschrieben und die Glättungsfunktion wird nicht mehr explizit als Funktion von h notiert. So gelten beispielsweise

$$f_\mathrm{a}=f_\mathrm{s}\left(\boldsymbol{x}_a\right), \tag{2.11}$$

$$\boldsymbol{x}_{\mathrm{ab}} = \boldsymbol{x}_a - \boldsymbol{x}_b \tag{2.12}$$

und

$$W_{\mathrm{ab}} = W\left(\boldsymbol{x}_a - \boldsymbol{x}_b, h\right). \tag{2.13}$$

2.3.3 Gradient und Laplace-Operator

Um Gleichungen, die das Kontinuum beschreiben, in den SPH-Formalismus überführen zu können, muss der Gradient ∇f_a approximiert werden. Er ist bspw. in den Navier-Stokes-Gleichungen enthalten. Der überführte Gradient ist [48, 49, 67]

$$\Phi_{\mathrm{a}} \nabla f_a = \sum_b m_b \frac{\Phi_b}{\rho_b} \left(f_b - f_a\right) \nabla W_{ab}. \tag{2.14}$$

Wobei Φ eine differenzierbare Funktion ist. Wird $\Phi = 1$ gewählt, so ergibt sich

$$\nabla f_a = \sum_b \frac{m_b}{\rho_b} \left(f_b - f_a\right) \nabla W_{ab}. \tag{2.15}$$

$\Phi = \rho$ liefert hingegen

$$\rho_a \nabla f_a = \sum_b m_b \left(f_b - f_a\right) \nabla W_{ab}. \tag{2.16}$$

Darüber hinaus wird teilweise auch die Divergenz des Gradienten Δf_{a} benötigt, z. B. für die Wärmeleitungsgleichung. In der Literatur wird diese statt mit dem Laplace-Operator häufig als $\nabla \cdot \nabla f_{\mathrm{a}}$ oder $\nabla^2 f_{\mathrm{a}}$ geschrieben. Brookshaw [70] schlägt

$$\Delta f_a = \sum_b 2 \frac{m_b}{\rho_b} \left(f_a - f_b\right) \frac{\boldsymbol{x}_{ab}}{\left|\boldsymbol{x}_{ab}\right|^2} \nabla W_{ab}. \tag{2.17}$$

für eine simple Approximation des Laplace-Operators vor. Es existieren weitere Formulierungen, die den Fehler der Lösung reduzieren können, z. B. von Schwaiger [71] oder Fatehi und Manzari [72]. Diese erhöhen allerdings den benötigten Rechenaufwand.

2.3.4 SPH Schemata

Abhängig vom jeweiligen Problem, das mit der SPH Methode untersucht werden soll, müssen unterschiedliche Gleichungen zugrunde gelegt werden. Für die meisten Probleme müssen die Erhaltungsgleichungen ausgewertet werden. Jedoch gibt es auch für deren Formulierung und Lösung zahlreiche Ansätze. In der Literatur wird ein solcher Ansatz häufig als *Schema* (engl. *scheme*) bezeichnet [49, 73]. Neben den zu lösenden Gleichungen müssen die Integrationsvorschrift und die Glättungsfunktion ausgewählt werden. Manche Schemata unterscheiden sich nur in

einigen Punkten voneinander, bzw. bauen aufeinander auf, andere wiederum unterscheiden sich grundsätzlich. In der vorliegenden Arbeit wird das Open-Source-Paket PySPH verwendet. Hier werden Klassen als Schemata bezeichnet, die den Einsatz verschiedener Berechnungsschemata vereinfachen sollen. Diese enthalten die benötigten Gleichungen, Integrationsvorschriften und Glättungsfunktion. Häufig enthalten sie darüber hinaus Funktionen zur adaptiven Berechnung des Zeitschritts oder Anpassung weiterer Parameter, die für den Einsatz des jeweiligen Schemas notwendig oder hilfreich sind. Im Folgenden werden drei Berechnungsschemata kurz vorgestellt.

Das geringfügig kompressible SPH Schema (WCSPH)

Bei den geringfügig kompressiblen (engl. *weakly compressible*) Formulierungen handelt es sich um eine Gruppe von Schemata. Zentrales Merkmal ist, dass das Medium zwar als inkompressibel angenommen werden soll, die Inkompressibilität aber nicht streng umgesetzt wird. Stattdessen wird über eine Zustandsgleichung (engl. *equation of state* kurz *EOS*) ein Zusammenhang zwischen der Dichte, dem Druck und ggf. weiteren Zustandsgrößen des Mediums hergestellt. Monaghan [74] wählt dafür die taitsche Gleichung [63, 64]

$$p_\mathrm{a} = \frac{c_\mathrm{s}^2 \rho_0}{\gamma} \left(\left(\frac{\rho_a}{\rho_0} \right)^\gamma - 1 \right). \tag{2.18}$$

Eine gegenüber einem Referenzwert ρ_0 erhöhte Dichte führt zu einer Erhöhung des Drucks und somit zu einer Beschleunigung. Werden die Parameter der EOS passend gewählt, bleiben die Dichteunterschiede vernachlässigbar klein. Für Medien, die Wasser ähnlich sind, wird der Exponent $\gamma = 7$ gewählt. Der zentrale Parameter ist eine künstliche Schallgeschwindigkeit c_s, die die Kompressibilität des Mediums festlegt. Es handelt sich dabei um eine numerische Größe, die nicht mit der physikalischen Schallgeschwindigkeit gleichzusetzen ist. Das Schema von Monaghan [74] ist der klassische Vertreter dieser Gruppe, mit dem die SPH-Methode erstmals für Strömungen mit freien Oberflächen eingesetzt werden konnte. Im Folgenden wird dieses Schema als *Weakly Compressible SPH* (WCSPH) bezeichnet, auch wenn es nicht das einzige aus dieser Gruppe ist. Diese Benennung ist konsistent mit der in PySPH [73]. Die Umsetzung in PySPH enthält außerdem optional die Korrektur gegen Zug-Instabilität von Monaghan [75], die *δ-SPH* Korrektur von Marrone et al. [76] und eine für Wände korrigierte taitsche Gleichung von Hughes und Graham [77].

Transportgeschwindigkeits-Schema (TVF, AHA & GTVF)

Das *Transportgeschwindigkeits-Schema* (engl. *Transport Velocity Formulation* kurz TVF) nach Adami et al. [78] ist ebenfalls ein geringfügig kompressibles Schema. Es unterscheidet sich vom klassi-

schen WCSPH durch die Einführung einer *Advektionsgeschwindigkeit* $\tilde{\boldsymbol{u}}_\mathrm{a}$, mit der die Partikel bewegt werden

$$\tilde{\boldsymbol{u}}_a(t+\delta t) = \boldsymbol{u}_a(t) + \delta t \left(\frac{\mathrm{d}\tilde{\boldsymbol{u}}_a}{\mathrm{d}t} - \frac{1}{\rho_a} \nabla p_\mathrm{H} \right). \tag{2.19}$$

$\tilde{\boldsymbol{u}}_\mathrm{a}$ wird nicht nur von der mithilfe der Impulsbilanz berechneten Geschwindigkeit $\boldsymbol{u}$ bestimmt, sondern zusätzlich von einem künstlichen *Hintergrunddruck* p_H. Mit der von Adami et al. [78] gewählten Diskretisierung des Laplace-Operators ergibt sich

$$\tilde{\boldsymbol{u}}_a(t+\delta t) = \boldsymbol{u}_a(t) + \delta t \left(\frac{\mathrm{d}\tilde{\boldsymbol{u}}_a}{\mathrm{d}t} - \frac{p_\mathrm{H}}{m_a} \left({V_a}^2 + {V_b}^2 \right) \nabla W_{ab} \right). \tag{2.20}$$

Der zusätzliche Term bleibt somit für einen homogenen Hintergrunddruck $\neq 0$. Der Ansatz soll insbesondere das Entstehen von *Klumpen* aus Partikeln bzw. numerisch induzierte Fehlstellen verhindern, wie es im WCSPH zu beobachten ist. Allerdings erlaubt das TVF deshalb keine freien Oberflächen. Wie bei inneren Fehlstellen würden die Partikel in das leere Simulationsgebiet geschoben. [78] Zhang et al. [79] haben deshalb eine verallgemeinerte Form des TVF, das GTVF (engl. *Generalized Transport Velocity Formulation*) eingeführt, das die Verwendung eines Hintergrunddrucks auch für Strömungen mit freien Oberflächen erlaubt. In PySPH liegen drei Varianten des TVF vor:

- Das TVF wie von Adami et al. [78] beschrieben.
- Das AHA (kurz für *Adami, Hu & Adams*), das weitgehend dieselben Gleichungen wie das TVF nutzt. In dieser Version wird der Hintergrunddruck nicht berücksichtigt, weswegen auch die Integrationsvorschrift angepasst ist. Außerdem wird die Dichte durch die Kontinuitätsgleichung und nicht durch einfache Summation bestimmt.
- Das GTVF, das die Verwendung eines Hintergrunddrucks auch für Strömungen mit freien Oberflächen erlaubt [79].

Alle drei Varianten enthalten, anders als das WCSPH, eine Geschwindigkeitsrandbedingung für Wände von Festkörpern nach Adami et al. [80].

Schema der Entropie gedämpften künstlichen Kompressibilität (EDAC)

Anders als die zuvor genannten Schemata wird im EDAC (engl. Entropically Damped Artificial Compressibility) keine Zustandsgleichung gelöst. Das Schema basiert auf der Arbeit von Clausen [81] und wurde von Ramachandran und Puri [82] als SPH-Schema übernommen und in PySPH implementiert [73]. Die zeitliche Änderung des Drucks wird durch die *EDAC-Gleichung*

$$\frac{\mathrm{d}p}{\mathrm{d}t} = -\rho c_\mathrm{s}^2 \nabla \cdot \boldsymbol{u} + \nu \Delta p \tag{2.21}$$

bestimmt. Die Kompressibilität des Fluids wird auch hier über eine künstliche Schallgeschwindigkeit eingestellt.

2.3.5 SPH in ABAQUS

In der kommerziellen Software ABAQUS können das Integrationsschema und der verwendete Gleichungssatz nicht ausgewählt werden. Der Programmcode kann durch einen gewöhnlichen Nutzer weder eingesehen noch modifiziert werden. Es liegt nur eine nicht näher beschriebene, explizite SPH-Formulierung vor [83]. Einzig die Zustandsgleichung kann bei der Materialdefinition aus einer Liste ausgewählt werden [84]. Außerdem besteht die Möglichkeit, eigene Subroutinen zu formulieren [85]. Die Literaturhinweise in der Dokumentation [83] deuten darauf hin, dass sich die verwendete Formulierung an der klassischen geringfügig kompressiblen von Monaghan [74] orientiert. Außerdem wird auf einige Korrekturen verwiesen [75, 86–89].

3 Methodenentwicklung

3.1 Materialparameter

Der MEX-Prozess wird sowohl in ABAQUS als auch PYSPH modelliert. In beiden Fällen wird als Material für die Schmelze *Polymilchsäure* (engl. *polylactic acid*, kurz *PLA*) eingesetzt. Die Materialparameter für PLA in Tabelle 3.1 werden der Materialdatenbank der Software AUTODESK MOLDFLOW 2021 [90] entnommen.

Tab. 3.1: Materialparameter für die PLA-Schmelze.

Referenzdichte	ρ_0	1072,7	kg/m^3
Wärmeleitfähigkeit	κ	0,195	W/mK
Spezifische Wärmekapazität	c_p	2060	J/kgK
Emissionskoeffizient	ε	0,98	-
Konvektiver Wärmetransportkoeffizient	h_c	100	W/m^{2}K
Cross-WLF Koeffizienten	$\tau*$	100861	Pa
	n	0,25	-
	D_1	3,31719e9	Pa s
	D_2	373,15	K
	D_3	0	K/Pa
	A_1	20,194	-
	A_3	51,3	K

Zur Modellierung des konduktiven Wärmeübergangs zwischen Fluid und Bauplattform in PYSPH werden die Werte in Tabelle 3.2 verwendet [91].

Tab. 3.2: Materialparameter für legierten Stahl.

Dichte	ρ	7900	kg/m^3
Wärmeleitfähigkeit	κ	14	W/mK
Spezifische Wärmekapazität	c_p	510	J/kgK

Die neu in PySPH implementierte konduktive Wärmeleitung wird anhand eines Stabes aus Aluminium validiert. Die thermischen Eigenschaften von Aluminium sind in Tabelle 3.3 aufgeführt [91].

Tab. 3.3: Materialparameter für Aluminium.

Dichte	ρ	2700	kg/m^3
Wärmeleitfähigkeit	κ	204	W/mK
Spezifische Wärmekapazität	c_p	940	J/kgK

3.2 Modellierung in ABAQUS

Die kommerzielle Simulationssoftware ABAQUS ermöglicht seit 2017 SPH-Simulationen mit expliziter Integration [92]. Dabei können, wie auch für FEM-Modelle, sowohl die in der Software hinterlegten Materialmodelle als auch vom Benutzer formulierte Subroutinen eingesetzt werden. [83] Die Software wird am FAST-LB bereits vielfach zur Modellierung verschiedener Fertigungsprozesse für Polymere und FVK eingesetzt. Entsprechend liegen bereits zahlreiche Materialmodelle und Subroutinen vor. Es wäre wünschenswert, diese in ein Modell der MEX einbringen zu können, weshalb ein solches in ABAQUS erstellt wird.

Die Vorgänge in der Schmelzeinheit und der Düse werden zunächst nicht betrachtet. Der Aufbau ist in Abbildung 3.1 schematisch dargestellt. Zu Beginn der Simulation enthält die Domäne noch keine Partikel, sondern nur die Bauplattform und einen Partikeleinlass, der den Ausgang aus der Düse darstellt. Beide werden mit *R3D4*-Elementen vernetzt. Die Vernetzung der Geometrie ist notwendig, hat aber keinen Einfluss auf die Ergebnisse. Der Düse muss nun die ABAQUS interne Funktion *Partikelgenerator* zugewiesen werden. Der Elementtyp, der Partikelradius, das Material und eine maximale Partikelzahl müssen in der Funktion spezifiziert werden. Wird die maximale Partikelzahl während der Simulation überschritten, werden keine Partikel mehr generiert, die Simulation wird dadurch jedoch nicht gestoppt. SPH-Partikel werden in ABAQUS als *PC3D*-Elemente umgesetzt. Diese sind für Simulationen im dreidimensionalen Raum vorgesehen. Alternative 2D-Elemente gibt es nicht. Die Partikel besitzen zudem nicht

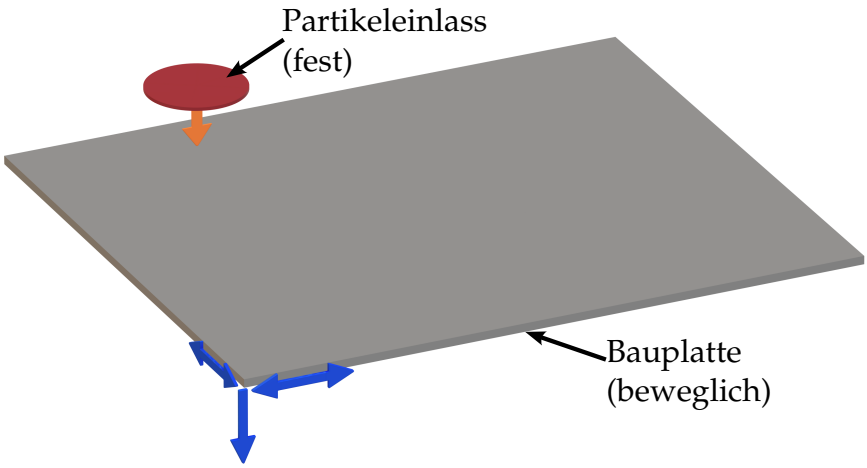


Abb. 3.1: Schematischer Aufbau der Simulation mit beweglicher Bauplattform (grau) und Partikelgenerator (rot).

den Freiheitsgrad *NT11*, der zur Betrachtung der Temperaturverteilung notwendig ist. Auf die Partikel wirkt eine Gravitationskraft g. [93] Anders als in der Dokumentation beschrieben [94], kann der Partikelgenerator nicht bewegt werden. Stattdessen wird die Bauplattform bewegt. Die Bewegung wird durch entsprechende Randbedingungen und Amplituden festgelegt. Da dies schon für einfache Muster aufwändig und fehleranfällig ist, werden diese durch ein MATLAB-Skript erstellt. Das Skript erlaubt das Ablegen eines einzelnen Strangs oder mehrerer Stränge neben- und übereinander mit konstanter Geschwindigkeit. Für den Partikelgenerator müssen eine Extrusionsgeschwindigkeit v_{Ex} und ein Massenstrom $\dot{m}$ definiert werden. Die beiden Größen werden aufeinander abgestimmt, sodass der Zusammenhang

$$\dot{m} = \rho v_{\mathrm{Ex}} A_{\mathrm{D}} \tag{3.1}$$

gilt. Die Extrusionsgeschwindigkeit ist zeitlich konstant. Im realen Prozess stellt sich über die Düse eine nicht konstante Geschwindigkeitsverteilung ein, da das Fluid an den Wänden haftet. Dieses Geschwindigkeitsprofil wird im Modell nicht berücksichtigt. Zu Beginn steht die Bauplattform still, bis die Partikel mit ihr in Kontakt kommen. Danach bewegt sich die Platte mit konstanter Vorschubgeschwindigkeit v_{Vor} und ändert dabei ggf. die Richtung. Außerdem wird die Extrusionsgeschwindigkeit v_{Ex}, mit der die Partikel die Düse verlassen, mit v_{Vor} gleich gesetzt. Für das Materialmodell der Schmelze wird das bereits in ABAQUS implementierte Modell ohne zusätzliche Benutzerroutinen verwendet. Als EOS wird die in ABAQUS verfügbare von Mie-Grüneisen-Gleichung in der linearen $U_{\mathrm{S}} - U_{\mathrm{P}}$-Form gewählt [95]. Im Kontakt zwischen Plattform und Fluid wird eine Haftbedingung in tangentiale Richtung vorgegeben, sodass die Geschwindigkeit der Partikel im Kontakt der Geschwindigkeit der Bauplattform

entsprechen. Zusätzlich wird die Trennung der Partikel von der Plattform nach dem Kontakt unterbunden. Die dynamische Viskosität der Schmelze η wird als konstant angenommen. Sie ist ausschlaggebend für den maximal zulässigen Zeitschritt der Simulation. Um die Rechenzeit möglichst gering zu halten, liegt sie mehrere Größenordnungen unter dem realen Wert für eine PLA-Schmelze. Die wesentlichen Modellparameter können Tabelle 3.4 in entnommen werden.

Tab. 3.4: Wesentliche Parameter für die Modellierung in ABAQUS.

Δx	v_{Vor} & v_{Ex}	c_{S}	d_{D}	η	g
0,05 mm	60 mm/s	0,01 m/s bis 1 m/s	0,4 mm	0,01 Pa s bis 1 Pa s	9,81 m/s^2

Die Dichte des PLA kann Tabelle 3.1 in Abschnitt 3.1 entnommen werden. Zunächst muss eine geeignete Parameterkonfiguration für das Modell gefunden werden. Dazu wird die Simulation mehrfach in verschiedenen Konfigurationen durchgeführt. Um die Rechenzeit gering zu halten, werden dabei einzelne, kurze Polymerstränge abgelegt (siehe Abschnitt 4.1.1). Sind geeignete Parameter gefunden, wird eine Struktur aus mehreren parallelen Strängen betrachtet (siehe Abschnitt 4.1.2).

3.3 Modellierung in PySPH

PySPH ist ein Open Source Framework für SPH Simulationen entwickelt von Ramachandran et al. [73]. Das Framework kann durch den Nutzer beliebig angepasst und der Code eingesehen werden. Zudem ist es stark modular aufgebaut. Gleichungen, Integratoren oder ganze Schemata können mit relativ geringem Aufwand ausgetauscht werden. Dadurch ist PySPH insbesondere im Kontext der Forschung attraktiv. Wie ABAQUS fehlen PySPH diverse Funktionen, die für die Modellierung der MEX notwendig sind. Dazu gehören insbesondere die Berechnung von Temperaturen und Wärmeleitung sowie temperaturabhängige Materialmodelle, aber auch die Bewegung des Partikeleinlasses in Strömungsrichtung. [96] Für die vorliegende Arbeit wird der frei verfügbare PySPH-Code [97] für die Modellierung der MEX erweitert und modifiziert. Alle Erweiterungen und Modifikationen sind in Anhang A aufgeführt.

3.3.1 Aufbau des Modells

Der Aufbau des Modells ist dem in Abschnitt 3.2 sehr ähnlich: Wie in Abbildung 3.2 gezeigt, sind auch hier zunächst keine Fluidpartikel in der Domäne. Zu Beginn existieren nur die Bauplattform und der Partikeleinlass. Die Partikel sind jeweils in einem Array zusammengefasst. Anders als in ABAQUS ist in PySPH das Bewegen des Düsenausgangs möglich. Die bestehende

Funktion für Partikeleinlässe wird modifiziert, um auch die Bewegung in Strömungsrichtung zu ermöglichen. In diesem Modell ist die Bauplattform daher fest und die Düse beweglich. Diese Konfiguration ist wünschenswert, da so das Fluid in Simulationen mit verringerter Gravitation oder Viskosität weniger zerfließt oder sich gegen die Bauplattform verschiebt. Die Bauplattform und der Partikeleinlass werden jeweils durch drei Schichten von Partikeln repräsentiert. Für die

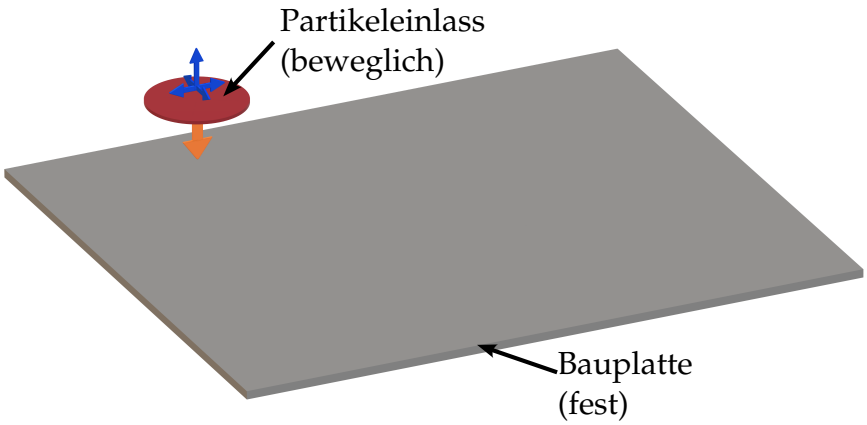


Abb. 3.2: Schematischer Aufbau der Simulation mit fester Bauplattform (grau) und beweglichem Partikelgenerator (rot).

Partikel der Düse und des Fluids werden dieselben Materialparameter für PLA verwendet wie für die Modellierung in Abaqus. Für die Wärmeleitung der Bauplattform werden für legierten Stahl typische Werte angenommen. Diese sind in Tabelle 3.2 aufgeführt.

3.3.2 SPH-Schemata

In PySPH lässt sich die SPH-Formulierung mit geringem Aufwand austauschen. Für das Modell wird wahlweise eines der folgenden in Abschnitt 2.3.4 erläuterten expliziten Schemata verwendet:

- WCSPH
- AHA
- GTVF
- EDAC.

Zusätzlich zu den bereits im jeweiligen Schema festgeschriebenen Gleichungen werden die in den folgenden Abschnitten aufgeführten Gleichungen, die zur Berechnung der Temperatur und Viskosität notwendig sind, gelöst.

3.3.3 Partikeleinlass

Die Einlass-Partikel werden mit der Vorschubgeschwindigkeit der Düse und zusätzlich mit der Extrusionsgeschwindigkeit in Strömungsrichtung bewegt. Diese Geschwindigkeit ist die Anfangsbedingung für die Partikel, sobald sie den Einlass verlassen und in die eigentliche Domäne eintreten. Der Einlass ist u. a. durch einen Referenzpunkt und einen Normalenvektor definiert. Die beiden Vektoren spannen eine Ebene auf. Durchdringt ein Partikel des Einlasses diese Ebene, wird er in einen Fluidpartikel umgewandelt. In PySPH ist die Bewegung des Einlasses grundsätzlich möglich, allerdings führt die Bewegung in Richtung des Normalenvektors nur zu einer erhöhten bzw. verringerten Extrusionsgeschwindigkeit. Aus diesem Grund muss der Einlass so angepasst werden, dass der Referenzvektor für jeden Zeitschritt unter Berücksichtigung von v_{Vor} und v_{Ex} angepasst wird. [98] Zusätzlich wird durch einen neu implementierten Algorithmus die Geschwindigkeit der Partikel im Einlass festgelegt, um die Stränge dem gewünschten Muster entsprechend abzulegen. Analog zum ABAQUS-Modell erlaubt diese Gleichung das Ablegen eines einzelnen Strangs oder mehrerer Stränge neben- und übereinander.

3.3.4 Temperatur und konduktive Wärmeübertragung

Den Partikeln sind zusätzliche Variablen für die Temperatur und deren Änderung zugewiesen. Dazu ist der Integrator angepasst, um die zusätzlichen Variablen zu berücksichtigen. Die zeitliche Temperaturänderung durch konduktive Wärmeleitung in Abhängigkeit der spezifischen Wärmekapazität c_{p} und der Wärmeleitfähigkeit κ ist

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{1}{\rho c_{\mathrm{p}}} \nabla(\kappa \nabla T) \tag{3.2}$$

und kann nach Cleary und Monaghan [99] unter Verwendung des Laplace-Operators von Brookshaw [70] (s. Gleichung 2.17) als

$$\frac{\mathrm{d}T_a}{\mathrm{d}t} = \sum_b \frac{m_b}{c_{\mathrm{p},a}\rho_a\rho_b} \frac{4\kappa_a\kappa_b}{(\kappa_a + \kappa_b)} (T_a - T_b) \frac{\boldsymbol{x}_{ab}}{|\boldsymbol{x}_{ab}|^2} \nabla W_{ab} \tag{3.3}$$

approximiert werden. Dazu kommen für freie Oberflächen die Wärmeströme in die Umgebung durch Strahlung $\dot{Q}_{\mathrm{S}}$ und Konvektion $\dot{Q}_{\mathrm{K}}$

$$\dot{Q}_{\mathrm{S}} = \varepsilon\sigma A \left(T_{\mathrm{U}}^4 - T^4\right) \tag{3.4}$$

$$\dot{Q}_{\mathrm{K}} = h_{\mathrm{c}} A \left(T_{\mathrm{U}} - T\right) \tag{3.5}$$

mit dem Emissionskoeffizient ε, der Stefan-Boltzmann-Konstante σ, dem konvektiven Wärmeübergangskoeffizient der Umgebungstemperatur T_U und der freien Oberfläche A. Die Berechnung der freien Oberfläche wird in Abschnitt 3.3.5 beschrieben. Die Temperaturänderung der Partikel an der freien Oberfläche ergeben sich zu

$$\frac{\mathrm{d}T_a}{\mathrm{d}t} = A_a \frac{\varepsilon\sigma}{m_a c_{\mathrm{p},a}} \left(T_\mathrm{U}^4 - T_a^4\right) \tag{3.6}$$

für die Strahlung und

$$\frac{\mathrm{d}T_a}{\mathrm{d}t} = A_a \frac{h_\mathrm{c}}{m_a c_{\mathrm{p},a}} \left(T_\mathrm{U} - T_a\right) \tag{3.7}$$

für die Konvektion. Die beschriebenen Gleichungen werden ausschließlich für die Fluidpartikel gelöst. Strahlung und Konvektion werden zudem nur für Partikel ausgewertet, die durch den in Abschnitt 3.3.5 beschriebenen Algorithmus der Oberfläche zugeordnet wurden. Für die Partikel der Bauplattform und im Partikeleinlass werden konstante Temperaturen angenommen. Sie werden aber in der Berechnung der konduktiven Wärmeleitung als Randbedingungen berücksichtigt. Aus diesem Grund ist für den konduktiven Wärmeübergang keine Identifizierung der Kontaktflächen notwendig. Die in das Modell eingesetzten Temperaturen können Tabelle 3.5 entnommen werden.

Tab. 3.5: Temperaturen für die Modellierung in PySPH

$T_\mathrm{Düse}$	$T_\mathrm{Bauplattform}$	T_U
220 °C	55 °C	20 °C

3.3.5 Oberflächenerkennung

Um den Wärmeübergang zur Umgebung durch Konvektion und Strahlung wie in 3.3.4 berechnen zu können, ist eine Methode notwendig, der jedem Partikel die freie Oberfläche, die er repräsentiert, zuweist. Da sich die Anordnung der SPH-Partikel beliebig ändern kann, ist zunächst nicht definiert, welche Partikel einen Teil der Oberfläche repräsentieren und welche vollständig innerhalb des Fluids liegen. Wie in Abschnitt 1.1 erwähnt, kann dies als Vorteil der Methode verstanden werden, da die Phasengrenzen ohne zusätzlichen Rechenaufwand abgebildet werden können. Das Aufbringen von Temperaturrandbedingungen an den Phasengrenzen macht einen solchen Algorithmus jedoch notwendig. Zur Unterscheidung wird ein von Doring [100] vorgeschlagener Algorithmus verwendet. Dieser wird, gefolgt von einem weiteren Schritt, von Marrone et al. [101] zur Identifikation von Oberflächen eingesetzt. Afrasiabi et al. [102] nutzen ihn ebenfalls zur Modellierung von Wärmeübergängen. Zunächst wird die von Randles

und Libersky [103] eingeführte Inverse der *Renormierungsmatrix*

$$\boldsymbol{A}_a^{-1} = \sum_b (\boldsymbol{r}_{ba} \otimes \nabla W_{ab}) V_b \tag{3.8}$$

berechnet und ihre Eigenwerte bestimmt. Der kleinste Eigenwert λ strebt für Partikel im Inneren des Fluids gegen eins und für Partikel ganz ohne Nachbarn gegen null. Somit ist er ein Indikator dafür, welche Partikel die freie Oberfläche bilden. Marrone et al. [101] sowie Afrasiabi et al. [102] gehen davon aus, dass alle Partikel, für die gilt $\lambda > 0,75$, im Inneren des Fluids liegen. In Abbildung 3.3 sind die Partikel entsprechend ihres λ-Werts für einen Balken mit rechteckigem Querschnitt eingefärbt. Zusätzlich sind beispielhaft die Werte für ausgewählte Partikel notiert. Wie erwartet gilt für die meisten Partikel im Inneren $\lambda \approx 1,0$. Die von Außen zweite Schicht

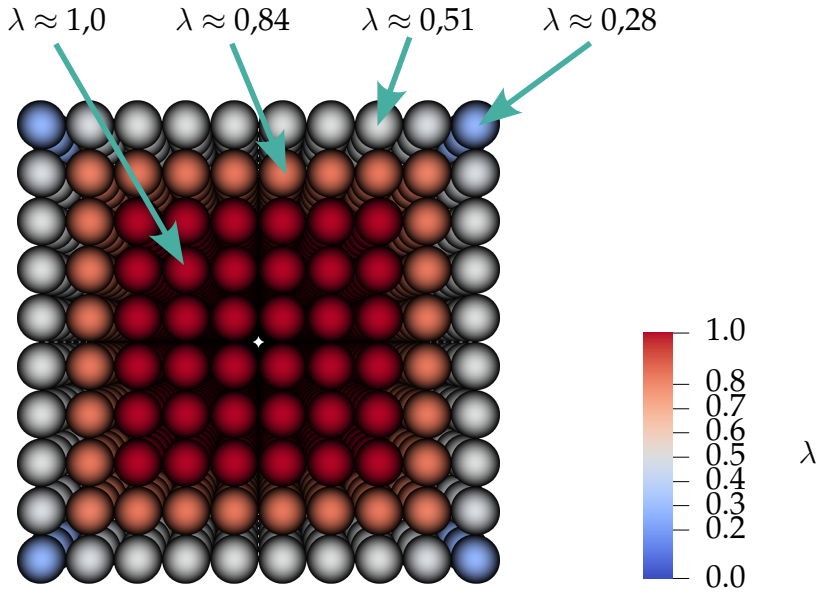


Abb. 3.3: Schematischer Aufbau der Simulation mit beweglicher Bauplattform (grau) und Partikelgenerator (rot).

nimmt Werte $0,75 < \lambda < 1,0$ an. Für die äußere Schicht, welche die freie Oberfläche bildet, gilt $\lambda < 0,75$, wobei λ entlang der Kanten des Balkens deutlich kleiner ist als entlang der Flächen. Dies ist zu erwarten, da die Partikel auf den Kanten doppelt so viel der freien Oberfläche repräsentieren. Obwohl SPH-Partikel häufig als Kugeln oder Würfel dargestellt werden, repräsentieren sie ein Kontinuum anstatt diskreter Teilchen. Somit haben sie im eigentlichen Sinne keine definierte Oberfläche. Da insbesondere die konvektive Kühlung in der MEX schwer abzuschätzen ist, wird zunächst ein einfaches Modell verwendet, um jedem Partikel einen Teil der freien Oberfläche zuzuordnen. Es wird angenommen, dass ein Partikel ohne Nachbarn die Oberfläche eines Würfels hat. Da das Volumen eines Partikels bekannt ist, berechnet sich die

maximale freie Oberfläche zu

$$A_{\max} = 6V_a^{\frac{2}{3}} \tag{3.9}$$

bzw.

$$A_{\max} = 4\sqrt{V_a}. \tag{3.10}$$

für zweidimensionale Rechnungen. Partikel mit $\lambda > 0,75$ repräsentieren keine freie Oberfläche. Dazwischen wird der lineare Zusammenhang

$$A_a = \left(1 - \frac{\lambda_a}{0,75}\right) * A_{\max} \tag{3.11}$$

angenommen. Dieser ist lediglich eine erste Annäherung und wird v. a. aufgrund seiner Einfachheit gewählt. In Abbildung 3.4 sind für ausgewählte Partikel aus dem in Abbildung 3.3 gezeigten Stab jeweils die korrekte freie Oberfläche über den zugehörigen Werten für λ aufgetragen. Dazu ist der angenommene lineare Zusammenhang als Kurve dargestellt. Die freie Oberfläche ist mit der für einen Partikel maximal möglichen Oberfläche normiert. Dieses einfache Beispiel

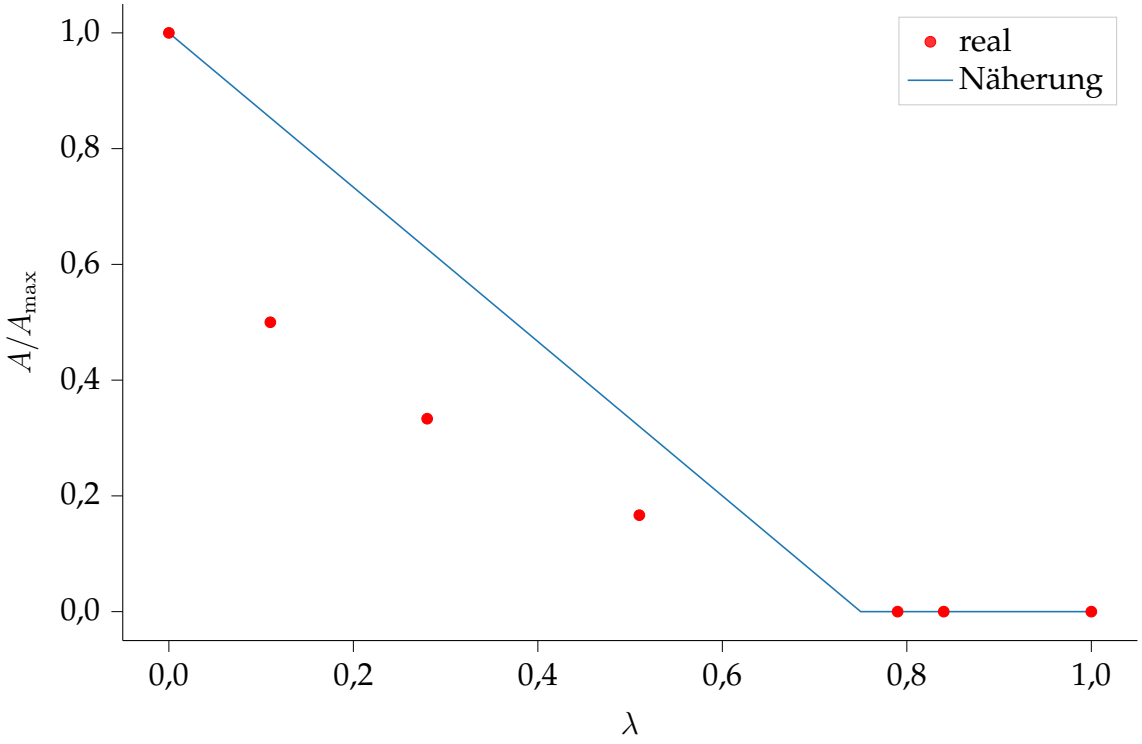


Abb. 3.4: Reale freie Oberfläche der Partikel in einem quadratischen Stab über λ und angenommener Zusammenhang zwischen freier Oberfläche und λ.

zeigt bereits, dass die freie Oberfläche der Partikel durch den angenommenen Zusammenhang überschätzt wird. Die Näherung erfüllt die Bedingungen $A_{\text{rel}} = 0$ für $\lambda \geq 0,75$ und $A_{\text{rel}} = 1$

für $\lambda = 0$. Allerdings weichen die dazwischenliegenden Werte deutlich ab: Bei $\lambda \approx 0,49$ ist die berechnete Oberfläche mehr als doppelt so hoch wie die reale.

3.3.6 Viskosität

Durch die Einführung der Temperatur ist die scherraten- und temperaturabhängige Berechnung der Viskosität nach dem Cross-WLF-Ansatz [90] möglich. Das Modell wird mehrfach für die Prozesssimulation der MEX eingesetzt (s. Abschnitt 2.2.3). Die Scherrate wird in PySPH standardmäßig nicht berechnet. Zunächst muss der Geschwindigkeitsgradiententensor

$$\boldsymbol{L} = \nabla \boldsymbol{v}_a = \sum_b \frac{m_b}{\rho_b} \boldsymbol{v}_{ba} \otimes \nabla W_{ab} \tag{3.12}$$

bestimmt werden. Dessen symmetrischer Anteil

$$\boldsymbol{D} = \frac{1}{2} \left(\boldsymbol{L} + \boldsymbol{L}^T \right) \tag{3.13}$$

wird Deformationsgeschwindigkeitstensor genannt [104]. Die Scherrate berechnet sich zu [104]

$$\dot{\gamma} = \sqrt{\frac{1}{2} \boldsymbol{D} \cdot \boldsymbol{D}^T}. \tag{3.14}$$

Die Viskosität berechnet sich nach dem Cross-WLF-Ansatz zu

$$\eta = \frac{\eta_0}{1 + \left(\frac{\eta_0 \dot{\gamma}}{\tau^*} \right)^{1-n}}. \tag{3.15}$$

Die kritische Spannungsebene beim Übergang zur Strukturviskosität τ^* und das Potenzprofil bei hoher Schergeschwindigkeit n müssen experimentell bestimmt werden. Die Viskosität für geringe Scherraten η_0 berechnet sich zu

$$\eta_0 = D_1 \exp \left[-\frac{A_1 \left(T - T^* \right)}{A_2 + \left(T - T^* \right)} \right]. \tag{3.16}$$

Der Koeffizient A_2 und die durch Kurvenanpassung bestimmte Glasübergangstemperatur T^* können druckabhängig durch die Zusammenhänge

$$A_2 = A_3 + D_3 p \tag{3.17}$$

bzw.

$$T^* = D_2 + D_3 p \tag{3.18}$$

berechnet werden. Laut der Materialdatenbank von Autodesk Moldflow 2021 [90] ist allerdings $D_3 = 0$ für PLA, weshalb der Druck nicht berücksichtigt wird. Die übrigen Koeffizienten A_1, A_3, D_1 und D_2 werden experimentell bestimmt. Die Werte aller Koeffizienten sind in Tabelle 3.1 in Abschnitt 3.1 aufgeführt. In Abbildung 3.5 ist die mit dem Cross-WLF-Ansatz berechnete Viskosität für ausgewählte Temperaturen über der Scherrate aufgetragen. Der Ansatz kann

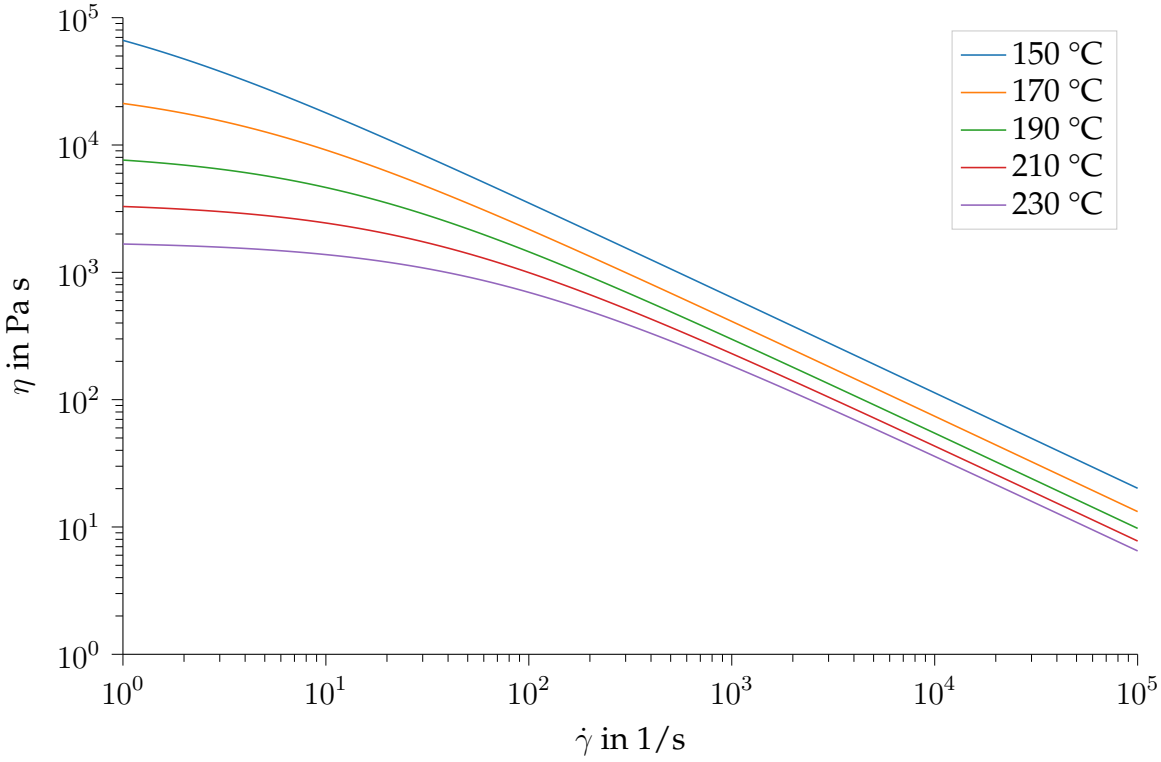


Abb. 3.5: Mit dem Cross-WLF-Ansatz berechnete Viskositäten für verschiedene Temperaturen über die Scherrate

die Viskosität nur oberhalb der durch Kurvenanpassung bestimmten Glasübergangstemperatur abbilden. Für PLA liegt diese bei $T^* = 100\,°C$. Für eine Schmelze mit einer relativ hohen Temperatur von 230 °C liegt die Viskosität bei bis zu $1{,}7{\times}10^{-3}$ MPa/s. Durch Modifikationen kann in PySPH statt mit einer konstanten Viskosität auch mit der durch den Cross-WLF-Ansatz erhaltenen gerechnet werden.

4 Ergebnisse

4.1 Modellierung in ABAQUS

In Abschnitt 3.2 ist der Aufbau des MEX-Modells in ABAQUS beschrieben. Im Folgenden werden die Ergebnisse der Modellierung vorgestellt.

4.1.1 Ablegen eines einzelnen Strangs

Zunächst wird ein einzelner PLA-Strang abgelegt, um geeignete Simulationsparameter zu finden. Insbesondere die künstliche Schallgeschwindigkeit c_s und Viskosität η sind entscheidend. Da der maximal mögliche Zeitschritt maßgeblich durch die künstliche Schallgeschwindigkeit begrenzt [38] wird, ist diese so niedrig wie möglich zu wählen. Eine zu geringe Schallgeschwindigkeit führt allerdings zu unphysikalischem Verhalten, insbesondere einer zu großen Kompressibilität. In Abbildung 4.1 ist das Ergebnis eines Durchlaufs mit einer Viskosität von 0,01 Pa s und einer künstlichen Schallgeschwindigkeit von $c_\mathrm{s} = 0{,}01\,\mathrm{m/s}$ abgebildet. Die Partikel sind entsprechend

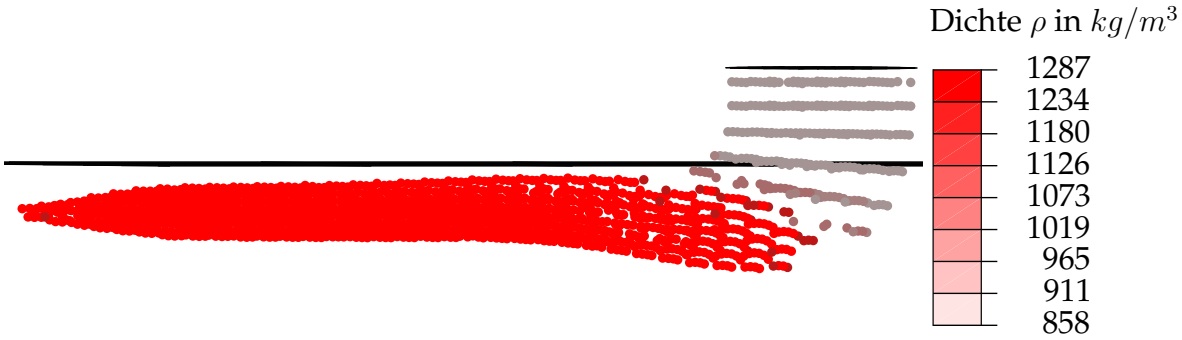
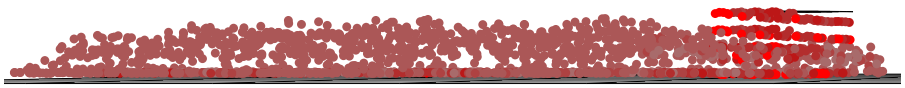


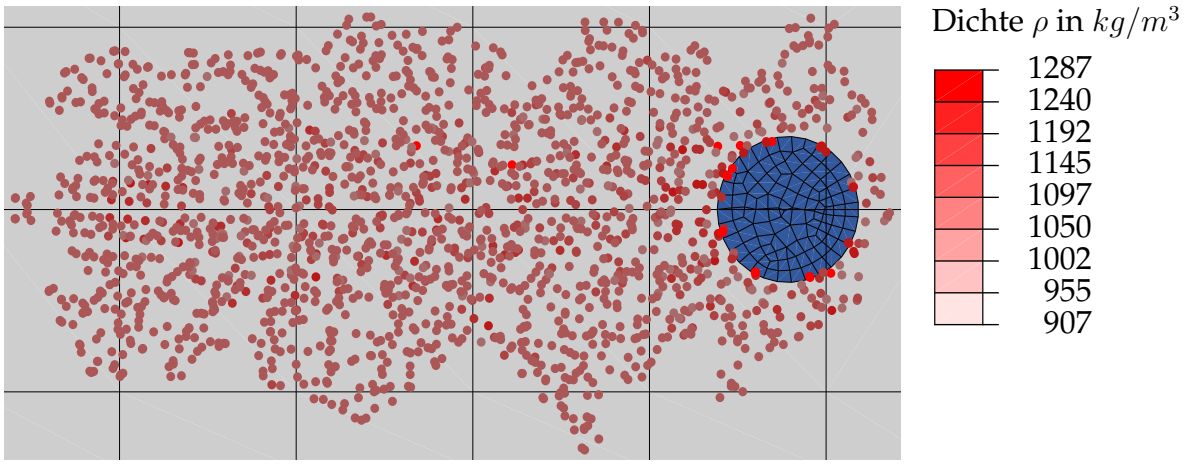
Abb. 4.1: Durchdringung der Bauplattform aufgrund zu geringer Schallgeschwindigkeit von $c_\mathrm{s} = 0{,}01\,\mathrm{m/s}$ bei der Ablage eines Strangs mit einer Viskosität von $\eta = 0{,}01\,\mathrm{Pa\,s}$ in der Seitenansicht.

der Dichte eingefärbt. Der Abstand zwischen Düse und Plattform beträgt 0,2 mm, was dem Düsenradius entspricht. Die Dichte ist unmittelbar unter dem Düsenaustritt gegenüber der

Referenzdichte von $\rho_0 = 1072{,}7\,\mathrm{kg/m^3}$ deutlich reduziert. Die Partikel durchdringen die Bauplattform, bewegen sich allerdings nicht weiter durch den Raum. Den tiefsten Punkt erreichen Sie kurz hinter der Düse, an dem die Dichte deutlich erhöht ist. Anschließend bewegen sie sich wieder etwas nach oben, steigen aber nicht mehr über die Plattform. Dies ist vermutlich darauf zurückzuführen, dass die Kontaktbedingung festschreibt, dass sich die Partikel nach dem erstmaligen Kontakt nicht mehr von der Plattform trennen. Die Dichte ist über den gesamten Strang unter der Bauplatte um etwa 20 % erhöht und der Strang zerfließt nicht wie erwartet. Sowohl das Durchdringen der Platte, als auch die Fixierung der Partikel zeigen, dass die Schallgeschwindigkeit zu niedrig gewählt ist. In Abbildung 4.2 (a) ist das Ergebnis einer Simulation mit einer Schallgeschwindigkeit von $c_\mathrm{s} = 1\,\mathrm{m/s}$ dargestellt. Das unphysikalische Kontaktver-



(a)



(b)

Abb. 4.2: Ablage eines Strangs mit einer Viskosität von $\eta = 0{,}01\,\mathrm{Pa\,s}$ in (a) Seitenansicht und (b) Draufsicht.

halten tritt durch die erhöhte Schallgeschwindigkeit nicht mehr auf. Die Dichte weicht nach wie vor um bis zu 20 % vom vorgegebenen Wert ab. Allerdings ist sie v. a. direkt unter dem Düsenaustritt stark erhöht. Dahinter zerfließt das Fluid und die Dichte sinkt auf Werte nahe der Referenzdichte. Vereinzelt finden sich im Strang jedoch Partikel mit deutlich erhöhter oder reduzierter Dichte. Eine weitere Erhöhung der Schallgeschwindigkeit, die der Kompressibilität entgegenwirkt, ist mit erhöhtem Rechenaufwand verbunden, da die Zeitschritte immer kleiner gewählt werden müssen. Aus diesem Grund werden diese Abweichungen zunächst akzeptiert. In der in Abbildung 4.2 gezeigten Simulation liegt der Zeitschritt bei $\delta t \approx 4{,}5 \times 10^{-5}\,\mathrm{s}$. Die erhöhte Schallgeschwindigkeit führt außerdem dazu, dass ein kleiner Teil des Fluids vor der

Düse hergeschoben wird. Dass sich eine solche der Düse vorgelagerte Fließfront bildet, ist bei geringem Abstand zwischen Düse und Bauplattform zu erwarten und wird in Experimenten beobachtet bzw. numerischen Modellen abgebildet [26, 32, 35, 36]. Abbildung 4.2 (b) zeigt eine Draufsicht auf die Bauplattform aus derselben Simulation. Die Abbildung zeigt, dass das Fluid hinter dem Düsenaustritt stark zerfließt und kein klar erkennbarer Strang gebildet wird. Die Viskosität ist also zu gering, um den Prozess sinnvoll abzubilden. Im realen Prozess ist die Viskosität von der Scherrate und Temperatur abhängig (siehe Abschnitt 3.3.6), liegt aber in jedem Fall mehrere Größenordnungen höher. So müsste für die Viskosität ein Wert zwischen $10\,\mathrm{Pa\,s}$ und $10\,000\,\mathrm{Pa\,s}$ eingesetzt werden. Um die Simulation dennoch stabil zu halten, müssen für so hohe Viskositäten aber sehr kleine Zeitschritte gewählt werden [38], wodurch der Rechenaufwand stark zunimmt. In Abbildung 4.3 (a) ist die Seitenansicht und in 4.3 (b) die Draufsicht auf die Ablage eines Strangs aus einer Simulation mit $\eta = 1\,\mathrm{Pa\,s}$ gezeigt. Durch die erhöhte

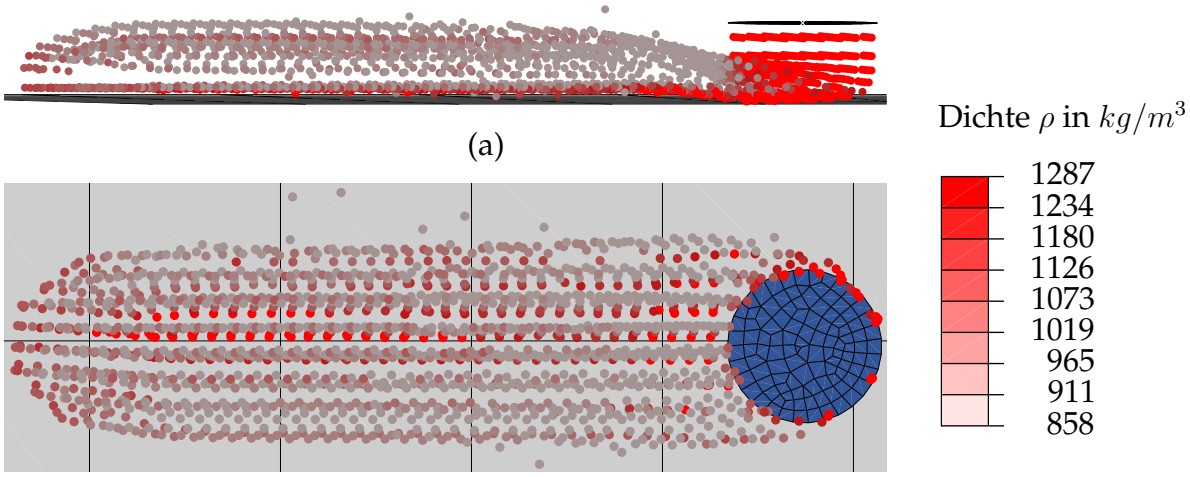


Abb. 4.3: Ablage eines Strangs mit einer Viskosität von $\eta = 1\,\mathrm{Pa\,s}$ in (a) Seitenansicht und (b) Draufsicht.

Viskosität bildet sich ein definierter Polymerstrang. Unterhalb der Düse ist die Dichte nach wie vor deutlich erhöht und der Querschnitt des Strangs sehr gering. Die Partikel sind in diesem Bereich eng gepackt. Der vorgegebene Partikelabstand Δx wird so deutlich unterschritten, dass eine noch höhere Dichte zu erwarten wäre. Unmittelbar stromab des Düsenaustritts bewegen sich die Partikel wieder auseinander. Der Querschnitt des Strangs nimmt zu. Die Dichte sinkt in diesem Bereich teils deutlich unter den Referenzwert ρ_0. Stromab der Düse ist zu erkennen, dass der Strang parallel zur Bauplattform geteilt wird. Ein Teil der Partikel verbleibt auf der Plattform, ein anderer trennt sich davon und scheint darüber zu „schweben“. Vermutlich ist auch dies ein Effekt der hohen Kompressibilität in Verbindung mit der gewählten Kontaktbedin-

gung. Zudem lösen sich vereinzelt Partikel vom Strang und entfernen sich davon. Auch diese werden vermutlich durch Kompressibilitätseffekte beschleunigt. Die in der Simulation mit der Viskosität $\eta = 0{,}01\,\mathrm{Pa\,s}$ ausgebildete Fließfront vor der Düse tritt nun nicht mehr auf. Der stabile Zeitschritt ist deutlich reduziert und sinkt zum Ende der Simulation auf $\delta t \approx 2{,}4 \times 10^{-8}\,\mathrm{s}$, was eine erheblich längere Rechenzeit bedeutet. Um die starke Kompression des Fluids unmittelbar nach dem Düsenaustritt zu reduzieren, wird der Abstand zwischen Düse und Bauplattform auf $h_{\mathrm{in}} = 0{,}4\,\mathrm{mm}$ erhöht. Dies entspricht dem Düsendurchmesser d_{D}. Das Ergebnis dieser Simulation ist in Abbildung 4.4 in (a) als Seitenansicht, (b) als Draufsicht und (c) in einer Rückansicht auf den Strang dargestellt. In der Seitenansicht (a) ist zu erkennen, dass die Partikelabstände

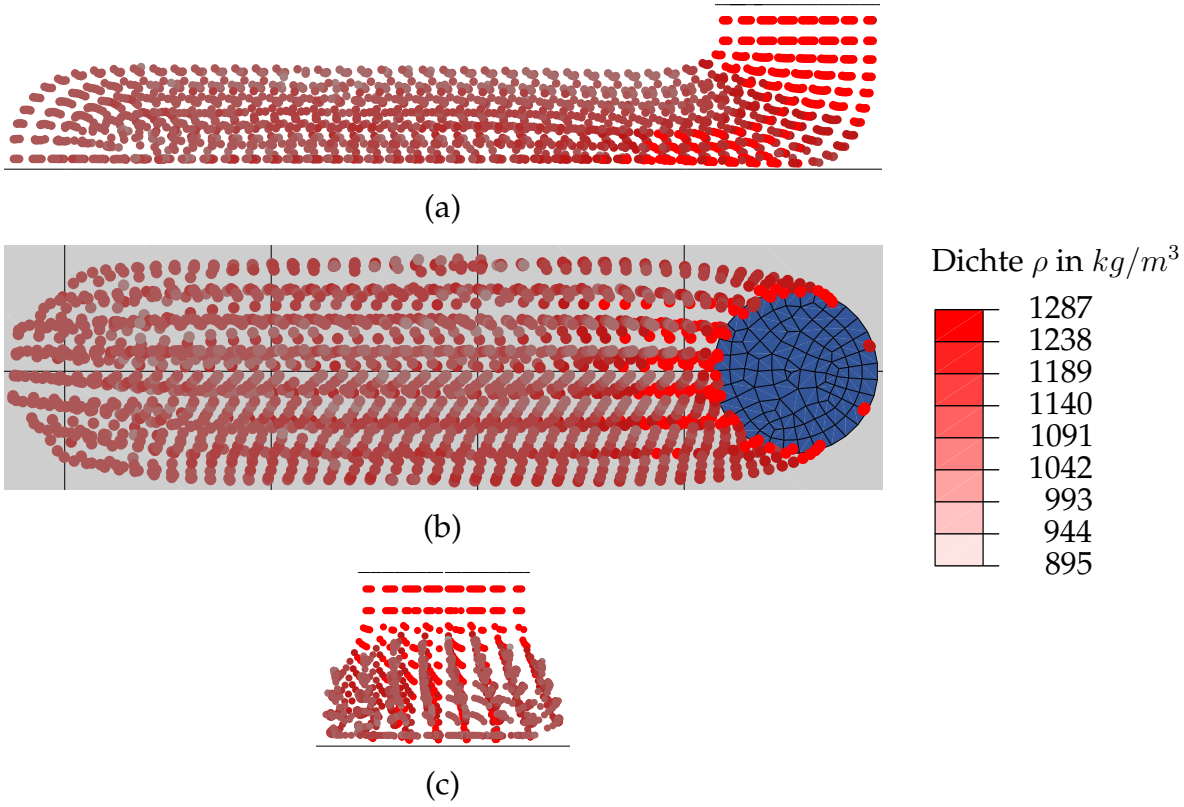


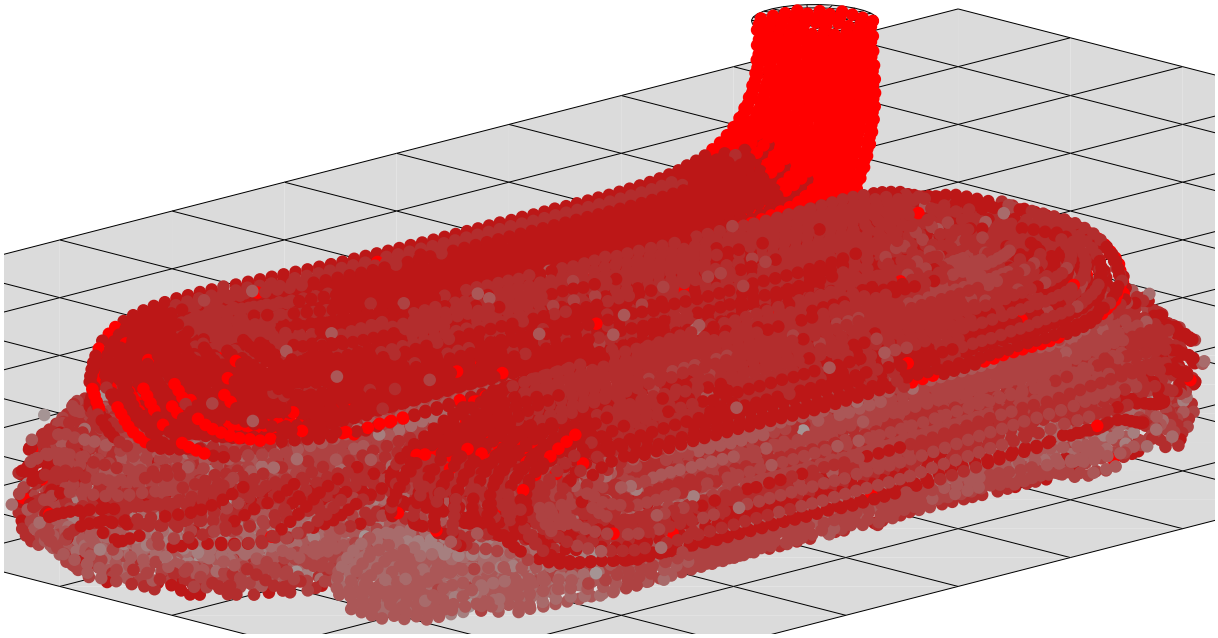
Abb. 4.4: Ablage eines Strangs mit einer Viskosität von $\eta = 1\,\mathrm{Pa\,s}$ und erhöhter Düsenposition $h_{\mathrm{in}} = 0{,}4\,\mathrm{mm}$ in (a) Seitenansicht, (b) Draufsicht und (c) Rückansicht.

unmittelbar stromab des Düsenaustritts deutlich näher am vorgegebenen Wert liegen, als in den Simulationen zuvor. Das Fluid wird weniger komprimiert. Die berechnete Dichte in diesem Bereich ist allerdings gegenüber dem Referenzwert ρ_0 ebenfalls um ca. 20 % erhöht. Zwischen dem Partikelabstand und der Dichte scheint nicht, wie erwartet, ein direkter Zusammenhang zu bestehen. Weiter stromab trennt sich der Strang nicht mehr auf. Zudem lösen sich keine einzelnen Partikel aus dem Fluid heraus. Die Draufsicht (b) und Rückansicht (c) zeigen, dass der Strang nicht zerfließt, sondern seine Form beibehält. Obwohl der Abstand zwischen Düse

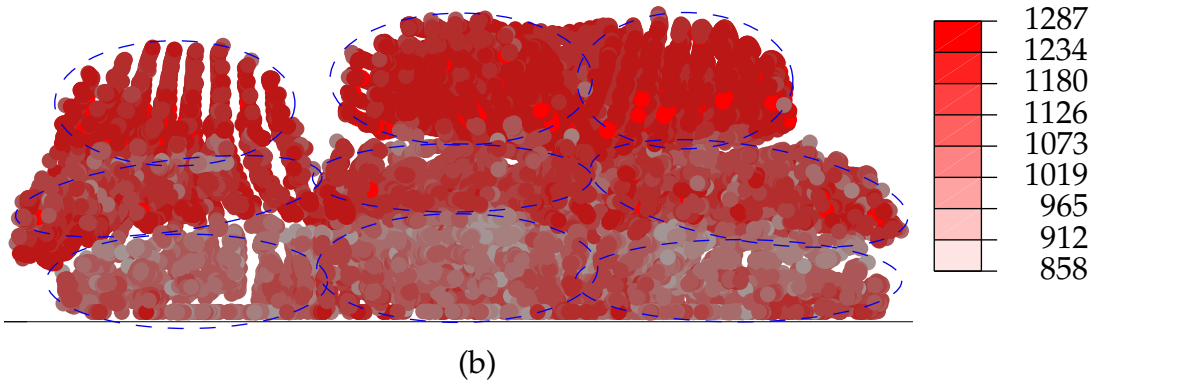
und Plattform größer gewählt ist, ist die Breite des Strangs nicht signifikant geringer. Auch der Zeitschritt ist mit $\delta t \approx 1{,}5 \times 10^{-7}\,\mathrm{s}$ etwas größer.

4.1.2 Ablegen mehrerer Schichten

Durch das Ablegen eines einzelnen Polymerstrangs ist eine Parameterkombination bekannt, für die die Simulation stabil ist und ein Strang ausgebildet wird, der die erwartete Gestalt aufweist. Dieselbe Parameterkombination wird genutzt, um das Ablegen mehrerer Schichten zu modellieren. Es werden drei Schichten übereinander abgelegt. Die Schichtdicke ist $d_\mathrm{S} = 0{,}4\,\mathrm{mm}$, was dem Düsendurchmesser d_D entspricht. Nachdem eine Schicht abgelegt ist, verfährt die Bauplatte um die Schichtdicke nach unten. Jede Schicht besteht jeweils aus drei parallelen Polymersträngen, die mit einem Versatz von $b_\mathrm{S} = 0{,}6\,\mathrm{mm}$ zueinander extrudiert werden. Insgesamt werden neun Stränge abgelegt. Jeder Strang ist $l_\mathrm{S} = 3\,\mathrm{mm}$ lang. Das Ergebnis dieser Simulation ist in Abbildung 4.5 dargestellt. Auch hier sind die Partikel entsprechend ihrer Dichte eingefärbt. Zusätzlich ist die ungefähre Lage der einzelnen Stränge mit blauen Ellipsen markiert. In der Übersicht (a) ist der gesamte extrudierte Körper gezeigt. Die unteren Schichten erhalten ihre Form unter der Last der darüberliegenden ausreichend, sodass ein annähernd quaderförmiger Körper extrudiert wird. An den Seiten des Körpers sind die abgelegten Stränge nach wie vor als solche zu erkennen. In Ansicht (b) ist ein Querschnitt durch den Körper abgebildet. Im Inneren des Körpers sind die einzelnen Stränge optisch teils nicht sicher voneinander zu unterscheiden. Das Fluid ist durch die Gravitation und Bewegung der Bauplattform teilweise in die Lücken zwischen den einzelnen Strängen geflossen und hat diese gefüllt. Die Dichte ist in jeder Schicht höher als in der darunterliegenden. Auch innerhalb einer Schicht weisen früher extrudierte Stränge geringere Dichten auf. In der unteren Schicht liegt die Dichte eines Großteils der Partikel unter dem Referenzwert $\rho_0 = 1072{,}7\,\mathrm{kg/m^3}$. In der mittleren Schicht weißt die Mehrheit der Partikel eine Dichte über dem Referenzwert auf. In der oberen Schicht gibt es nur vereinzelt Partikel, deren Dichte den Referenzwert unterschreitet. Auch innerhalb einer Schicht ist die Dichte der zuerst extrudierten Stränge etwas geringer. Es besteht zudem eine große Lücke zwischen dem ersten extrudierten Strang der obersten Ebene und den beiden weiteren. Eine mögliche Ursache hierfür ist die Bewegung der Plattform und Trägheit des Fluids. Bei Betrachtung der Verschiebungen der Bauplattform fällt jedoch auf, dass diese nicht exakt dem vorgegebenen Pfad folgt. In Abbildung 4.6 sind die Verschiebungen in die drei Raumrichtungen abgebildet. Die Werte wurden auf die vorgegebene maximale Verschiebung in die jeweilige Richtung normiert, sodass nur Werte zwischen 0 und -1 erwartet werden. Die Verschiebung weicht teils deutlich vom vorgegebenen Wert ab. Zum Zeitpunkt $t \approx 0{,}36\,\mathrm{s}$ weicht die Verschiebung in Längsrichtung der Stränge u um ca. 0,22 mm vom vorgegebenen Wert ab, was etwa 7,2 % der Gesamtlänge des Strangs entspricht. Die größte Abweichung vom Sollwert in Querrichtung wird während



(a)



(b)

Abb. 4.5: Extrusion dreier Schichten bestehend aus jeweils drei Strängen mit einer Viskosität von $\eta = 1\,\mathrm{Pa\,s}$ und erhöhter Düsenposition $h_{\mathrm{in}} = 0{,}4\,\mathrm{mm}$. (a) zeigt den gesamten Körper und (b) einen Querschnitt.

der Extrusion des letzten Strangs ab $t \approx 0{,}48\,\mathrm{s}$ erreicht. Die Sollposition wird hier nur um ca. 0,05 mm verfehlt. Da die Bauplattform entlang dieser Achse um 1,2 mm verschoben werden soll, beträgt der Fehler relativ zur maximalen Verschiebung in y-Richtung ca. 3,8 %. Entlang der Hochachse z soll die Plattform um $w = -0{,}8\,\mathrm{mm}$ verschoben werden. Zum Ende der Simulation befindet sie sich aber nur bei $z \approx -0{,}76\,\mathrm{mm}$. Die Abweichung relativ zur Gesamtstrecke in z-Richtung beträgt damit ca. 5 %. Die Abweichungen von den Sollwerten sind höher als die zu erwartenden numerischen Fehler und spiegeln sich teilweise deutlich erkennbar im Ergebnis wider. In Abbildung 4.5 (a) ist deutlich zu erkennen, dass der letzte Strang der mittleren Schicht und der erste Strang der obersten etwas kürzer sind als die daneben liegenden. Die

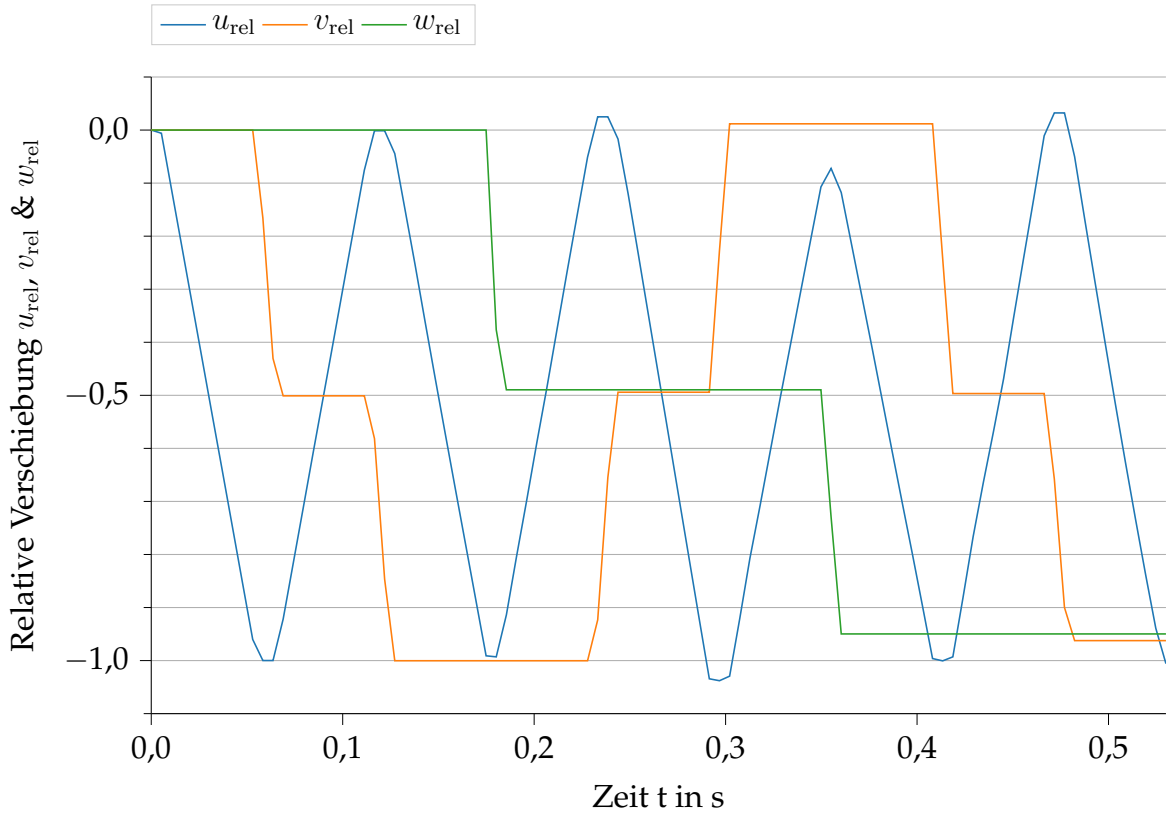


Abb. 4.6: Verschiebung der Bauplattform bei der Extrusion dreier Schichten aus je drei Strängen relativ zur maximalen vorgegebenen Verschiebung.

Verschiebung in x-Richtung u in Abbildung 4.6 ist zum Zeitpunkt $t \approx 0{,}36\,\mathrm{s}$ zeigt ebenfalls, dass die Plattform die Position $x = 0\,\mathrm{mm}$ nicht erreicht. Die Abweichungen sind gegen Ende der Simulation größer als in der Anfangsphase. Da die Position für jeden Schritt als Absolutwert hinterlegt ist, sollte allerdings ausgeschlossen sein, dass der Positionsfehler über den Verlauf der Simulation kumuliert wird. Die Ursache für die teils deutlichen Abweichungen ist nicht bekannt.

Die Rechenzeit dieser Simulation unter Verwendung aller sechs Kerne eines INTEL *i5-9500* beträgt etwa 33 h. Die Simulationszeit beträgt $\Delta t \approx 0{,}53\,\mathrm{s}$. Die lange Rechendauer ist insbesondere durch die geringen Zeitschritte begründet. Legt man die Extrusionsgeschwindigkeit v_{ex}, den Düsendurchmesser d_{D} und die maximal eingesetzte Viskosität von 1 Pa s zugrunde, berechnet sich die Reynoldszahl des Problems zu

$$Re = \frac{\rho v_{\mathrm{ex}} d_{\mathrm{D}}}{\eta} \approx 0,026. \tag{4.1}$$

Die Reynoldszahl kann als Verhältnis der Trägheitskräfte zu den viskosen Kräften im Fluid interpretiert werden. Ein Wert $Re \ll 1$ gibt an, dass der Vorgang durch die Viskosität dominiert wird.

4.2 Validierung des thermischen Modells in PySPH

Wie in Abschnitt 3.3.4 beschrieben, wird zur Berechnung der Temperaturverteilung eine zusätzliche Gleichung in den PySPH-Code implementiert, durch die die konduktive Wärmeleitung in der Schmelze sowie der Wärmeaustausch mit der Umgebung durch Strahlung und Konvektion berücksichtigt werden kann. Diese neu implementierte Wärmeleitungsgleichung wird mit der für die FEM bereits vorhandenen in Abaqus für einen einfachen Testfall verglichen, um zu überprüfen, ob das thermische Modell vergleichbare Ergebnisse wie ein etabliertes produziert. Dazu wird das Temperaturprofil

$$T(z) = 2500\,°\mathrm{C/m^2}(z - 0{,}05\,\mathrm{m})^2 + 150\,°\mathrm{C} \tag{4.2}$$

auf einen Stab mit quadratischem Querschnitt mit den Seitenlängen 0,02 m und Länge $z_{\max} = 0{,}2\,\mathrm{m}$ aufgebracht. Der Stab wird aus SPH-Partikeln in PySPH und aus *DC3D8*-Elementen [105] in Abaqus aufgebaut. In Abbildung 4.7 sind beide Modelle zum Zeitpunkt $t = 0\,\mathrm{s}$ mit dem beschriebenen Temperaturprofil dargestellt. Die Temperaturverteilung zum Startzeitpunkt ist

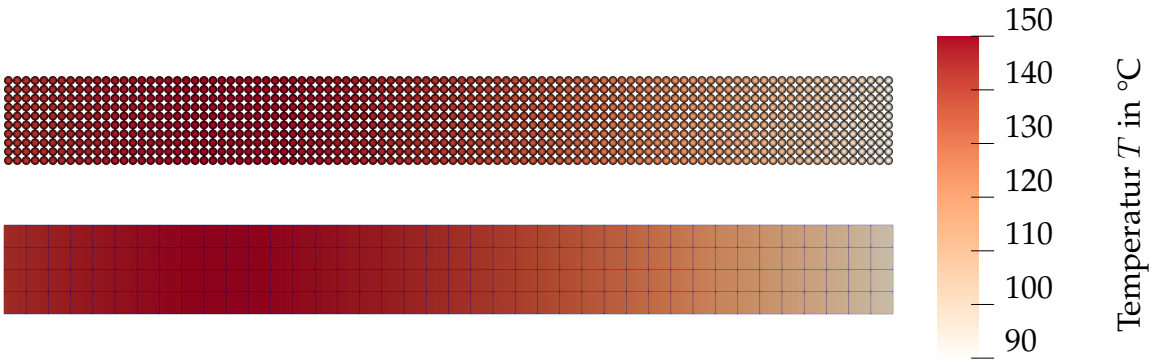
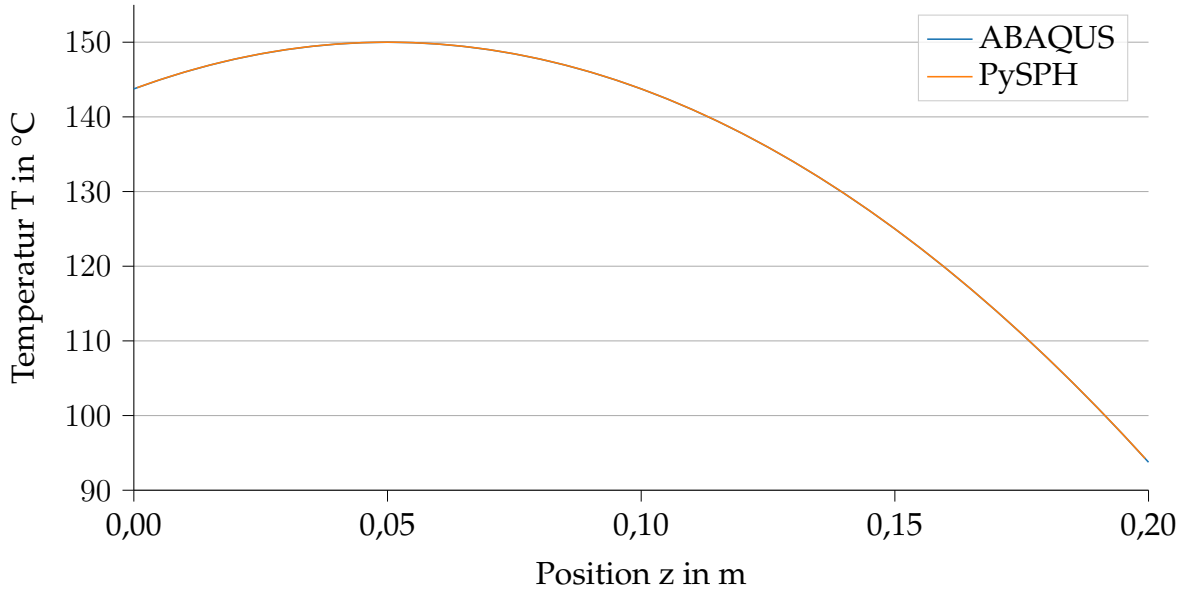
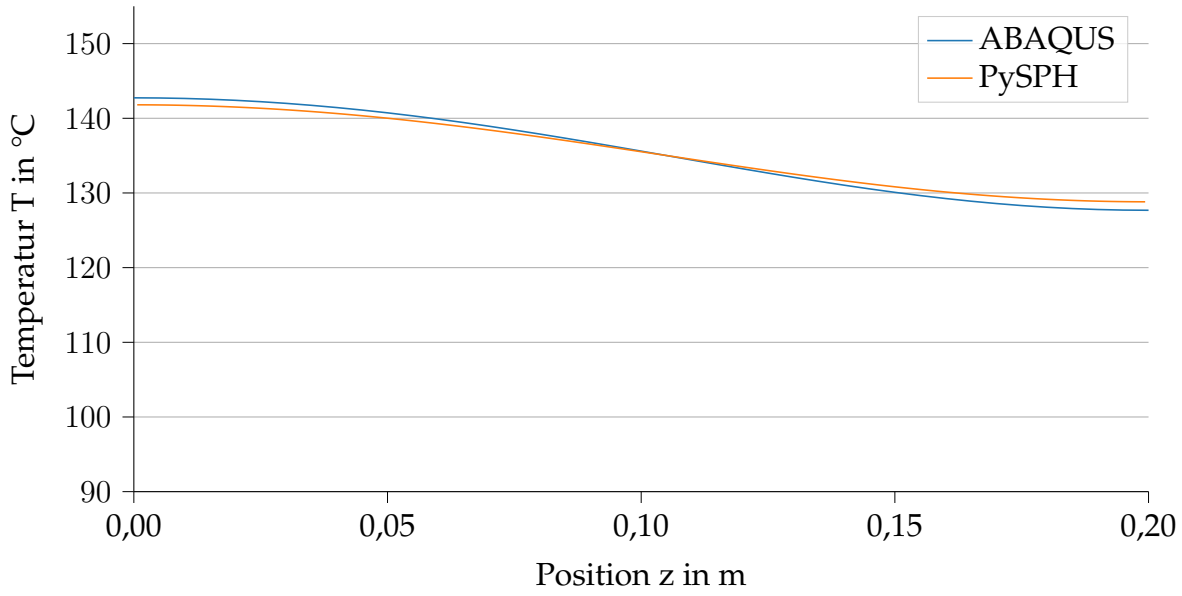


Abb. 4.7: Stab zur Validierung der konduktiven Wärmeleitung mit quadratischer Temperaturverteilung zum Zeitpunkt $t = 0\,\mathrm{s}$ aus SPH-Partikeln (oben) und Finiten Elementen (unten).

in Abbildung 4.8 (a) über die Position entlang des Stabes z aufgetragen. Das Maximum der Temperatur liegt bei $z = 0{,}05\,\mathrm{m}$ und beträgt 150 °C. Zu den Stabenden hin nimmt die Temperatur ab. Als Material wird Aluminium angenommen. Die Materialparameter sind in 3.3 in Abschnitt 3.1 aufgeführt.



(a)



(b)

Abb. 4.8: Verlauf der in ABAQUS und PySPH unter Berücksichtigung der Konduktion berechneten Temperaturverteilung über den Stab zu den Zeitpunkten (a) $t = 0\,\mathrm{s}$ und (b) $t = 50\,\mathrm{s}$.

4.2.1 Konduktive Wärmeleitung

Um die konduktive Wärmeleitung zu validieren, werden die Seiten des Stabes als adiabat angenommen. Die simulierte Zeit beträgt 50 s. Zum Ende der Simulation $t = 50\,\mathrm{s}$ stellt sich die

Temperaturverteilung, wie sie in Abbildung 4.8 (b) abgebildet ist, ein. Wie erwartet, gleichen sich die Temperaturen über die Länge des Stabes aneinander an. Der Unterschied zwischen dem Maximal- und Minimalwert ist deutlich geringer. Die Verläufe beider Modelle liegen nah beieinander, die größte Abweichung ist an den Stabenden zu erkennen und liegt bei −1,1 °C bzw. 0,9 °C. In Abbildung 4.9 ist die Differenz der Temperaturverläufe relativ zu den Ergebnissen aus ABAQUS über z zum Ende der Simulation $t = 50\,\mathrm{s}$ abgebildet. Auch hier ist zu erkennen, dass die

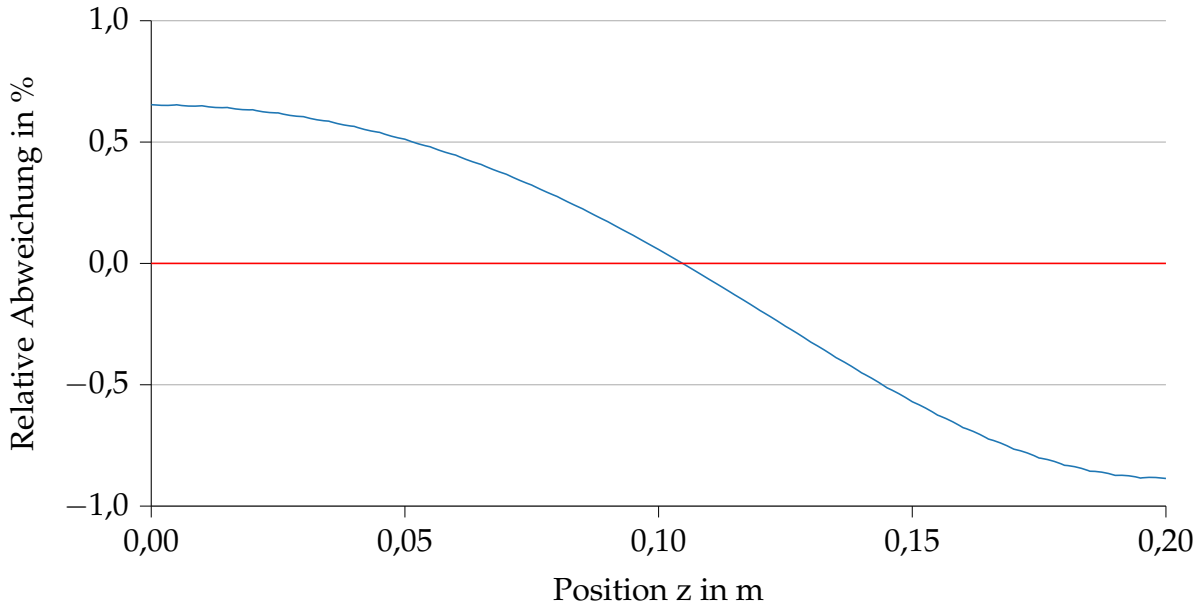


Abb. 4.9: Relative Abweichung der in ABAQUS und PYSPH unter Berücksichtigung der Konduktion berechneten Temperaturverteilung über den Stab zum Zeitpunkt $t = 50\,\mathrm{s}$

Abweichung an den Stabenden am größten ist. Die betragsmäßig größte relative Abweichung liegt bei ca. 0,89 %. Es wird angenommen, dass diese Abweichung ausreichend gering ist.

4.2.2 Konduktive Wärmeleitung, Strahlung und Konvektion

Zum Vergleich der beiden weiteren thermischen Effekte, dem Austausch von Wärme mit der Umgebung durch Wärmestrahlung und Konvektion, wird derselbe Testfall wie für die Konduktion betrachtet. Allerdings werden die Seiten des Aluminiumstabes nun nicht mehr als adiabat modelliert. Für die Wärmestrahlung werden ein Emissionskoeffizient von $\varepsilon = 1,0$ und ein konvektiver Wärmetransportkoeffizient von $h_\mathrm{c} = 100\,\mathrm{W/m^2K}$ angenommen. Die Umgebungstemperatur, die in beide Effekte mit einfließt, beträgt $T_\mathrm{U} = 20\,°\mathrm{C}$. Nach $\Delta t = 50\,\mathrm{s}$ Simulationszeit stellen sich entlang der Mitte des Stabes $x = y = 0{,}01\,\mathrm{m}$ die in Abbildung 4.10 dargestellten Temperaturverläufe ein. Gegenüber dem Fall ohne Strahlung und Konvektion sind die Temperaturen zum Ende der Simulationszeit deutlich geringer, da dem System nun

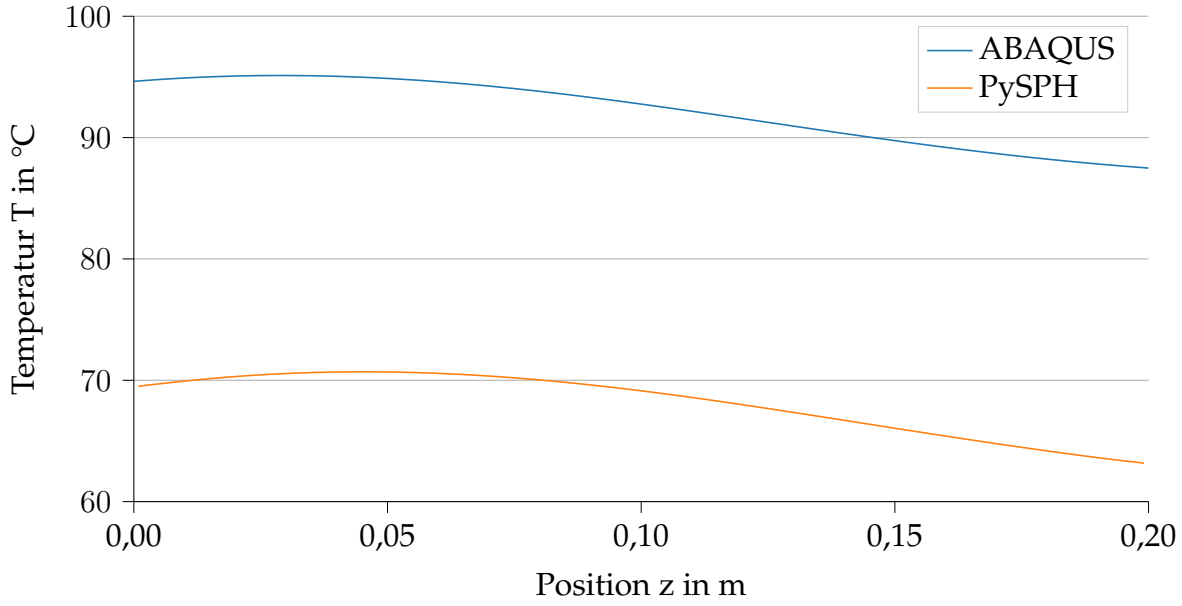


Abb. 4.10: Verlauf der in ABAQUS und PySPH unter Berücksichtigung aller Effekte berechneten Temperaturverteilung in der Stabmitte zum Zeitpunkt $t = 50\,\mathrm{s}$

über die Ränder Energie entzogen wird. Auch qualitativ ähneln sich die Verläufe stark: Das Temperaturmaximum liegt jeweils im ersten Viertel des Stabes. Zu den Stabenden nimmt die Temperatur ab. Am geringsten ist die Temperatur am Stabende bei $z = 0{,}2\,\mathrm{m}$. Allerdings liegen die Temperaturen in PySPH über die gesamte Länge des Stabes deutlich unter aus ABAQUS. Am Stabende beträgt die Differenz ca. 25 °C. Die Signifikanz der Abweichung zwischen den beiden Modellen wird auch durch die relative Abweichung in Abbildung 4.11 deutlich. Die in ABAQUS berechneten Werte sind zwischen 25,3 % und 27,9 % höher. Der Einfluss der Wärmestrahlung und Konvektion wird durch das thermische Modell in PySPH also signifikant überschätzt. Qualitativ ähneln sich die beiden Verläufe allerdings. Konvektion und Strahlung haben also einen ähnlichen Einfluss auf den Temperaturverlauf.

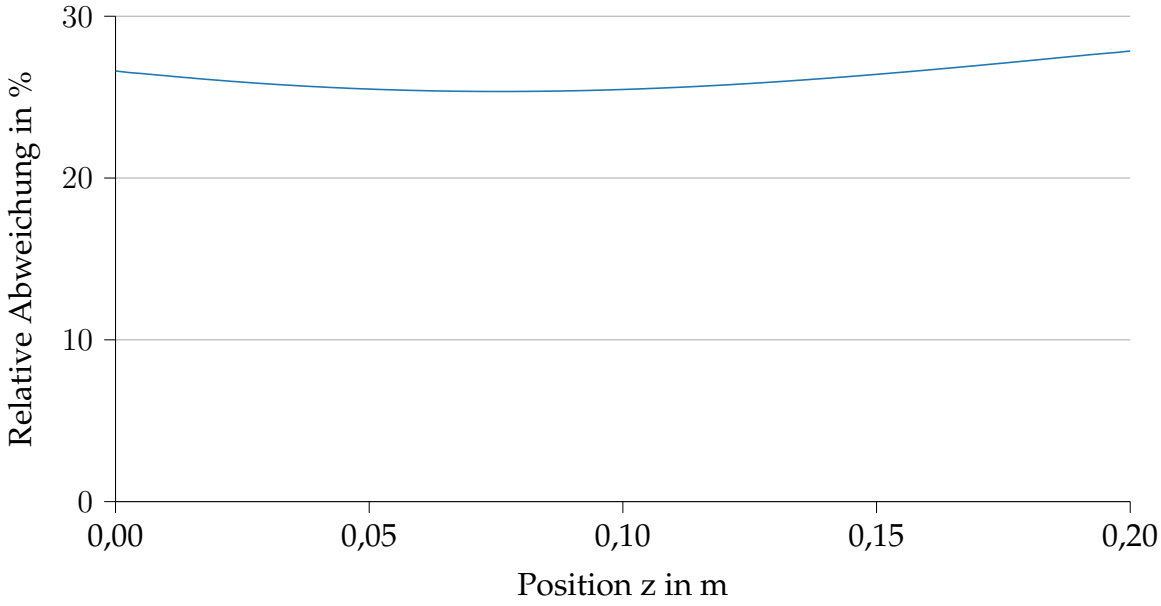


Abb. 4.11: Relative Abweichung der in ABAQUS und PYSPH unter Berücksichtigung aller Effekte berechneten Temperaturverteilung in der Stabmitte zum Zeitpunkt $t = 50\,\mathrm{s}$.

4.3 Modellierung in PYSPH

Der Aufbau des in PYSPH implementierten MEX-Modells wird in Abschnitt 3.3 beschrieben. In diesem Abschnitt werden die mit diesem Modell erhaltenen Ergebnisse vorgestellt.

4.3.1 Auswahl eines Schemas

In einem ersten Schritt soll aus den in Abschnitt 2.3.4 erläuterten Schemata ein geeignetes ausgewählt werden. Für das GTVF-Schema kann keine Parameterkombination gefunden werden, bei der die Simulation stabil ist. Deshalb beschränkt sich der folgende Abschnitt auf das WCSPH-, AHA- und EDAC-Schema. Auch die Simulationsparameter müssen angepasst werden. Analog zum Vorgehen in ABAQUS werden dafür kurze einzelne Stränge extrudiert. In PYSPH sind sowohl zwei- als auch dreidimensionale Rechnungen möglich, wodurch Parameterkombinationen schneller überprüft werden können. Im Folgenden werden Ergebnisse dreidimensionaler Simulationen vorgestellt. Analog zum Vorgehen in ABAQUS werden zunächst einzelne Stränge extrudiert. In PYSPH können deutlich mehr Parameter durch den Nutzer selbst gewählt werden. Im Vergleich zu ABAQUS ist es generell herausfordernder, eine stabile Parameterkombination zu finden.

WCSPH

Obwohl das WCSPH-Schema der SPH-Implementierung in ABAQUS vermutlich am nächsten kommt, ist die Simulation für eine mit der aus Abschnitt 4.1 vergleichbare Parameterkombination nicht stabil. Insbesondere die Viskosität muss unter dem zuvor eingesetzten Wert liegen. In Abbildung 4.12 ist die Ablage eines Strangs bei einer Viskosität von $\eta = 0{,}05\,\mathrm{Pa\,s}$ und einer Schallgeschwindigkeit von $c_\mathrm{s} = 1{,}2\,\mathrm{m/s}$ abgebildet. Wie in Abschnitt 4.1 werden die Extrusions- und Vorschubgeschwindigkeit zu $v_\mathrm{ex} = v_\mathrm{vor} = 60\,\mathrm{mm/s}$ gewählt. Der Düsendurchmesser beträgt auch hier $d_\mathrm{D} = 0{,}4\,\mathrm{mm}$. Die Fluidpartikel sind entsprechend ihrer Dichte eingefärbt. In dieser wie in allen folgenden Abbildungen, die die Dichte der Partikel zeigen, wird die Referenzdichte $rho_0 = 1072{,}7\,\mathrm{kg/m^3}$ weiß dargestellt. Partikel mit höherer Dichte werden rot, Partikel mit geringerer Dichte blau dargestellt. Die Grenzen der Farbskalen sind jeweils die maximalen und minimalen auftretenden Dichten. So können Bereiche, in denen die Partikel komprimiert, bzw. expandiert sind, einfach identifiziert werden. Die Partikel der Bauplattform sind in einem hellen, diejenigen, die den Einlass noch nicht verlassen haben in einem dunklen Grau dargestellt. Um die Stabilität der Simulation zu gewährleisten, muss der Abstand zwischen Düse und Bauplattform anders als zuvor auf $h_\mathrm{in} = 0{,}6\,\mathrm{mm}$ gesetzt werden. Der Partikelabstand ist auf ein Viertel des Düsenradius eingestellt. In der geschnittenen Seitenansicht (a) ist gut zu erkennen, dass die Partikel von Fluid und Bauplatte nicht unmittelbar aneinander angrenzen, sondern durch einen Spalt von ca. 0,1 mm getrennt sind. Dies entspricht dem halben Düsenradius bzw. dem Doppelten des Partikelabstands Δx. Da das Schema keine Haftbedingung zwischen Wand und Fluid enthält, verbleiben die Partikel nicht an jenem Ort, an dem sie auf die Plattform treffen, sondern werden von der Düse mitgeschleppt. Die Abstände zwischen den Partikeln variieren stark. Am Beginn des Strangs, in Abbildung 4.12 links, sind die Partikel ungeordnet. Im Mittel scheinen die Abstände dem vorgegebenen Δx zu entsprechen. Die später extrudierten Partikel sind dichter gepackt und behalten ihre durch den Auslass vorgegebene Ordnung. Sie erscheinen wie nebeneinanderliegenden Perlenketten. Obwohl die Kompaktierung der Partikel deutlich zu erkennen ist, spiegelt diese sich nicht in der berechneten Dichte wider. Sie weicht nur um etwa 6,5 % vom Referenzwert ρ_0 ab. Unmittelbar unterhalb des Einlasses bspw. entspricht die Dichte für die meisten Partikel fast dem Referenzwert, obwohl die Partikelabstände in diesem Bereich erkennbar zu gering sind. Partikelgruppen hoher und niedriger Dichte liegen teils unmittelbar aneinander. In der Draufsicht (b) ist die Plattform zur bessern Übersicht ausgeblendet. In dieser Ansicht wird das leichte Zerfließen zu den Seiten sichtbar.

AHA

Die Simulation wird mit dem AHA-Schema wiederholt. Die Viskosität kann etwa doppelt so hoch zu $\eta = 0{,}1\,\mathrm{Pa\,s}$ und die Düsenhöhe mit $h_\mathrm{in} = 0{,}4\,\mathrm{mm}$ etwas geringer gewählt werden.

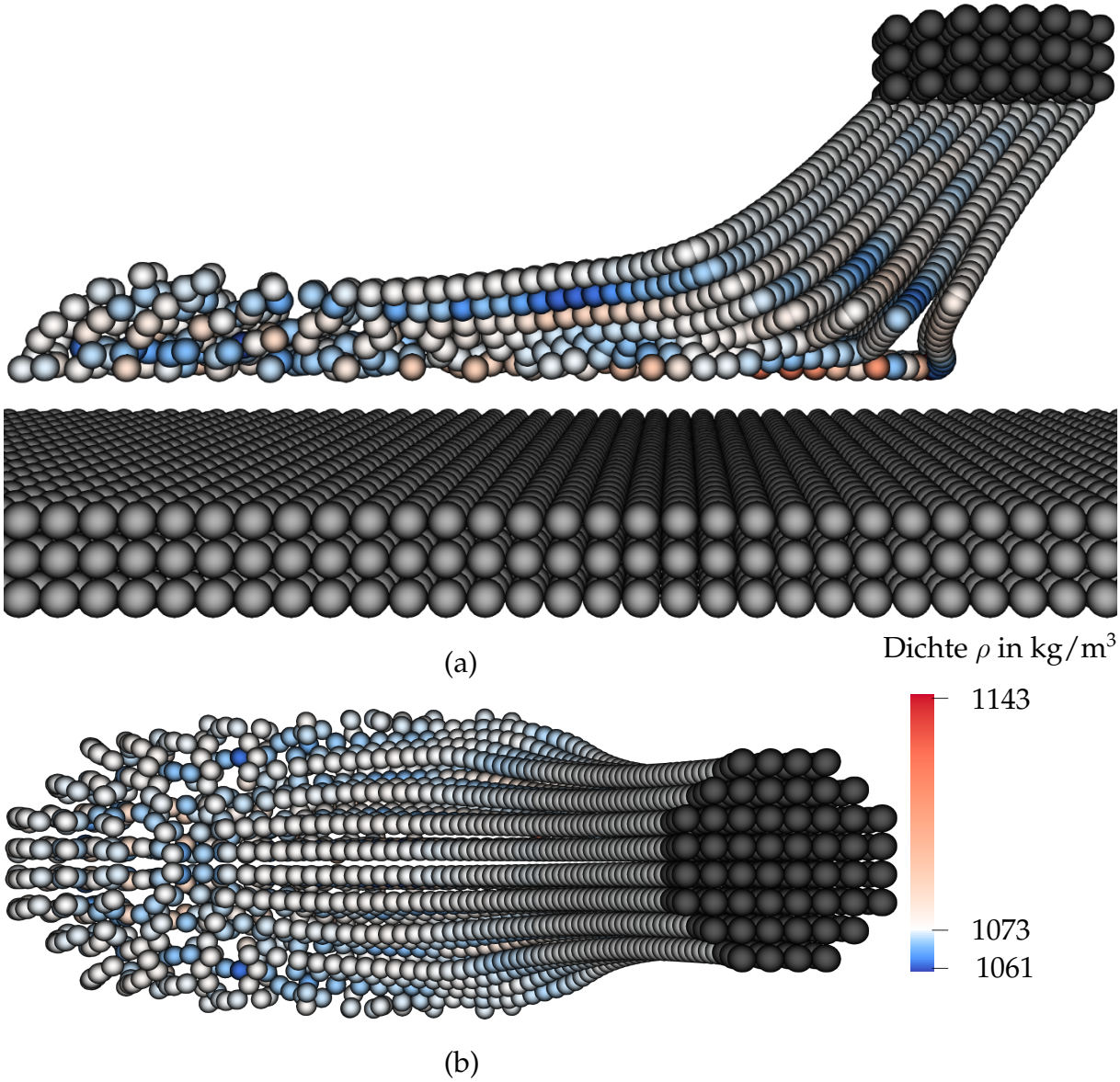


Abb. 4.12: Ablage eines Stranges mit dem WCSPH-Schema eingefärbt entsprechend der Dichte bei einer Viskosität von 0,05 Pa s in (a) geschnittener Seitenansicht und (b) Draufsicht mit ausgeblendeter Bauplattform.

Allerdings muss die Schallgeschwindigkeit auf $c_\mathrm{s} = 0{,}2\,\mathrm{m/s}$ reduziert werden. Die Ergebnisse sind in Abbildung 4.13 der Dichte entsprechend eingefärbt dargestellt. Durch die geschnittene Seitenansicht (a) wird der Effekt der Haftbedingung an der Bauplattform deutlich. Das Extrudat folgt nicht mehr widerstandslos der Düse. Der Strang hat die korrekte Länge. Die Partikelabstände sind lokal verschieden. Wie beim WCSPH entsprechen die Abstände am Beginn des Strangs in etwa dem Vorgegebenen. Es folgt ein Übergang, in dem die Partikel sehr dicht gepackt sind. Auffällig ist, dass die Dichte in diesem Bereich dennoch am geringsten ist. Auch in anderen

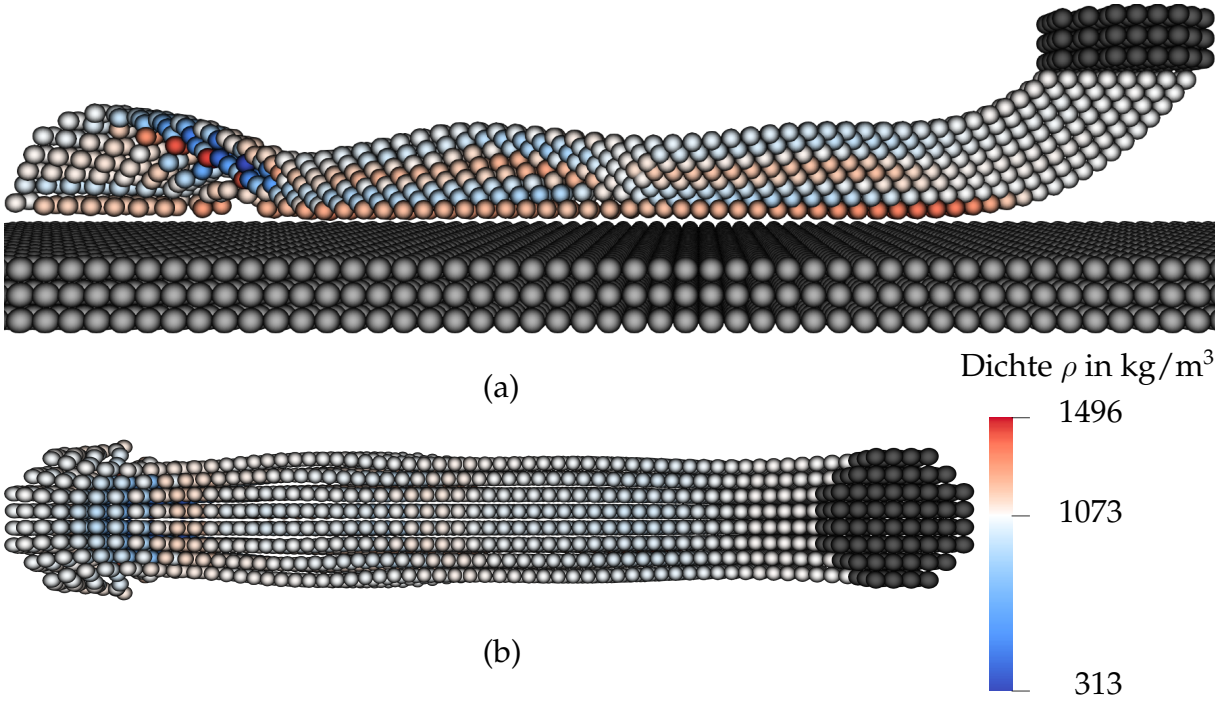


Abb. 4.13: Ablage eines Stranges mit dem AHA-Schema eingefärbt nach der Dichte bei einer Viskosität von 0,1 Pa s in (a) geschnittener Seitenansicht und (b) Draufsicht mit ausgeblendeter Bauplattform.

Bereichen der Simulation nimmt die Dichte Werte an, die aufgrund der Nähe der Partikel zueinander nicht zu erwarten sind. Beinahe im gesamten Strang sind die Partikel dichter gepackt als vorgegeben. Wie zuvor scheinen die Partikel in Ketten angeordnet. Die Dichte weicht teils um mehr als 70 % vom Referenzwert ρ_0 ab. Generell scheinen die Partikel kaum voneinander abgleiten zu können, sondern verharren in der Konfiguration, in der sie aus der Düse austreten. Für Durchläufe mit höheren Viskositäten ist die Kompaktierung der Partikel nochmals stärker. Es ergibt sich nur noch ein sehr dünner Strang. Die Draufsicht (b) zeigt zudem, dass der Strang nur unwesentlich breiter als der Düsenaustritt ist. Nur zu Beginn des Strangs verteilen die Partikel sich zu den Seiten.

EDAC

Als dritte Variante wird das EDAC-Schema betrachtet. Da hier keine herkömmliche EOS gelöst wird, wirkt sich die künstliche Schallgeschwindigkeit in deutlich geringerem Maße auf das Ergebnis und die Stabilität aus und wird auf $c_\mathrm{s} = 6\,\mathrm{m/s}$ festgelegt. Die Viskosität wird zu $\eta = 0{,}1\,\mathrm{Pa\,s}$ gewählt. Die Extrusion eines einzelnen Stranges ist in Abbildung 4.14 dargestellt. Die Partikelverteilung in der geschnittenen Seitenansicht (a) ähnelt stark der mit

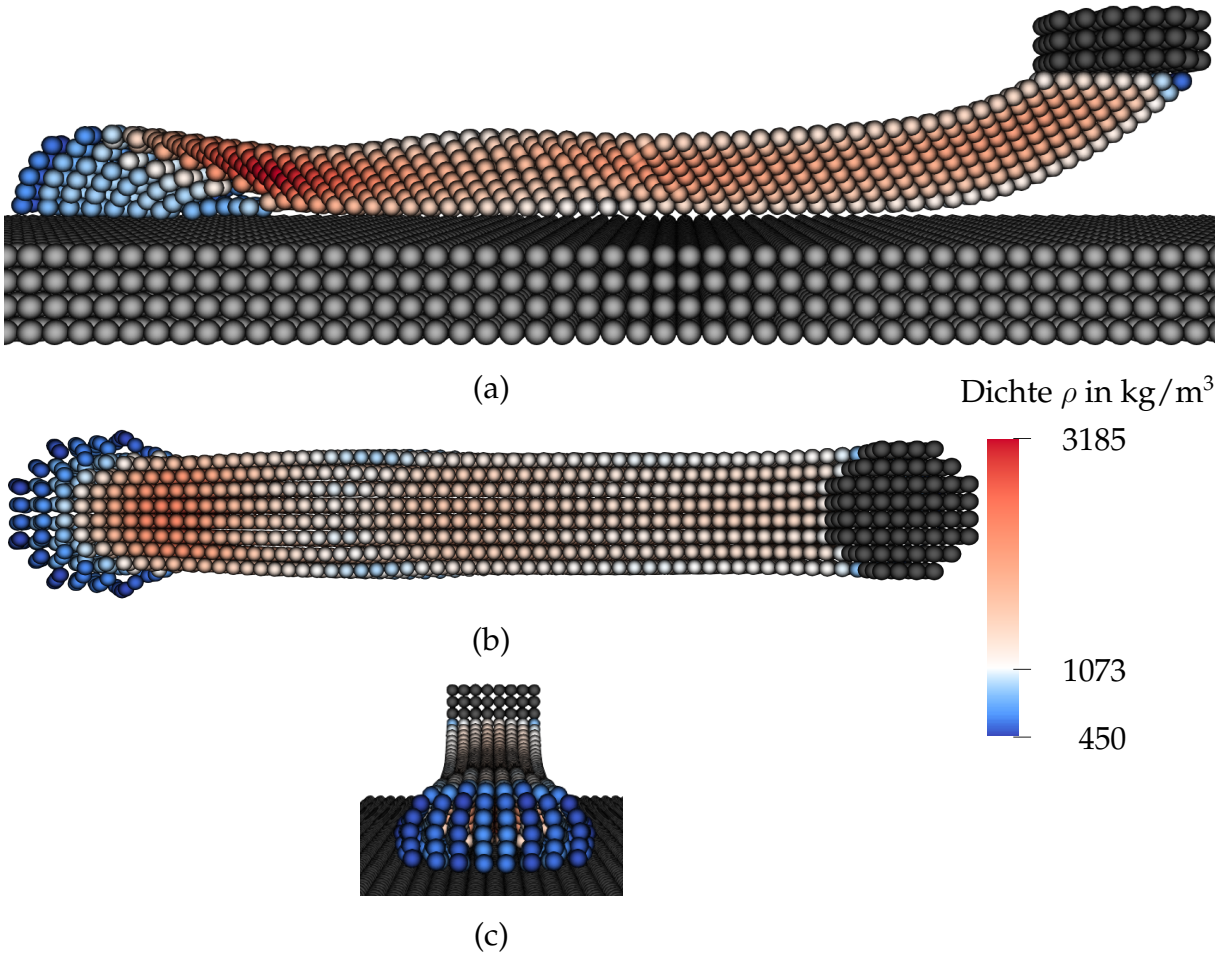


Abb. 4.14: Ablage eines Stranges mit dem EDAC-Schema eingefärbt entsprechend der Dichte bei einer Viskosität von 0,1 Pa s in (a) geschnittener Seitenansicht und (b) Draufsicht mit ausgeblendeter Bauplattform.

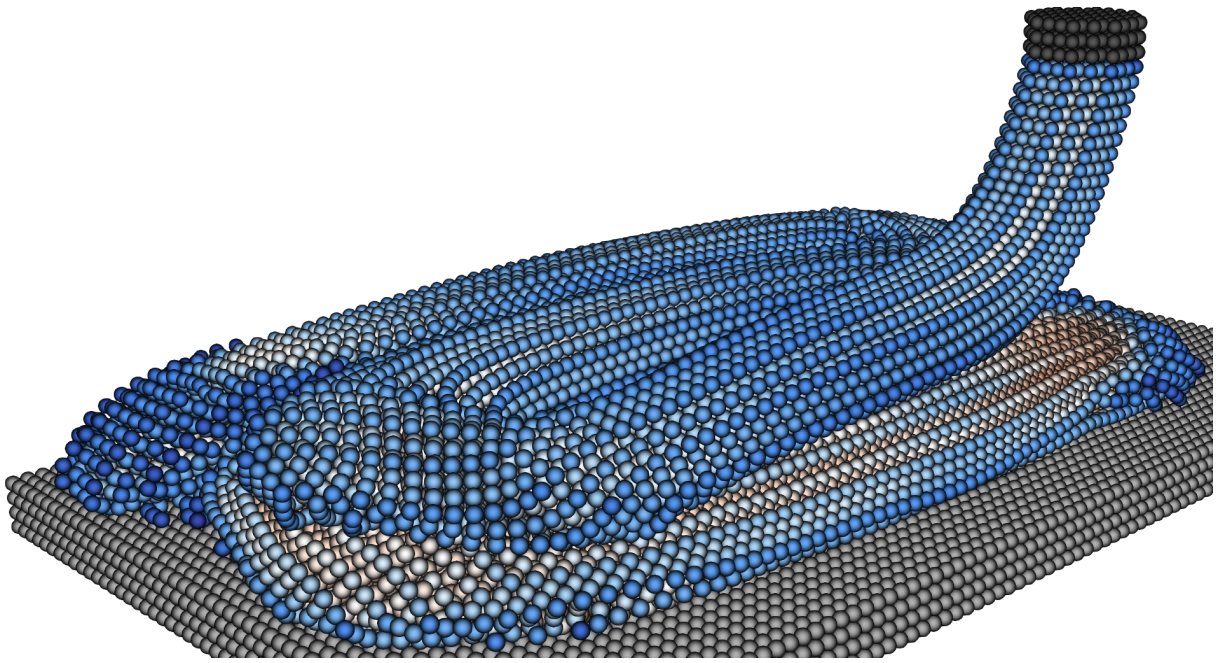
dem AHA-Schema erhaltenen aus Abbildung 4.13 (a). Auch hier sind die Partikel am Beginn des Strangs in etwa so weit voneinander entfernt, wie zu erwarten ist. Im weiteren Verlauf sind sie erkennbar kompaktiert. Erneut sind die Partikel am Übergang direkt hinter dem Strangbeginn am dichtesten gepackt. Obwohl die Partikel augenscheinlich sehr ähnlich verteilt sind, ergibt sich mit dem EDAC-Schema eine gänzlich andere Dichteverteilung. Es besteht ein eindeutiger Zusammenhang zwischen den Partikelabständen und der Dichte. Die Bereiche des Fluids, in denen die Partikel eng beieinander liegen, weisen eine hohe Dichte auf und umgekehrt. Im Bereich der stärksten Kompaktierung ist die berechnete Dichte bis zu dreimal so hoch wie der Referenzwert ρ_0. Für höhere Viskositäten ist eine unverhältnismäßig stärkere Kompaktierung der Partikel zu beobachten. Die Höhe des Strangs reduziert sich dadurch erheblich. Die gewählte Viskosität stellt einen guten Kompromiss dar, um diese starke Kompression einerseits und das schnelle Zerfließen des Strangs andererseits zu beschränken. Dass der Strang, zumindest in der

kurzen Simulationszeit von ca. $t \approx 0{,}047\,\mathrm{s}$, nur geringfügig zu den Seiten zerfließt, zeigen die Draufsicht (b) und die Rückansicht (c).

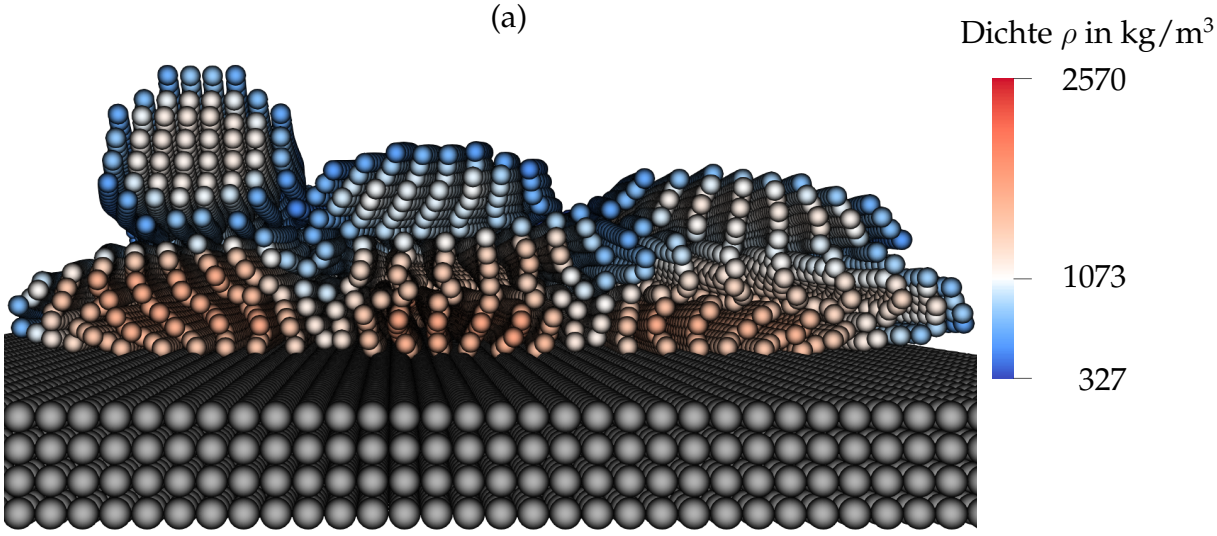
4.3.2 Ablegen mehrerer Schichten

Analog zum Vorgehen in Abschnitt 4.1.2 soll ein Körper aus drei Schichten, die aus jeweils drei Strängen bestehen, extrudiert werden. Dazu wird das EDAC-Schema verwendet. Die Viskosität und Schallgeschwindigkeit aus Abschnitt 4.3.1 werden beibehalten. Für Schichtdicke d_S, Versatz zwischen den Strängen b_S und Stranglänge l_S werden dieselben Größen wie in Abschnitt 4.1.2 eingesetzt. Abbildung 4.15 zeigt den extrudierten Körper in einer Übersicht (a) und einen Querschnitt (b). Die Partikel sind entsprechend der Dichte eingefärbt. In der Übersicht (a) ist zu erkennen, dass die Stränge durch die längere Simulationszeit von $t \approx 0{,}47\,\mathrm{s}$ und die geringe Viskosität ihre Form nicht mehr beibehalten. Sie zerfließen zu den freien Seiten hin. Die Form des zuletzt abgelegten Stranges ist deutlich zu erkennen. Insbesondere die Stränge der unteren beiden Schichten behalten ihre Form jedoch nicht und lassen sich nicht mehr klar voneinander abgrenzen. Die Lücken zwischen den Strängen werden weitgehend gefüllt. Der extrudierte Körper ist breiter und länger als beabsichtigt. Noch deutlicher wird das Zerfließen der Struktur im Querschnitt (b). Oben links ist der zuletzt abgelegte Strang zu sehen. Dieser hat in etwa denselben Querschnitt wie die Partikel des Einlasses. Die beiden zuvor extrudierten Stränge der Schicht sind noch deutlich zu erkennen und abzugrenzen. Im Gegensatz dazu können die Stränge der beiden unteren Schichten nur noch von ihren Nachbarn innerhalb einer Schicht klar unterschieden werden, da die Lücken nicht vollständig verfüllt sind. Die übereinander liegenden Stränge der beiden unteren Schichten sind jedoch nicht mehr eindeutig voneinander abzugrenzen. Weiter ist zu sehen, dass die Partikel auch mit dem EDAC-Schema in Ketten bzw. Lamellen angeordnet sind, anstatt sich frei zu verteilen. Die gesamte Struktur ist deutlich niedriger als das dreifache der eingestellten Schichthöhe. Hauptursache dafür ist das Fließen des Fluids zu den Seiten und in die Lücken zwischen den Strängen. Allerdings ist die Dichte insbesondere im unteren Teil der Struktur erhöht bzw. die Partikelabstände geringer als vorgegeben. Die Inkompressibilität des Fluids ist nicht gewährleistet. Im Vergleich zur Extrusion eines einzelnen Strangs fallen die Dichteunterschiede geringer aus. Offenbar ist die Expansion in Bereichen erhöhter Dichte durch die Viskosität behindert und kann deshalb erst bei längerer Simulationszeit beobachtet werden. Die maximale Dichte ist etwa 140 % höher als der Referenzwert ρ_0.

Die Rechenzeit der Simulation beträgt etwa 3 h. Gerechnet wurde auf allen acht Kernen eines Intel *i7-11800H*. Da die Viskosität auf $\eta = 0{,}1\,\mathrm{Pa\,s}$ beschränkt werden musste, berechnet sich



(a)



(b)

Abb. 4.15: Ablage dreier Schichten aus je drei Strängen mit dem EDAC-Schema eingefärbt entsprechend der Dichte bei einer Viskosität von 0,1 Pa s. (a) zeigt den gesamten Körper und (b) einen Querschnitt.

die Reynoldszahl des Problems zu

$$Re = \frac{\rho v_{\mathrm{ex}} d_{\mathrm{D}}}{\eta} \approx 0,26. \tag{4.3}$$

4.3.3 Oberflächenerkennung

In Abschnitt 3.3.5 ist der zur Erkennung der freien Oberfläche implementierte Algorithmus beschrieben. Dieser wird benötigt, um die Wärmeverluste durch Konvektion und Strahlung berechnen zu können. Das Maß für die Oberfläche, die ein Partikel repräsentiert, ist der Wert λ. Für gänzlich freie Partikel ohne Nachbarn strebt λ gegen 0, für Partikel, die keinerlei freie Oberfläche repräsentieren gegen 1. Der Übergangswert, bei dem ein Partikel gerade keine freie Oberfläche repräsentiert, ist 0,75. In Abbildung 4.16 ist der mit dem EDAC-Schema extrudierte Körper in zwei Schnitten dargestellt. In (a) sind drei übereinander liegende Stränge längs geschnitten. (b) zeigt den Querschnitt durch alle Stränge. Die Partikel sind entsprechend ihres λ-Wertes eingefärbt. Alle Partikel innerhalb des Fluids, für die also gilt $1,0 \geq \lambda > 0,75$, sind rot. Partikel, für die gilt $\lambda = 0$, die also gerade keine Oberfläche repräsentieren, sind weiß eingefärbt. Die Partikel, die als Teil der Oberfläche erkannt werden, sind mit zunehmender Oberfläche in kräftigeren Blautönen dargestellt. Im Schnitt (a) stimmen die λ-Werte der Partikel größtenteils mit den Erwartungen überein. Die Partikel, die gerade die Düse verlassen haben, werden korrekt identifiziert. Hier bildet genau eine Partikelschicht die freie Oberfläche. In den zuvor extrudierten Bereichen kommt es teilweise vor, dass zwei übereinanderliegende Partikelschichten der freien Oberfläche zugeordnet werden. Allerdings haben die Partikel der innen liegenden Schicht hohe λ-Werte, sodass ihnen eine sehr kleine Oberfläche zugeordnet wird. Auch zwischen den Schichten wird korrekt erkannt, dass die freie Oberfläche von der obenliegenden Schicht bedeckt wird. Dies kann man gut unterhalb der Düse erkennen, wo der neue Strang den alten bedeckt und sich die freie Oberfläche reduziert. Vereinzelt finden sich Partikel, die scheinbar innerhalb des Fluids liegen, aber dennoch als Teil der freien Oberfläche erkannt werden. Dies ist insbesondere am Strangbeginn, in Abbildung 4.16 links, der Fall. In diesem Bereich knickt die Bahn der Düse allerdings ab, wodurch möglicherweise eine Lücke entsteht. Auch der Bereich, in dem das Fluid mit der Bauplattform in Kontakt steht, wird korrekterweise nicht der freien Oberfläche zugeordnet. Im Schnitt (b) sind einige offensichtliche Fehler des Algorithmus sichtbar. In der Abbildung mittig und rechts gibt es zwei Bereiche, die fälschlicherweise der freien Oberfläche zugeordnet werden. In beiden Bereichen ist auffällig, dass die Partikel in Längs- und Hochrichtung sehr dicht gepackt sind. Sie erscheinen in Lamellen angeordnet. Diese sind jedoch in Querrichtung deutlich weiter voneinander entfernt. Die Oberflächenerkennung scheint diese Bereiche nicht korrekt einordnen zu können.

4.3.4 Thermisches Modell

Durch der Oberflächenerkennung und das in Abschnitt 3.3.4 beschriebene thermische Modell kann die Temperaturverteilung im Fluid berechnet werden. Abbildung 4.17 zeigt zwei Ansichten

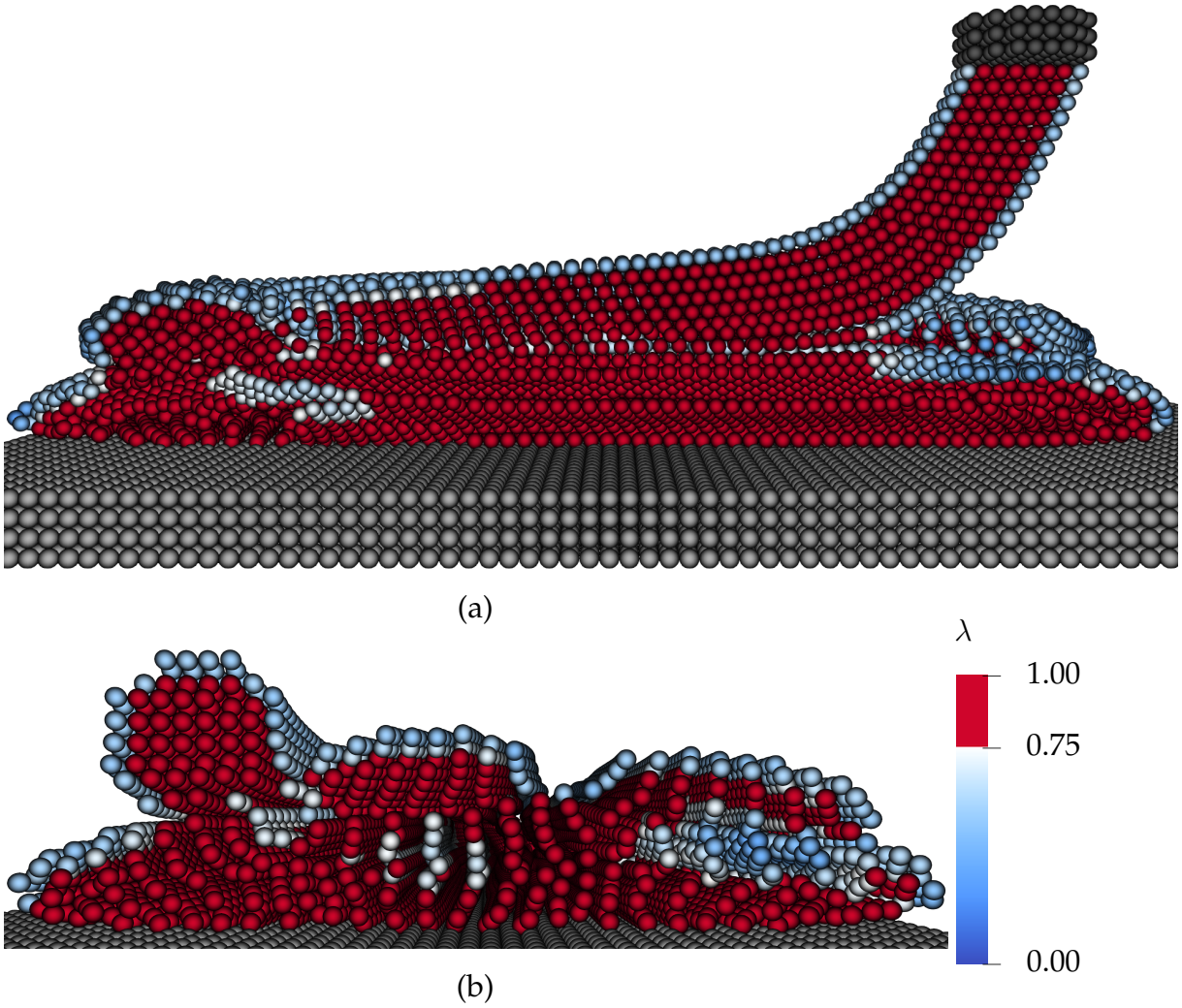


Abb. 4.16: Ablage dreier Schichten aus je drei Strängen mit dem EDAC-Schema eingefärbt entsprechend λ. (a) zeigt einen Schnitt längs und (b) quer durch den Körper.

des extrudierten Körpers, in denen die Partikel entsprechend ihrer Temperatur eingefärbt sind. Im Längsschnitt (a) ist zu sehen, wie die Partikel die Düse mit der vorgegebenen Temperatur von 220 °C verlassen. Die Partikel, die in direktem Kontakt mit der Bauplattform stehen, haben sich bereits stark deren Temperatur von 55 °C angenähert. Dazwischen ergibt sich ein stetiger Temperaturgradient. Die Oberflächenpartikel sind nur geringfügig kühler als die Partikel innerhalb des Fluids. Der Vorgang wird von der konduktiven Wärmeleitung dominiert. Aus diesem Grund sind im Querschnitt (b) die fälschlich als Oberflächenpartikel erkannten Partikel, die tatsächlich im Inneren des Fluids liegen, nicht zu erkennen.

In Abbildung 4.18 sind noch einmal dieselben Schnitte dargestellt. Die Partikel sind hier entsprechend der zeitlichen Änderung der Temperatur $\dot{T}$ eingefärbt. Die Temperatur roter Partikel steigt, blaue Partikel werden kälter und weiße Partikel erfahren keine Temperaturänderung. Im

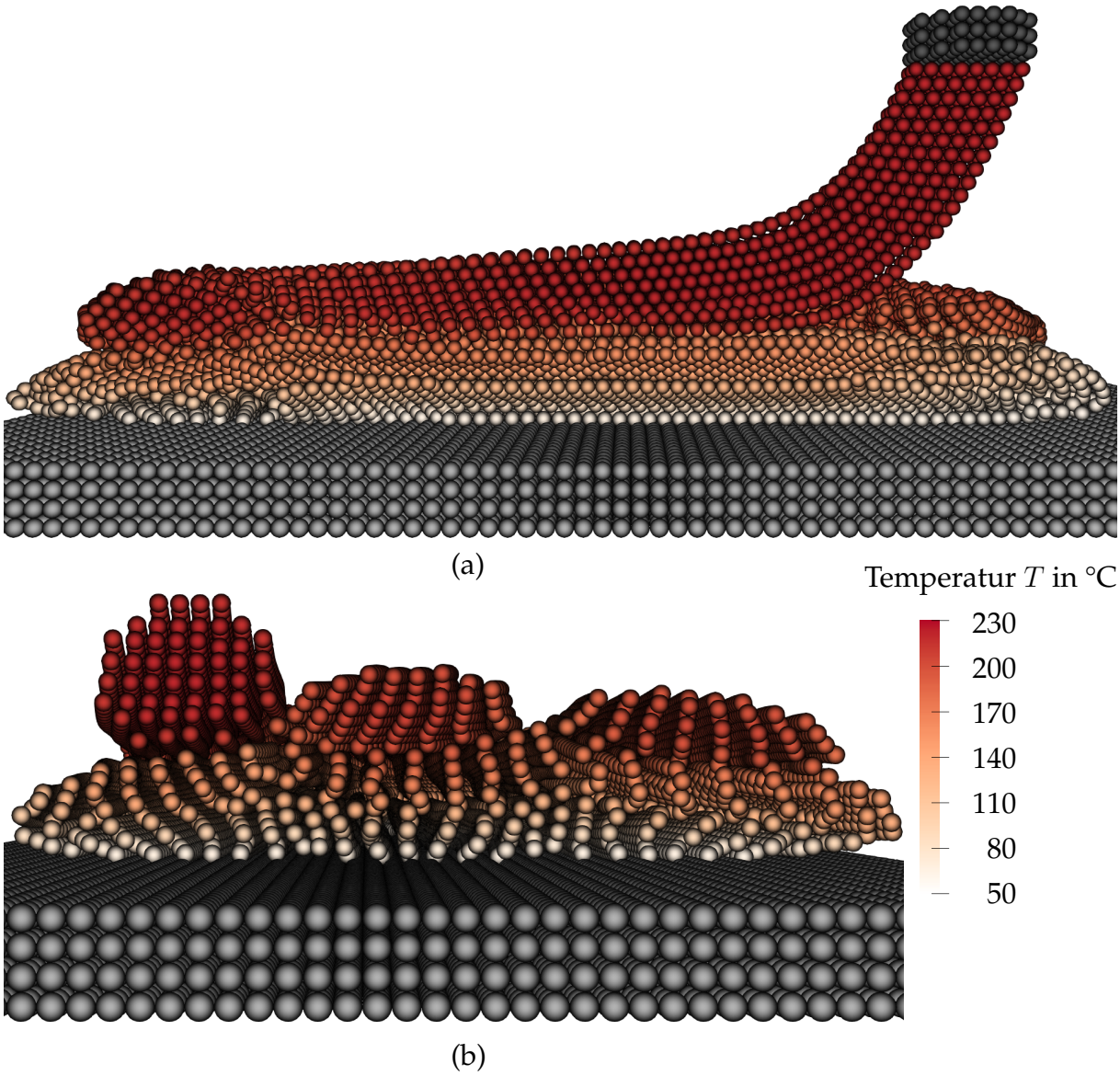


Abb. 4.17: Ablage dreier Schichten aus je drei Strängen mit dem EDAC-Schema eingefärbt entsprechend der Temperatur. (a) zeigt einen Schnitt längs und (b) quer durch den Körper.

Längsschnitt (a) ist direkt unterhalb der Düse gut der Einfluss der Konvektion und Strahlung zu erkennen. Die äußere Schicht des Extrudats verliert erheblich schneller an Temperatur als die Partikel im Inneren. Ebenfalls bemerkenswert ist der Kontakt des zuletzt abgelegten Strangs mit der darunterliegenden Ebene. Die mittlere Schicht ist bereits deutlich kühler als das Extrudat aus der Düse. So kommt es zu einem schnellen Wärmeaustausch zwischen den beiden Schichten. Im Querschnitt (b) sind deutlich die irrtümlich als Teil der Oberfläche identifizierten Bereiche in Abbildung 4.18 mittig und rechts zu erkennen (siehe Abbildung 4.16 (b) für einen Vergleich

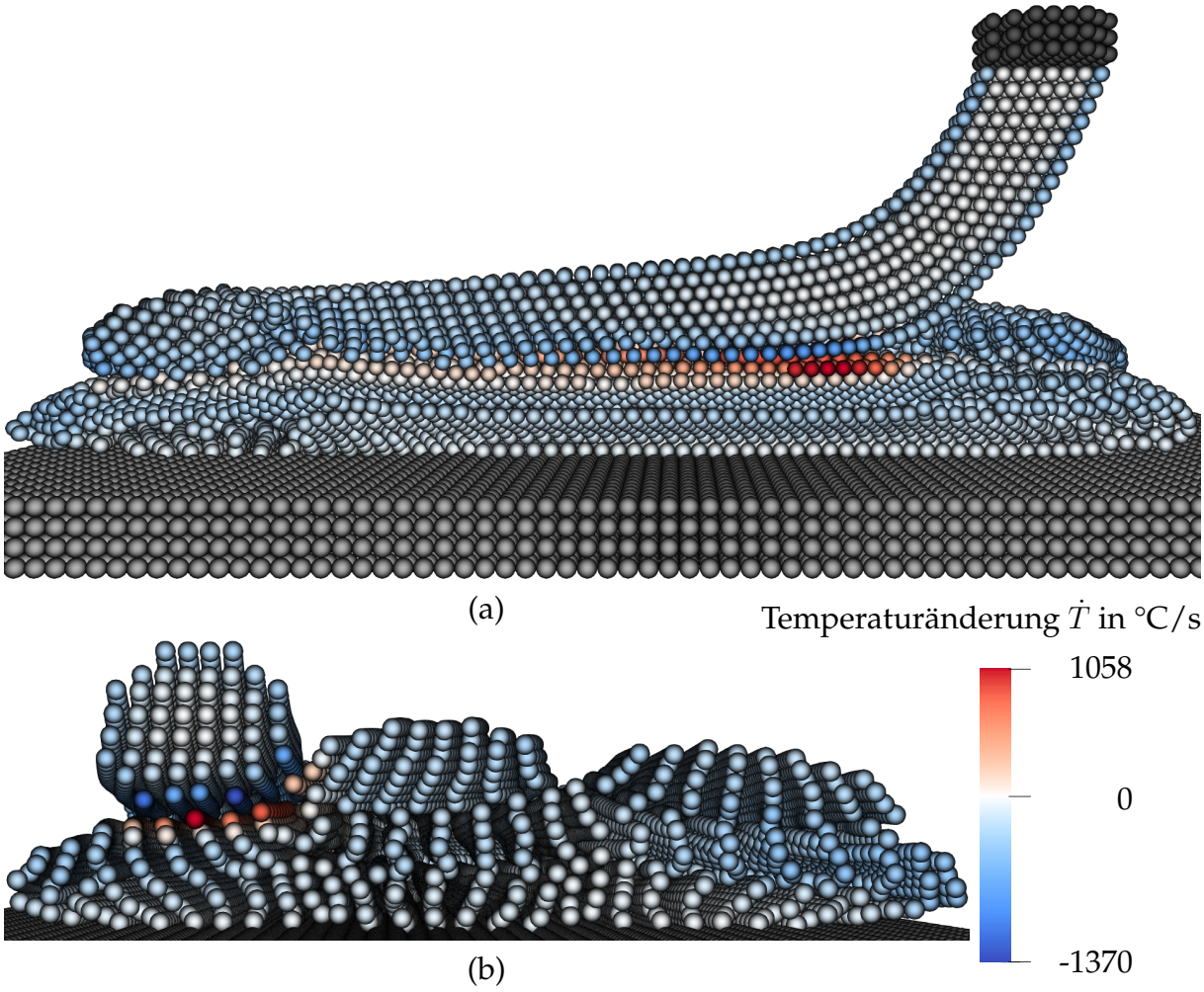


Abb. 4.18: Ablage dreier Schichten aus je drei Strängen mit dem EDAC-Schema eingefärbt entsprechend der Temperaturänderung. (a) zeigt einen Schnitt längs und (b) quer durch den Körper.

mit der Verteilung für λ). Sie verlieren mehr Wärme, als umliegende und korrekt identifizierte Partikel.

4.3.5 Viskosität

In Abschnitt 3.3.6 wurde die temperatur- und scherratenabhängige Berechnung der Viskosität erläutert. In Abbildung 4.19 ist der Querschnitt des extrudierten Körpers abgebildet. Die Partikel sind entsprechend der auf Basis der Scherrate und des in Abbildung 4.17 dargestellten Temperaturverlaufs berechneten Viskosität η eingefärbt. Die Viskosität ist von der Temperatur dominiert. Der Einfluss der Scherrate ist kaum zu erkennen, da diese im vorliegenden Fall gering ist. Einzig unmittelbar hinter dem Einlass erreicht sie Werte von bis zu 130 1/s. Der

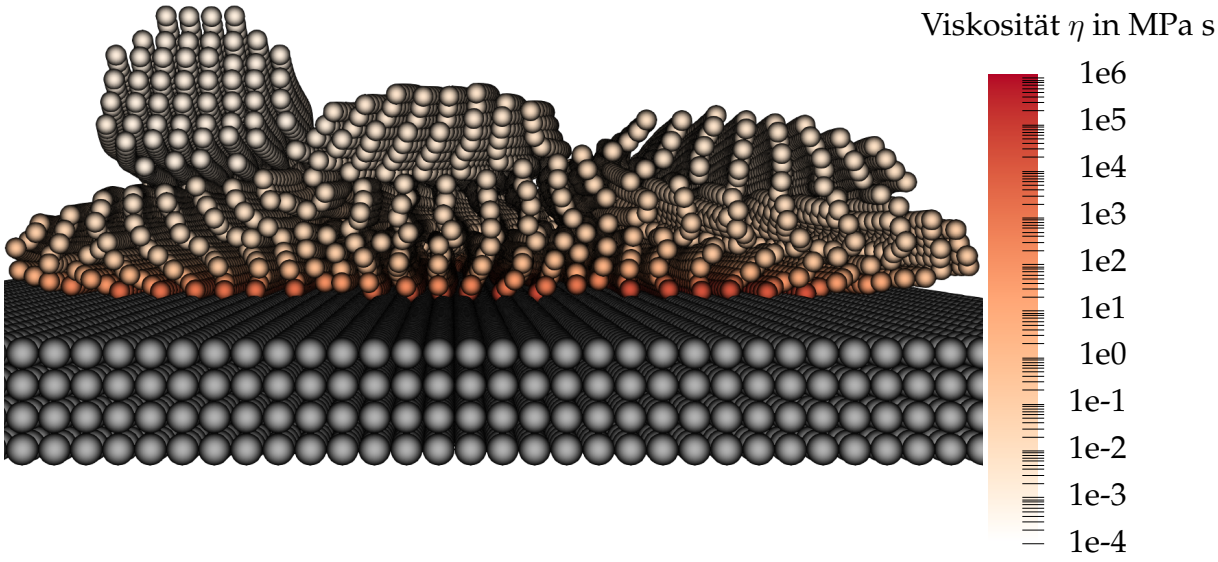


Abb. 4.19: Ablage dreier Schichten aus je drei Strängen mit dem EDAC-Schema im Querschnitt eingefärbt entsprechend der temperatur- und scherratenabhängigen Viskosität.

Verlauf der Viskosität ist daher dem Temperaturverlauf sehr ähnlich. Im unteren Bereich ist das Fluid bereits weit abgekühlt und die Viskositäten liegen sehr hoch. Je später ein Strang abgelegt wurde, desto geringer ist die Viskosität. Der verwendete Cross-WLF-Ansatz gilt allerdings nur oberhalb der durch Kurvenanpassung bestimmten Glasübergangstemperatur T^*, die in großen Teilen der Simulation unterschritten wird. Die extrem hohen Viskositäten nahe der Bauplatte sind deshalb nicht korrekt. Da die Simulation zudem mit keinem der getesteten Schemata für so hohe Viskositäten stabil ist, fließen die berechneten Viskositäten bisher nicht in die Berechnung der Strömungsgrößen ein. Stattdessen wird die Viskosität auf einen durch den Nutzer gewählten Wert einheitlich für alle Partikel festgelegt.

5 Diskussion

5.1 Modellierung in ABAQUS

In Abschnitt 3.2 wird der Aufbau eines MEX-Prozessmodells in ABAQUS beschrieben. Unter 4.1 sind die aus diesem Modell erhaltenen Ergebnisse aufgeführt. Durch das Extrudieren einzelner kurzer Polymerstränge können mit geringem Aufwand physikalische und numerische Modellparameter überprüft werden. So wird in Abschnitt 4.1.1 eine geeignete Parameterkombination bestimmt. Dabei besteht ein Zielkonflikt zwischen der Genauigkeit der Ergebnisse und dem Rechenaufwand. Eine höhere künstliche Schallgeschwindigkeit c_s und Viskosität η, erfordern kürzere Zeitschritte, damit die Stabilität der Simulation gewährleistet ist. Eine zu geringe Schallgeschwindigkeit führt zu unphysikalischem Verhalten des Fluids, wie etwa der Durchdringung der Bauplattform oder starkem Kompaktieren der Partikel unmittelbar unter dem Düsenaustritt. Um die Rechenzeit zu beschränken, wird die Schallgeschwindigkeit gerade hoch genug gewählt, um diese Phänomene einzuschränken. Allerdings sind im als inkompressibel angenommenen Fluid relativ große Dichteunterschiede von bis zu 20 % zu beobachten. Auch die Viskosität wird auf einen Wert beschränkt, der mehrere Größenordnungen unter dem physikalischen liegt. Diese Beschränkungen sind notwendig, um den Rechenaufwand auf ein vertretbares Maß zu reduzieren. Verschiedene Beobachtungen lassen jedoch den Schluss zu, dass die Genauigkeit der Simulation dadurch nicht ausreichend ist. Ein SPH Schema aus der Gruppe der geringfügig kompressiblen muss prinzipbedingt kleine Dichteänderungen zulassen. In der Simulation treten jedoch Abweichungen von der Referenzdichte von bis zu 20 % auf, sodass die Annahme der Inkompressibilität nicht mehr zutrifft. So kann mit den gewählten Parametern der Abstand zwischen Düse und Bauplattform nicht sehr klein gewählt werden. Dies ist in der Anwendung des Verfahrens durchaus üblich, in der Simulation führt es jedoch zu starker Kompression im Kontakt mit der Platte (s. Abbildung 4.3). Das Auftrennen des Strangs und die großen Dichteunterschiede sind vermutlich Folgen dieser Kompression. In derselben Simulation kann außerdem die der Düse vorgelagerte Fließfront nicht reproduziert werden. Die Partikel werden lediglich kompaktiert, anstatt in Vorschubrichtung auszuweichen. Bei der Extrusion eines einfachen Körpers aus neun Strängen (s. Abbildung 4.5) ist die Dichte in früher

extrudierten Strängen geringer und fällt teilweise unter den Referenzwert ρ_0. Die Partikel scheinen sich voneinander zu entfernen. Auch dieses Phänomen hat keine physikalischen Gründe. Vermutlich ist auch diese Beobachtung durch die hohe Kompressibilität des Fluids zu erklären. All diese Beobachtungen sind Hinweise darauf, dass die Schallgeschwindigkeit deutlich höher gewählt werden muss. Womöglich können diese unerwünschten Effekte auch durch eine höhere räumliche Auflösung der Simulation, also einem geringeren Partikelabstand reduziert werden, was den Rechenaufwand ebenfalls erhöht. Weiter wäre es vorteilhaft, mit physikalisch korrekten Viskositäten rechnen zu können. Auch dies ist aufgrund der Limitierung des Zeitschritts nicht praktikabel.

Das Modell ist außerdem in seinen Möglichkeiten durch das Simulationswerkzeug ABAQUS beschränkt. Dem Nutzer steht nur ein SPH-Schema zur Verfügung und es können keine weiteren Formulierungen implementiert werden. Es kann nur die Bauplattform, nicht aber der Partikelgenerator, bewegt werden. Es ist, insbesondere für eine geringe Viskosität und hohe Vorschubgeschwindigkeit, nicht auszuschließen, dass dies einen Einfluss auf die entstehende Struktur hat. Hinzu kommen die derzeit nicht erklärbaren Abweichungen zwischen den vorgegebenen und tatsächlich angefahrenen Positionen der Bauplattform. Das Fehlen des Freiheitsgrades *NT11* macht die Modellierung des Wärmeübergangs sowie der Temperaturverteilung unmöglich. Sowohl der Wärmeaustausch innerhalb des Fluids, sowie mit der Bauplattform und der weiteren Umgebung können nicht modelliert werden. Deren Einfluss ist aber entscheidend für die Ergebnisse des MEX-Prozesses, da Materialwerte wie Dichte und Viskosität temperaturabhängig sind.

5.2 Thermisches Modell in PySPH

Die Ergebnisse aus Abschnitt 4.2.1 zeigen, dass die konduktive Wärmeleitung durch das in PySPH eingeführte Modell mit vergleichbarer Genauigkeit wie durch das kommerzielle FEM-Werkzeug ABAQUS abgebildet werden kann. Der konduktive Wärmeübergang zwischen zwei Körpern wird durch den Testfall nicht abgebildet. Allerdings wird für solche Wärmeübergänge in ABAQUS ein eigenes Kontaktmodell verwendet [106], wohingegen SPH-Partikel verschiedener Körper über denselben Mechanismus Wärme austauschen können, wie Partikel innerhalb eines Körpers. Bei Verwendung der SPH-Methode müssen deshalb keine zusätzlichen Kontaktbedingungen definiert werden. ABAQUS ist deshalb nur bedingt zur Validierung des Wärmeübergangs im Kontakt geeignet. Der Vergleich der Temperaturen unter zusätzlicher Berücksichtigung der Wärmestrahlung und Konvektion in Abschnitt 4.2.2 zeigt, dass diese zusätzlichen Effekte durch das thermische Modell in PySPH deutlich überschätzt werden. Eine mögliche Erklärung hierfür ist, dass der in Abschnitt 3.3.5 beschriebene Algorithmus zur Oberflächenerkennung die Ober-

fläche der Partikel überschätzt. Der Vergleich der realen und errechneten freien Oberflächen für eine einfache Geometrie in Abbildung 3.3 bestätigt, dass die Oberfläche durch den linearen Zusammenhang zu λ signifikant überschätzt wird. Da die Oberfläche linear mit dem Wärmestrom über den Rand zusammenhängt, hat dieser Fehler einen großen Einfluss auf die Genauigkeit des thermischen Modells. Die Berechnung der Oberfläche sollte deshalb verbessert werden. Durch die Betrachtung weiterer geometrisch einfacher Testfälle mit bekannten Oberflächen kann bspw. ein empirischer Zusammenhang zwischen λ und der Oberfläche eines Partikels gefunden werden, der genauer als der hier gewählte lineare ist. Zudem ist der implementierte Algorithmus nicht in der Lage, geschlossene Kavitäten, in denen keine Kühlung durch Strahlung und Konvektion stattfindet, als solche zu erkennen. Allerdings ist auch mit einer stark verbesserten Oberflächenberechnung zu erwarten, dass Wärmestrahlung und Konvektion durch die SPH-Methode überschätzt werden. In der FEM liegen Knoten auf der Oberfläche der Geometrie. Die Temperaturwerte an diesen Knoten repräsentieren die Oberflächentemperatur. Die äußere Schicht Partikel eines SPH-Modells hingegen repräsentiert nicht die Oberfläche. Vielmehr liegen die Simulationsgrößen im Fluid bzw. Körper vor. Auch dies führt zur Überschätzung des Wärmeaustausches mit der Umgebung, da mit einer höheren Temperaturdifferenz gerechnet wird. Der aus diesem Umstand resultierende Fehler ist wahrscheinlich deutlich geringer als die ungenaue Berechnung der Oberfläche. Zur Reduktion dieses zweiten Fehlers könnte bspw. der Temperaturgradient hinzugezogen werden, um die tatsächliche Oberflächentemperatur anzunähern. Da die Bestimmung der Koeffizienten für Wärmestrahlung und konvektiven Wärmeübergang ebenfalls mit großer Unsicherheit verbunden ist, ist der Fehler des thermischen Modells zunächst akzeptabel.

5.3 Modellierung in PySPH

Das in PySPH implementierte MEX-Prozessmodell wird in Abschnitt 3.3 beschrieben und die Ergebnisse in Abschnitt 4.3 vorgestellt. Das Werkzeug enthält bereits mehrere SPH-Formulierungen, die mit relativ geringem Aufwand gegeneinander ausgetauscht werden können. Um von den verfügbaren expliziten Schemata ein geeignetes auszuwählen und jeweils passende Parameterkombinationen zu finden, werden kurze Stränge extrudiert. Stabile Parameterkombinationen können für das WCSPH-, AHA- und EDAC-Schema gefunden werden. Das WCSPH-Schema ist für die Modellierung der MEX nur bedingt geeignet, da hier eine Haftbedingung für Wände fehlt. Die Abweichungen der Dichte von der Referenzdichte sind geringer als im Abaqus-Modell. Allerdings werden die Partikel stark kompaktiert. Dies scheint sich wiederum nicht wie erwartet in der berechneten Dichte widerzuspiegeln. Die Kompaktierung tritt ebenfalls im AHA- und EDAC-Schema auf und verstärkt sich für höhere Viskositäten. Einzig bei Verwen-

dung des EDAC-Schemas passt jedoch die Dichte zu den bei Betrachtung der Partikelabstände erwarteten Werten. Insbesondere aus diesem Grund wird das EDAC-Schema für die Extrusion eines Körpers aus drei Schichten, aus je drei Strängen verwendet. Auch in dieses Modell können keine realistischen Viskositäten eingesetzt werden. Im Vergleich zum ABAQUS-Modell wird die Viskosität nochmals um den Faktor zehn auf $\eta = 0{,}1\,\mathrm{Pa\,s}$ reduziert. Dies führt dazu, dass die extrudierten Stränge innerhalb kurzer Zeit stark zerfließen. In Verbindung mit der hohen Kompressibilität des Fluids entsteht deshalb ein Körper, der nicht der gewünschten Gestalt entspricht (siehe Abbildung 4.15). Das Modell ist deshalb in seiner aktuellen Form nur eingeschränkt in der Lage, den MEX-Prozess abzubilden. Voraussetzung hierfür ist eine SPH-Formulierung, die die Inkompressibilität des Fluids gewährleistet, deutlich höhere Viskositäten zulässt und gleichzeitig relativ große Zeitschritte erlaubt. Die betrachteten Schemata erfüllen diese Bedingungen nicht hinreichend. Mit dem EDAC-Schema ist die Rechenzeit der Simulation relativ kurz.

Der in Abschnitt 3.3.5 beschriebene Algorithmus zur Oberflächenerkennung identifiziert die meisten Partikel auch für die extrudierte Geometrie korrekt. Vereinzelt werden Partikel fälschlich der Oberfläche zugeordnet. Dies ist aber in Bereichen der Fall, in denen die Partikel aufgrund der Kompaktierung ungewöhnlich angeordnet sind. Möglicherweise führt auch hier eine Formulierung, in der die Inkompressibilität gewährleistet ist, zu Verbesserungen. Auch das thermische Modell kann den Temperaturverlauf der Geometrie zuverlässig abbilden. Sowohl der konduktive Wärmeaustausch mit der Platte als auch die Kühlung durch Konvektion und Wärmestrahlung funktionieren. Auch der Wärmeaustausch zwischen den Strängen kann abgebildet werden. Dies ist besonders für das erneute Aufschmelzen und Verschweißen relevant. Da bisher keine experimentellen Vergleichsdaten vorliegen, können nur wenige Aussagen, über die Genauigkeit des errechneten Temperaturverlaufs getroffen werden. Der Abkühlvorgang ist durch die konduktive Wärmeleitung dominiert, was allerdings durch Anpassung des Koeffizienten h_c geändert werden kann. Außerdem ist die extrudierte Struktur relativ klein. In größeren Strukturen, sind die oberen Schichten weiter von der Bauplattform entfernt, sodass der Wärmeaustausch durch die konduktive Wärmeleitung reduziert wird. Es ist anzunehmen, dass die konstante Vorschubgeschwindigkeit der Düse einen relativ großen Fehler im Temperaturverlauf zur Folge hat. Das Modell berücksichtigt nicht die Beschleunigung der Düse, die stattdessen stets mit konstanter Geschwindigkeit bewegt wird. Dadurch ist die berechnete Zeit gegenüber der Realität verkürzt. In einem Experiment sind deshalb bei gleicher Geometrie geringere Temperaturen zu erwarten. Die Vernachlässigung der Beschleunigung wirkt sich auch auf die entstehende Struktur aus, die durch die Trägheit des Fluids verformt wird. Die tatsächliche Vorschubgeschwindigkeit sollte deshalb in der Simulation berücksichtigt werden. Die extrudierte Geometrie besteht aus relativ wenigen und kurzen Strängen. Sie unterscheidet sich insofern von den üblicherweise im MEX-Prozess entstehenden Mesostrukturen. Möglicherweise werden bei der simulierten

Extrusion größerer Strukturen Ergebnisse erzielt, die den Mesostrukturen in der Praxis ähnlicher sind.

Das Modell enthält bereits eine Gleichung zur Berechnung der scherraten- und temperaturabhängigen Viskosität. Die errechneten Werte können aufgrund der Limitierungen der Schemata (siehe oben) nicht in die Berechnung einfließen. Wird das verwendete SPH-Schema jedoch durch eines ersetzt, dass die hohen Viskositäten abbilden kann, kann der Cross-WLF-Ansatz eingesetzt werden. Das Materialmodell kann außerdem um eine temperaturabhängige Dichte erweitert werden. Auch weitere Phänomene, die vom Temperaturverlauf abhängen, wie etwa Kristallisationsvorgänge, können mit einem erweiterten Materialmodell abgebildet werden.

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

In der vorliegenden Arbeit wird der MEX-Prozess auf Mesoebene mit der SPH-Methode abgebildet. Dazu werden zwei Simulationswerkzeuge, ABAQUS und PySPH eingesetzt, in denen jeweils ein Modell des Prozesses implementiert wird. Mit beiden Modellen können einzelne Polymerstränge und einfache mehrschichtige Strukturen extrudiert werden.

Mit der kommerziell verfügbaren Software ABAQUS werden mit geringem Aufwand bereits gute Ergebnisse erzielt. Um die Rechenzeit zu beschränken, müssen jedoch eine relativ hohe Kompressibilität der Polymerschmelze und eine Viskosität, die um mehrere Größenordnungen geringer als die reale ist, akzeptiert werden. In ABAQUS steht nur eine SPH-Formulierung zur Verfügung, mit der die reale Viskosität und Inkompressibilität nicht bei praktikablem Rechenaufwand abzubilden sind. Bei der Extrusion einer mehrschichtigen Struktur zerfließt der entstehende Körper deshalb relativ schnell. Des Weiteren treten relativ hohe Dichteänderungen auf. Hinzu kommt der eingeschränkte Funktionsumfang der Software. So muss bspw. zwingend die Bauplattform anstatt der Düse verfahren werden. Weiter sind keine zweidimensionalen Simulationen möglich, die beim Aufbau eines Modells schnelles Iterieren ermöglichen würden. Entscheidend ist allerdings das Fehlen des für die Modellierung der Temperaturverteilung unerlässlichen Freiheitsgrades *NT11*. Die entstehende Mesostruktur ist in hohem Maße von den thermischen Bedingungen abhängig, weshalb der Nutzen des Modells ohne Berücksichtigung der Temperatur erheblich eingeschränkt ist.

Mithilfe des Open Source Pakets PySPH kann die MEX mit drei unterschiedlichen SPH-Schemata modelliert werden, dem WCSPH-, AHA- und EDAC-Schema. Von den dreien wird das EDAC-Schema als am besten geeignet identifiziert. Auch in diesem Modell ist das Fluid stark kompressibel und es können nur geringere Viskositäten eingesetzt werden als im ABAQUS-Modell. Die extrudierte Struktur zerfließt deshalb noch stärker und auch die Dichteunterschiede sind nochmals höher. Dennoch ist PySPH für die Modellierung des MEX-Prozesses besser geeignet, da es durch den Open Source Ansatz beliebig erweiterbar ist. So ist in der Stan-

dardversion der Software ebenfalls keine Möglichkeit vorhanden, Temperaturen zu betrachten. In der vorliegenden Arbeit wird das Paket um diese Möglichkeit erweitert. Das thermische Modell berücksichtigt, konduktive Wärmeleitung sowie die Wärmeabgabe an die Umgebung durch Konvektion und Strahlung. Voraussetzung dafür ist ein ebenfalls neu implementierter Algorithmus zur Verfolgung der freien Oberfläche. Auf Basis der Temperaturverteilung wird die Viskosität des Fluids nach dem Cross-WLF-Ansatz berechnet. Die berechneten Viskositäten können aus dem oben genannten Grund jedoch noch nicht in die Berechnungen mit einfließen. Des Weiteren ist der Partikeleinlass, durch den der Austritt des Fluids aus der Düse modelliert wird, modifiziert, sodass ein Verfahren in alle Raumrichtungen möglich ist. Diese Änderungen demonstrieren die Anpassbarkeit von PySPH, das auch grundlegende Änderungen an den vorhandenen SPH-Formulierungen oder die Implementierung weiterer Schemata zulässt. Aufgrund der Anpassbarkeit und der Möglichkeit, den Code vollständig einzusehen, ist PySPH von den beiden betrachteten Simulationswerkzeugen besser für die weitere Arbeit an der Modellierung des MEX-Prozesses geeignet.

6.2 Ausblick

Um die Modellierung des Prozesses mit der SPH-Methode zu verbessern, ist es notwendig, ein SPH-Schema zu finden, mit dem bei moderater Rechenzeit die Inkompressibilität des Fluids gewährleistet ist und deutlich höhere Viskositäten eingesetzt werden können. Vielversprechend sind implizite Lösungsverfahren, bei denen die Inkompressibilität des Fluids nicht über eine Zustandsgleichung mit künstlicher Schallgeschwindigkeit realisiert wird. So könnten längere Zeitschritte gewählt und der Rechenaufwand reduziert werden. Auch Bertevas et al. [35] weisen auf die Problematik der kleinen Zeitschritte hin und schlagen ein implizites Verfahren als möglichen Lösungsansatz vor. Ist ein solches Verfahren gefunden, können Untersuchungen mit höheren Auflösungen durchgeführt werden und die numerischen Parameter optimiert werden. Außerdem können die Materialkennwerte temperaturabhängig modelliert werden. Im Falle der Viskosität wäre dies durch sehr geringen Mehraufwand möglich, da das Materialgesetz bereits implementiert ist. Der Cross-WLF-Ansatz muss jedoch ergänzt werden, damit auch Viskositäten für Temperaturen unterhalb dessen Gültigkeitsbereich korrekt berechnet werden können. Das Materialmodell sollte den Phasenübergang von der Schmelze in einen glasartigen Zustand korrekt abbilden können. Auch weitere Effekte wie thermische Volumenänderung können berücksichtigt werden. Grundlage für temperaturabhängige Materialmodellierung ist das thermische Modell, dessen Fehler in Zukunft ebenfalls reduziert werden können. Dies kann durch eine Verbesserung der Zuordnung freier Oberfläche zu λ-Werten erfolgen. Hierfür können Referenzfälle bekannter Geometrie betrachtet und ausgewertet werden. Der in dieser

Arbeit verwendete lineare Zusammenhang kann dann durch ein Polynom höherer Ordnung ersetzt werden. Sowohl der Temperaturverlauf als auch die entstehende Geometrie werden zudem durch die als konstant angenommene Vorschubgeschwindigkeit der Düse beeinflusst. Um diesen Fehler zu reduzieren, muss die Fähigkeit der Maschine zu beschleunigen, berücksichtigt werden. Als Grundlage kann der mit einem Slicer erstellte *G-Code* dienen, der den Pfad sowie das Geschwindigkeitsprofil enthält. Sowohl die extrudierte Geometrie als auch die Temperaturverteilung sollten mit Experimenten verglichen werden. So kann auch der konvektive Wärmeübergangskoeffizient bestimmt werden. Darüber hinaus könnten in Zukunft auch Faserverstärkte Kunststoffe als MEX-Werkstoffe untersucht werden. So kann die im Prozess festgelegte Faserorientierung berechnet werden. Auf der Mesoebene können diskrete Fasern nicht aufgelöst werden. Mikromodelle können dies leisten. Von Meyer et al. [38] wird die SPH-Methode für die Kunststoffmatrix in Verbindung mit einem *bead-chain* Model für diskrete Fasern in einem solchen Mikromodell eingesetzt, allerdings nicht im Kontext der MEX. Die Ergebnisse eines solchen Mikromodells können auf Mesoebene in Form eines Faserorientierungstensors berücksichtigt werden.

Ein verbessertes Modell des MEX-Prozesses kann zur Durchführung von Parameterstudien und der Vorhersage der Mesostruktur additiv gefertigter Bauteile eingesetzt werden. Da sich die SPH-Methode grundsätzlich auch für die Struktursimulation von Festkörpern eignet, ist eine rein simulative Charakterisierung von additiv gefertigten Strukturen denkbar, womit wiederum Simulationen auf Bauteilebene informiert werden könnten.

Anhang A.

Quellcode

A.1 PySPH Programm zur Modellierung der MEX

A.1.1 examples/fff.py

```
""" Fused Filament Fabrication """

import os
from pathlib import Path
import json
from math import sqrt
import numpy as np
from termcolor import colored
from matplotlib import pyplot as plt

from pysph.base.utils import get_particle_array
from pysph.fff.basic_flow_variables import ShearRate
from pysph.fff.thermal_aha import ThermalAHAScheme
from pysph.fff.thermal_edac import TEDACStep, ThermalEDACScheme
from pysph.fff.thermal_iisph import ThermalIISPHScheme,
    ↪ ThermalIISPHStep
from pysph.fff.thermal_tvf import ThermalTransportVelocityStep
from pysph.fff.viscosity_models import Cross_WLF_Viscosity
from pysph.solver.application import Application
from pysph.solver.utils import load
from pysph.solver.vtk_output import main as vtk_output_main
from pysph.sph.basic_equations import VelocityGradient3D
from pysph.sph.bc.characteristic.simple_inlet_outlet import
    ↪ SimpleInletOutlet
from pysph.sph.integrator_step import InletOutletStep
from pysph.sph.scheme import AdamiHuAdamsScheme, TVFScheme,
    ↪ WCSPHScheme
```

```
25 from pysph.sph.equation import Group
26 from pysph.base.kernels import CubicSpline
27 from pysph.fff.thermal import HeatEquation, SurfaceDetection
28 from pysph.sph.wc.edac import EDACScheme
29 from pysph.fff.fff_inlet import FFFInletInfo, FFFInlet,
       ↪ FFFMoveNozzle, InletOutletStepOneStage
30 from pysph.fff.thermal_wcsph import TWCSPHStep
31 from pysph.sph.wc.gtvf import GTVFScheme, GTVFStep
32
33 # Data about the run
34 DATA = {
35     'Outcome' : 'still running'
36 }
37
38 # choose unit system
39 UNITS = {
40     'SI' :      1.0,
41     'SImm' :    1.0e3,
42     }
43 L_CONVERSION = UNITS['SImm']
44
45 # constants
46 CON = {
47     'T_zero' :  -273.15,
48     'sigma'  :   5.67e-8 /L_CONVERSION,
49     'g'      :  -9.81 *L_CONVERSION,
50     'R'      :   8.31 *L_CONVERSION
51 }
52
53 # parameters
54 PARAM = {
55     # temperatures
56     'T_nozzle'          : 220.0,
57     'T_plate'           : 55.0,
58     # environment
59     'T'                 : 20.0,
60     'p0'                : 0.0/L_CONVERSION**2,
61     'pb'                : 0.0/L_CONVERSION**2,
62     'pref'              : 0.0/L_CONVERSION**2,
63     # geometry
64     'r_nozzle'          : 0.2e-3*L_CONVERSION,
65     # movement
66     'y_steps'           : 2,
67     'z_steps'           : 2,
68     'nozzlespeed'       : 60.0e-3*L_CONVERSION,
69     # numeric parameters
70     'scheme'            : 'EDAC',
71     'hdx'               : 1.5,
```

```
72      'dim'                : 3,
73      'timestep_factor'    : 1.0,
74      'adaptive_dt'        : True,
75      'alpha'              : 0.0,
76      'beta'               : 0.0,
77      'gamma'              : 7.0,
78      'delta'              : 0.1,
79      'r_nozzle/dx'        : 4,
80      'tdamp'              : 0.0,
81      # IISPH
82      'omega'              : 0.5,
83      'tolerance'          : 0.01,
84      # booleans
85      'tensile_corr'       : True,
86      # output
87      'no_of_outputs'      : 100,
88  }
89
90  # parameters that need to be calculated from others
91  # numeric
92  PARAM['kernel'] = CubicSpline(dim=PARAM['dim'])
93  PARAM['dx'] = PARAM['r_nozzle']/PARAM['r_nozzle/dx']
94  PARAM['h'] = PARAM['hdx']*PARAM['dx']
95  if PARAM['dim'] == 2:
96      PARAM['shape'] = 'square'
97      PARAM['z_steps'] = 0
98  else:
99      PARAM['shape'] = 'cube'
100 # movement
101 PARAM['x_length'] = 3.0e-3*L_CONVERSION - 2*PARAM['r_nozzle']
102 PARAM['h_nozzle'] = 2.0*PARAM['r_nozzle']
103 PARAM['extrusionspeed'] = PARAM['nozzlespeed']
104 PARAM['z_diff'] = 3*PARAM['r_nozzle']
105 PARAM['y_diff'] = 2*PARAM['r_nozzle']
106 PARAM['t_wait'] = 0.6*PARAM['h_nozzle']/PARAM['extrusionspeed']
107 PARAM['tf'] = PARAM['t_wait'] + \
108                 ((PARAM['z_steps']+1)*(PARAM['y_steps']+1) * (PARAM['
                    ↪ x_length']) + \
109                 (PARAM['y_steps']+1) * PARAM['z_steps']*PARAM['z_diff
                    ↪ '] + \
110                 PARAM['y_steps'] * PARAM['y_diff'] \
111                 )/PARAM['nozzlespeed']
112
113 # material definitions
114 PLA = {
115     'name':      'PLA',
116     'rho0':      1072.7/L_CONVERSION**4,
117     'c0':        100*PARAM['nozzlespeed'], # Value *L_CONVERSION,
```

```
118     'eta':      0.1/L_CONVERSION**2,
119     'n':        0.25,
120     'tau*':     100_861/L_CONVERSION**2,
121     'D1':       3.31719e9/L_CONVERSION**2,
122     'D2':       373.15 + CON['T_zero'],
123     'D3':       0.0 *L_CONVERSION**2,
124     'A1':       20.195,
125     'A2':       51.6,
126     'kappa':    0.195,
127     'cp':       2060.0*L_CONVERSION**2,
128     'epsilon':  0.98,
129     'hc':       100.0/L_CONVERSION,
130 }
131 ALUMINIUM = {
132     'name':     'Aluminium',
133     'rho0':     2700.0/L_CONVERSION**4,
134     'c0':       PLA['c0'],
135     'eta':      PLA['eta'],
136     'n':        PLA['n'],
137     'tau*':     PLA['tau*'],
138     'D1':       PLA['D1'],
139     'D2':       PLA['D2'],
140     'D3':       PLA['D3'],
141     'A1':       PLA['A1'],
142     'A2':       PLA['A2'],
143     'kappa':    204.0,
144     'cp':       940.0,
145     'epsilon':  0.4,
146     'hc':       100.0/L_CONVERSION,
147 }
148 IRON = {
149     'name':     'Iron',
150     'rho0':     7870.0/L_CONVERSION**4,
151     'c0':       PLA['c0'],
152     'eta':      PLA['eta'],
153     'n':        PLA['n'],
154     'tau*':     PLA['tau*'],
155     'D1':       PLA['D1'],
156     'D2':       PLA['D2'],
157     'D3':       PLA['D3'],
158     'A1':       PLA['A1'],
159     'A2':       PLA['A2'],
160     'kappa':    81.0,
161     'cp':       470.0,
162     'epsilon':  0.4,
163     'hc':       100.0/L_CONVERSION,
164 }
165 STEEL_ALLOY = {
```

```
166     'name':      'Steel_Alloy',
167     'rho0':      7900.0/L_CONVERSION**4,
168     'c0':        PLA['c0'],
169     'eta':       PLA['eta'],
170     'n':         PLA['n'],
171     'tau*':      PLA['tau*'],
172     'D1':        PLA['D1'],
173     'D2':        PLA['D2'],
174     'D3':        PLA['D3'],
175     'A1':        PLA['A1'],
176     'A2':        PLA['A2'],
177     'kappa':     14.0,
178     'cp':        510.0,
179     'epsilon':   0.4,
180     'hc':        100.0/L_CONVERSION,
181 }
182
183 # material assignment
184 MAT = {
185     'nozzle':    PLA,
186     'plate':     STEEL_ALLOY,
187     'fluid':     PLA,
188 }
189
190 INLETOUTLETS = ['nozzle']
191 FLUIDS = ['fluid']
192 SOLIDS = ['plate']
193 FLUIDS_WIO = FLUIDS + INLETOUTLETS
194 SOLIDS_WIO = SOLIDS + INLETOUTLETS
195 ALL = FLUIDS + SOLIDS + INLETOUTLETS
196
197 class FFF(Application):
198     """ Fused Filament Fabrication """
199     def initialize(self):
200         self.output_arrays = ['x', 'y', 'z', 'u', 'v', 'w', 'rho',
                ↪ 'm', 'p', 'V', 'T', 'aT',
201                               'Lambda', 'nu', 'eta', 'shear_rate']
202
203         self.timestep_estimates, self.min_estimated_timestep = self
                ↪ .estimate_timestep()
204
205     def create_particles(self):
206         # set necessary properties
207         general_props = ['V']
208         io_props = ['ioid', 'disp']
209         inlet_mod_props = ['ref_x', 'ref_y', 'ref_z']
210         temperature_props = ['T', 'T0', 'aT', 'kappa', 'cp', '
                ↪ Lambda', 'nu', 'eta',
```

```
211                     'Ainv00', 'Ainv01', 'Ainv02', 'Ainv11', 'Ainv12
                            ↪ ',
212                     'Ainv22']
213         shear_rate_props = ['v00', 'v10', 'v01', 'v11', 'v02', 'v20
               ↪ ', 'v12', 'v21', 'v22',
214                             'shear_rate']
215         # additional properties and constants for schemes
216         scheme_props = {
217             'WCSPH' : ['cs', 'ax', 'ay', 'az', 'arho', 'x0', 'y0',
                   ↪ 'z0',
218                        'u0', 'v0', 'w0', 'rho0', 'div', 'dt_cfl', '
                              ↪ dt_force'],
219             'EDAC'  : ['cs', 'ay', 'ax', 'az', 'wij', 'p0', 'ap', '
                   ↪ vf', 'uf', 'vg', 'ug', 'wf',
220                        'wg', 'u0', 'v0', 'w0', 'x0', 'y0', 'z0'],
221             'AHA'   : ['cs', 'arho', 'rho0', 'u0', 'v0', 'w0', 'x0'
                   ↪ , 'y0', 'z0', 'ax', 'ay', 'az',
222                        'wg', 'uf', 'vg', 'wij', 'ug', 'vf', 'wf', '
                              ↪ auhat', 'avhat', 'awhat'],
223             'TVF'   : ['cs', 'uf', 'wf', 'vg', 'wg', 'ug', 'wij', '
                   ↪ vf', 'awhat', 'avhat', 'auhat',
224                        'uhat', 'what', 'vhat', 'vmag2'],
225             'GTVF'  : ['cs', 'wij', 'vg', 'vf', 'wg', 'ug', 'uf', '
                   ↪ wf', 'vhat', 'uhat', 'arho',
226                        'rho0', 'what', 'auhat', 'awhat', 'p0', '
                              ↪ avhat', 'rhodiv', 'asigma',
227                        'sigma'],
228             'IISPH' : ['cs', 'uadv', 'vadv', 'wadv', 'rho_adv', 'au
                   ↪ ', 'av', 'aw', 'ax', 'ay', 'az',
229                        'dii0', 'dii1', 'dii2', 'dt_cfl', 'dt_force'
                              ↪ , 'aii', 'dijpj0', 'dijpj1',
230                        'dijpj2', 'p', 'p0', 'piter', 'compression'
                              ↪ ],
231         }
232         scheme_cons = {
233             'WCSPH' : {},
234             'EDAC'  : {},
235             'AHA'   : {},
236             'TVF'   : {},
237             'GTVF'  : {},
238             'IISPH' : {'tmp_comp': [0.0, 0.0]}
239         }
240
241         # giving the thermal versions of schemes the same
               ↪ properties
242         thermal_schemes = {
243             'TEDAC' : 'EDAC',
244             'TAHA'  : 'AHA',
```

```
245             'TIISPH': 'IISPH',
246         }
247         for thermal_scheme, scheme in thermal_schemes.items():
248             scheme_props[thermal_scheme] = scheme_props[scheme]
249             scheme_cons[thermal_scheme] = scheme_cons[scheme]
250 
251         additional_props = general_props + io_props +
                ↪ inlet_mod_props + temperature_props + \
252                         shear_rate_props + scheme_props[PARAM['
                            ↪ scheme']]
253 
254         constants = scheme_cons[PARAM['scheme']]
255 
256         # FLUID
257         # the fluid domain is initially empty
258         fluid = get_particle_array(additional_props =
                ↪ additional_props, constants=constants,
259                                    name='fluid')
260 
261         # NOZZLE
262         particles_outside_nozzle = []
263         # center of the nozzle should sit on x=0 and z=0 to match
                ↪ the refpoint
264         x, y    = np.mgrid[-2*PARAM['r_nozzle']+PARAM['dx']/2 :
265                            PARAM['r_nozzle'] :
266                            PARAM['dx'],
267 
268                            PARAM['h_nozzle']+PARAM['dx']/2 :
269                            PARAM['h_nozzle']+3*PARAM['dx']:
270                            PARAM['dx']]
271         z = 0
272         if PARAM['dim']==3:
273             x, y, z = np.mgrid[ -2*PARAM['r_nozzle']+PARAM['dx']/2
                    ↪ :
274                                 PARAM['r_nozzle'] :
275                                 PARAM['dx'],
276 
277                                 PARAM['h_nozzle']+PARAM['dx']/2 :
278                                 PARAM['h_nozzle']+3*PARAM['dx']:
279                                 PARAM['dx'],
280 
281                                 -2*PARAM['r_nozzle']+PARAM['dx']/2
                                    ↪ :
282                                 PARAM['r_nozzle'] :
283                                 PARAM['dx']]
284             # creating 1D arrays
285             x = x.ravel()
286             z = z.ravel()
```

```
287             # find particles outside nozzle
288             for idx, x_value in enumerate(x):
289                 if np.sqrt(x_value**2 + z[idx]**2) - PARAM['
                        ↪ r_nozzle'] > PARAM['dx']/1000:
290                     particles_outside_nozzle.append(idx)
291
292         v       =-np.ones_like(x)*PARAM['extrusionspeed']
293         h       = np.ones_like(x)*PARAM['dx']*PARAM['hdx']
294         m       = np.ones_like(x)*PARAM['dx']**PARAM['dim']*MAT['
                ↪ nozzle']['rho0']
295         rho     = np.ones_like(x)*MAT['nozzle']['rho0']
296         T       = np.ones_like(x)*PARAM['T_nozzle']
297         kappa   = np.ones_like(x)*MAT['nozzle']['kappa']
298         cp      = np.ones_like(x)*MAT['nozzle']['cp']
299         V       = np.ones_like(x)/(PARAM['dx']**PARAM['dim']) #
                ↪ pysph uses the inverse volume
300         nozzle = get_particle_array(additional_props =
                ↪ additional_props, constants=constants,
301         name='nozzle', x=x, y=y, z=z, v=v, h=h, m=m, rho=rho, T=T,
                ↪ kappa=kappa, cp=cp, V=V)
302
303         # delete outside particles
304         nozzle.remove_particles(particles_outside_nozzle)
305
306         # PLATE
307         x, y = np.mgrid[-5.0*PARAM['r_nozzle'] :
308                         PARAM['x_length']+5.0*PARAM['r_nozzle'] :
309                         PARAM['dx'],
310
311                         -2.5*PARAM['dx'] :
312                         0 :
313                         PARAM['dx']]
314         z = 0
315         if PARAM['dim'] == 3:
316             x, y, z = np.mgrid[ -5.0*PARAM['r_nozzle'] :
317                                 PARAM['x_length']+5.0*PARAM['
                                        ↪ r_nozzle'] :
318                                 PARAM['dx'],
319
320                                -2.5*PARAM['dx']     :
321                                 0 :
322                                 PARAM['dx'],
323
324                                 -3.0*PARAM['r_nozzle'] :
325                                 PARAM['z_steps']*PARAM['z_diff'
                                        ↪ ]+4.0*PARAM['r_nozzle'] :
326                                 PARAM['dx']]
327         h       = np.ones_like(x)*PARAM['dx']*PARAM['hdx']
```

```
328         m       = np.ones_like(x)*PARAM['dx']**PARAM['dim']*MAT['
               ↪ plate']['rho0']
329         rho     = np.ones_like(x)*MAT['plate']['rho0']
330         T       = np.ones_like(x)*PARAM['T_plate']
331         kappa   = np.ones_like(x)*MAT['plate']['kappa']
332         cp      = np.ones_like(x)*MAT['plate']['cp']
333         V       = np.ones_like(x)/(PARAM['dx']**PARAM['dim']) #
               ↪ pysph uses the inverse volume
334 
335         plate = get_particle_array(additional_props =
               ↪ additional_props, constants=constants,
336         name='plate', x=x, y=y, z=z, h=h, m=m, rho=rho, T=T, kappa=
               ↪ kappa, cp=cp, V=V)
337 
338         particles = [nozzle, fluid, plate]
339 
340         for array in particles:
341             array.set_output_arrays(self.output_arrays)
342 
343         return particles
344 
345     def create_scheme(self):
346         if PARAM['scheme'] == 'EDAC':
347             scheme = EDACScheme(FLUIDS, SOLIDS, dim=PARAM['dim'],
                   ↪ rho0=MAT['fluid']['rho0'],
348                 c0=MAT['fluid']['c0'], nu=MAT['fluid']['eta']/MAT['
                       ↪ fluid']['rho0'], gy=CON['g'],
349                 h=PARAM['hdx']*PARAM['dx'], edac_alpha=0.5, alpha=
                       ↪ PARAM['alpha'], bql=False,
350                 clamp_p=True, inlet_outlet_manager=self.iom,
                       ↪ inviscid_solids=None)
351 
352             extra_steppers = {'nozzle': InletOutletStep(), 'plate':
                   ↪  TEDACStep(),
353                               'fluid': TEDACStep()}
354 
355         elif PARAM['scheme'] == 'TEDAC':
356             scheme = ThermalEDACScheme(FLUIDS, SOLIDS, dim=PARAM['
                   ↪ dim'], rho0=MAT['fluid']['rho0'],
357                 c0=MAT['fluid']['c0'], nu=MAT['fluid']['eta']/MAT['
                       ↪ fluid']['rho0'], gy=CON['g'],
358                 h=PARAM['hdx']*PARAM['dx'], edac_alpha=0.5, alpha=
                       ↪ PARAM['alpha'], bql=False,
359                 clamp_p=True, inlet_outlet_manager=self.iom,
                       ↪ inviscid_solids=None)
360 
361             extra_steppers = {'nozzle': InletOutletStep(), 'plate':
                   ↪  TEDACStep(),
```

```
362                                 'fluid': TEDACStep()}
363
364         elif PARAM['scheme'] == 'AHA':
365             scheme = AdamiHuAdamsScheme(fluids=FLUIDS, solids=
                    ↪ SOLIDS, dim=PARAM['dim'],
366                 rho0=MAT['fluid']['rho0'], c0=MAT['fluid']['c0'],
367                 nu=MAT['fluid']['eta']/MAT['fluid']['rho0'],
368                 h0 = PARAM['hdx']*PARAM['dx'], gy=CON['g'], p0=
                        ↪ PARAM['p0'], gamma=PARAM['gamma'],
369                 tdamp=PARAM['tdamp'], alpha=PARAM['alpha']
370             )
371             extra_steppers = {'nozzle': InletOutletStep(), 'plate':
                    ↪  TWCSPHStep(),
372                                 'fluid': TWCSPHStep()}
373
374         elif PARAM['scheme'] == 'TAHA':
375             scheme = ThermalAHAScheme(fluids=FLUIDS, solids=SOLIDS,
                    ↪  dim=PARAM['dim'],
376                 rho0=MAT['fluid']['rho0'], c0=MAT['fluid']['c0'],
377                 nu=MAT['fluid']['eta']/MAT['fluid']['rho0'],
378                 h0 = PARAM['hdx']*PARAM['dx'], gy=CON['g'], p0=
                        ↪ PARAM['p0'], gamma=PARAM['gamma'],
379                 tdamp=PARAM['tdamp'], alpha=PARAM['alpha']
380             )
381             extra_steppers = {'nozzle': InletOutletStep(), 'plate':
                    ↪  TWCSPHStep(),
382                                 'fluid': TWCSPHStep()}
383
384         elif PARAM['scheme'] == 'WCSPH':
385             scheme = WCSPHScheme(fluids=FLUIDS, solids=SOLIDS, dim=
                    ↪ PARAM['dim'],
386                 rho0=MAT['fluid']['rho0'], c0=MAT['fluid']['c0'],
                        ↪ h0=PARAM['hdx']*PARAM['dx'],
387                 hdx=PARAM['hdx'], gamma=PARAM['gamma'], gy=CON['g'
                        ↪ ], delta=PARAM['delta'],
388                 nu=MAT['fluid']['eta']/MAT['fluid']['rho0'],
389                 tensile_correction=PARAM['tensile_corr']
390                 )
391
392             extra_steppers = {'nozzle': InletOutletStep(), 'plate':
                    ↪  TWCSPHStep(),
393                                 'fluid': TWCSPHStep()}
394
395         elif PARAM['scheme'] == 'TVF':
396             scheme = TVFScheme(fluids=FLUIDS, solids=SOLIDS, dim=
                    ↪ PARAM['dim'],
397             rho0=MAT['fluid']['rho0'], c0=MAT['fluid']['c0'],
```

```
398             nu=MAT['fluid']['eta']/MAT['fluid']['rho0'], p0=PARAM['
                    ↪ p0'],
399             pb=PARAM['pb'], h0=PARAM['hdx']*PARAM['dx'], gy=CON['g'
                    ↪ ],
400             alpha=PARAM['alpha'], tdamp=PARAM['tdamp']
401             )
402
403             extra_steppers = {'nozzle': InletOutletStep(), 'plate':
                    ↪  ThermalTransportVelocityStep(),
404                               'fluid': ThermalTransportVelocityStep
                                      ↪ ()}
405
406         elif PARAM['scheme'] == 'GTVF':
407             scheme = GTVFScheme(fluids=FLUIDS, solids=SOLIDS, dim=
                    ↪ PARAM['dim'],
408             rho0=MAT['fluid']['rho0'], c0=MAT['fluid']['c0'],
409             nu=MAT['fluid']['eta']/MAT['fluid']['rho0'], h0=PARAM['
                    ↪ hdx']*PARAM['dx'],
410             pref=PARAM['pref'], gy=CON['g'], b=1, alpha=PARAM['
                    ↪ alpha']
411             )
412
413             extra_steppers = {'nozzle': InletOutletStep(), 'plate':
                    ↪  GTVFStep(),
414                               'fluid': GTVFStep()}
415
416         elif PARAM['scheme'] == 'TIISPH':
417             scheme = ThermalIISPHScheme(fluids=FLUIDS, solids=
                    ↪ SOLIDS_WIO, dim=PARAM['dim'],
418             rho0=MAT['fluid']['rho0'], nu=MAT['fluid']['eta']/MAT['
                    ↪ fluid']['rho0'], gy=CON['g'],
419             omega=PARAM['omega'], tolerance=PARAM['tolerance'])
420
421             extra_steppers = {'nozzle': InletOutletStepOneStage(),
                    ↪ 'plate': ThermalIISPHStep(),
422                               'fluid': ThermalIISPHStep()}
423
424         PARAM['dt'] = PARAM['timestep_factor']*self.
                ↪ min_estimated_timestep
425
426         # only use the adaptive timestep after the waittime for
                ↪ schemes that support it
427         if PARAM['scheme'] == 'TIISPH':
428             kwargs = dict(tf=PARAM['tf'], kernel=PARAM['kernel'],
                    ↪ dt=PARAM['dt'],
429                           adaptive_timestep=PARAM['adaptive_dt'],
430                           adaptive_timestep_critical_time=PARAM['t_wait'
                                ↪ ])
```

```
431         else:
432             kwargs = dict(tf=PARAM['tf'], kernel=PARAM['kernel'],
                    ↪ dt=PARAM['dt'],
433             adaptive_timestep=PARAM['adaptive_dt'])
434 
435         # create list with times for outputs
436         outputs_at = np.arange(PARAM['tf']/PARAM['no_of_outputs'],
                ↪ PARAM['tf'],
437                                PARAM['tf']/PARAM['no_of_outputs']).
                                   ↪ tolist()
438 
439         # pass everything to the solver (maybe this is uneccessary
                ↪ ?)
440         scheme.configure_solver(output_at_times=outputs_at, pfreq=1
                ↪ e12,
441                                 extra_steppers=extra_steppers, **
                                    ↪ kwargs)
442 
443         # create the inlet outlet manager
444         # the IISPH scheme only has one step integration
445         self.iom = self._create_inlet_outlet_manager()
446         if PARAM['scheme'] == 'TIISPH':
447             self.iom.active_stages = [1]
448         else:
449             self.iom.active_stages = [2]
450         self.iom.setup_iom(dim=PARAM['dim'], kernel=scheme.solver.
                ↪ kernel)
451         self.iom.update_dx(dx=PARAM['dx'])
452 
453         return scheme
454 
455     def create_equations(self):
456         equations = self.scheme.get_equations()
457         g_0, g_1, g_2 = ([] for i in range(3))
458         g_0.append(
459             FFFMoveNozzle(dest='nozzle', sources=None, t_wait=PARAM
                    ↪ ['t_wait'],
460             x_length=PARAM['x_length'], y_diff=PARAM['y_diff'],
                    ↪ z_diff=PARAM['z_diff'],
461             y_steps=PARAM['y_steps'], z_steps=PARAM['z_steps'],
                    ↪ h_start=PARAM['h_nozzle'],
462             nozzlespeed=PARAM['nozzlespeed'], extrusionspeed=PARAM[
                    ↪ 'extrusionspeed'])
463         )
464         for fluid in FLUIDS:
465             g_0.extend([
466                 SurfaceDetection(dest=fluid, sources=ALL),
467                 VelocityGradient3D(dest=fluid, sources=ALL)
```

```
468             ])
469             g_1.extend([
470                 HeatEquation(dest=fluid, sources=ALL, epsilon=MAT[
                        ↪ fluid]['epsilon'],
471                 sigma=CON['sigma'], hc=MAT[fluid]['hc'], T_env=
                        ↪ PARAM['T'], T_zero=CON['T_zero'],
472                 shape=PARAM['shape']
473                 ),
474                 ShearRate(dest=fluid, sources=None),
475             ])
476 
477         for destination in ALL:
478             g_2.extend([
479                 Cross_WLF_Viscosity(dest=destination, sources=None,
                        ↪  rho0=MAT['fluid']['rho0'],
480                 n=MAT['fluid']['n'], tau_star=MAT['fluid']['tau*'],
                        ↪  D1=MAT['fluid']['D1'],
481                 D2=MAT['fluid']['D2'], D3=MAT['fluid']['D3'], A1=
                        ↪ MAT['fluid']['A1'],
482                 A2=MAT['fluid']['A2'], use_limits=False, nu_min
                        ↪ =0.0, nu_max=40.0,
483                 corrector=1.0 # 5.0e-5
484                 ),
485             ])
486 
487         if PARAM['scheme'] == 'GTVF':
488             print('FFF specific equations (surface detection,
                    ↪ temperature and nozzle movement)'\
489                   ' are not implemented!')
490         else:
491             equations.insert(0, Group(g_0))
492             equations.insert(1, Group(g_1))
493             equations.insert(2, Group(g_2))
494 
495         return equations
496 
497     def _create_inlet_outlet_manager(self):
498         inlet_info = FFFInletInfo(
499             pa_name='nozzle', normal=[0.0,1.0,0.0],
500             refpoint=[0.0,PARAM['h_nozzle'],0.0],
501             has_ghost=False, update_cls=FFFInlet,
502             flowspeed=PARAM['extrusionspeed']
503         )
504 
505         iom = SimpleInletOutlet(
506             fluid_arrays=FLUIDS, inletinfo=[inlet_info],
507             outletinfo=[]
508         )
```

```
509
510         return iom
511
512     def create_inlet_outlet(self, particle_arrays):
513         iom = self.iom
514         return iom.get_inlet_outlet(particle_arrays)
515
516     def estimate_timestep(self):
517         """estimating the timestep according to the criteria from
                ↪ Meyer et al. "Parameter
518         Identification of Fiber Orientation Models Based on Direct
                ↪ Fiber Simulation with Smoothed
519         Particle Hydrodynamics" (2020)
520         (eta from the paper must be replaced with nu)
521         """
522         timestep_estimates = {
523         'compressibility'    : 0.4*PARAM['dx']/(1.1*MAT['fluid']['c0
                ↪ ']),
524         'viscosity'          : 0.125*PARAM['dx']**2*MAT['fluid']['
                ↪ rho0']/MAT['fluid']['eta'],
525         'gravity'            : 0.25*sqrt(PARAM['dx']/-CON['g']),
526         }
527
528         min_key = min(timestep_estimates, key=timestep_estimates.
                ↪ get)
529         min_dt = timestep_estimates[min_key]
530         print('The estimated timesteps are:')
531         for key, item in timestep_estimates.items():
532             print(f'{key}: {item:.2e}')
533         print('The timestep is determined by', colored(f'{min_key}'
                ↪ , 'yellow'),
534               f'and estimated to be {min_dt:.2e}.')
535
536         return timestep_estimates, min_dt
537
538     def clear_directory(self):
539         """ Deletes old simulation results still in the folder
                ↪ before the run.
540         """
541         for file in os.listdir(self.output_dir):
542             filepath = os.path.join(self.output_dir, file)
543             if os.path.isfile(filepath):
544                 os.remove(os.path.join(self.output_dir, file))
545
546     def post_process(self, info_fname_or_directory):
547         # extract values for pp
548         self.extract_values()
549         # self.print_last_field()
```

```
550         self.plot_last_frame()
551         # convert hdf files to vtk
552         vtk_output_main(argv=['fff_output'])
553 
554     def print_last_field(self):
555         """prints a field to the console after the last step
556         """
557         last_output = self.output_files[-1]
558         data = load(last_output)
559         pa = data['arrays']['plate']
560         print(pa.nu)
561 
562     def write_parameters(self):
563         """writes the PARAM dictionary to a json in the output
                ↪ folder
564         """
565         # converting object data to strings where necessary
566         param = PARAM
567         param['kernel'] = type(PARAM['kernel']).__name__
568 
569         # filenames and data
570         file_names = ['parameters', 'fluid_material','
                ↪ plate_material', 'constants',
571                       'timestep_estimates', 'outputs', 'data']
572         dictionaries = [param, MAT['fluid'], MAT['plate'], CON,
                ↪ self.timestep_estimates,
573                         self.output_arrays, DATA]
574 
575         # create directory
576         folder = "10_dicts"
577         directory = os.path.join(self.output_dir, folder)
578         if not os.path.exists(directory):
579             os.makedirs(directory)
580 
581         # writing the parameters as JSON files
582         for (file_name, dictionary) in zip(file_names, dictionaries
                ↪ ):
583             file_path = os.path.join(directory, f"{file_name}.json"
                    ↪ )
584             with open(file_path, 'w', encoding='utf-8') as txt:
585                 json.dump(dictionary, fp=txt, indent=2)
586 
587     def plot_last_frame(self):
588         """plots and saves the last timestep
589         """
590         # create directory
591         folder = "fff_Ergebnisse/10_plots"
```

```
592         directory = os.path.join(Path(self.output_dir).parent.
                ↪ absolute(), folder)
593         if not os.path.exists(directory):
594             os.makedirs(directory)
595 
596         # load output
597         last_output = self.output_files[-1]
598         data = load(last_output)
599         fluid_array = data['arrays']['fluid']
600         plate_array = data['arrays']['plate']
601         nozzle_array = data['arrays']['nozzle']
602         tf = data['solver_data']['t']
603 
604         # create plot
605         for array in [fluid_array, plate_array, nozzle_array]:
606             plt.scatter(array.x, array.y, marker='.',
607                         c=array.rho, cmap='Reds')
608         plt.gca().set_aspect('equal', adjustable='box')
609         plt.ylim(-0.2, PARAM['y_diff']*PARAM['y_steps']+1.5)
610         plt.xlim(-1.0, PARAM['x_length']+1.5)
611         plt.title(f"Particles at {round(tf, 5)} s")
612         plt.xlabel('x')
613         plt.ylabel('y')
614         rho0 = MAT['fluid']['rho0']
615         plt.colorbar(ticks=[0.95*rho0, rho0, 1.05*rho0])
616 
617         # save plot
618         fig = os.path.join(directory, (
619                         f"fff_{PARAM['scheme']}_eta_{MAT['fluid']['
                            ↪ eta']:.1e}"\
620                         f"_c0_{MAT['fluid']['c0']}_dx_{PARAM['dx
                            ↪ ']:.4f}"\
621                         f"_dim_{PARAM['dim']}.png"))
622         plt.savefig(fig, dpi=300)
623         print(f"Figure written to {fig}.")
624 
625     def extract_values(self):
626         """extracts values needed vor postprocssing and writes them
                ↪  to the DATA dictionary
627         """
628         # load last output
629         last_output = self.output_files[-1]
630         data = load(last_output)
631         fluid_array = data['arrays']['fluid']
632         # rho
633         DATA['rho_min'] = min(fluid_array.rho)
634         DATA['rho_max'] = max(fluid_array.rho)
635         # T
```

```
636             DATA['T_min'] = min(fluid_array.T)
637             DATA['T_max'] = max(fluid_array.T)
638             # aT
639             DATA['aT_min'] = min(fluid_array.aT)
640             DATA['aT_max'] = max(fluid_array.aT)
641
642 # running the program
643 if __name__ == '__main__':
644     # create app
645     app = FFF()
646     # write parameters to be sure to have them
647     app.write_parameters()
648
649     # delete files from last run from WD run app
650     try:
651         app.clear_directory()
652         app.run()
653         DATA['Outcome'] = 'successfull'
654         print(colored('Run successfull', 'green'))
655     except RuntimeError:
656         print(colored('Runtime Error', 'red'))
657         DATA['Outcome'] = 'runtime error'
658     except KeyboardInterrupt:
659         print(colored('Run was aborted.', 'red'))
660         DATA['Outcome'] = 'interrupted'
661     except Exception:
662         print(colored('Run did not finish due to an unknown
            ↪ exception.', 'red'))
663         DATA['Outcome'] = 'failed'
664
665     # post processing
666     app.post_process(app.info_filename)
667     # write parameters again to reflect changes
668     app.write_parameters()
```

A.2 Modifikationen und Erweiterungen für PySPH

A.2.1 fff/basic_flow_variables.py

```
1 """ calculate some additional basic variables
2 """
3
4 from math import sqrt
5
6 from pysph.sph.equation import Equation
```

```
7  from pysph.fff.matrix_operations import
       ↪ double_contraction_sym3x3_with_self
8
9  class ShearRate(Equation):
10     def _get_helpers_(self):
11         return[double_contraction_sym3x3_with_self]
12
13     def initialize(self, d_idx, d_shear_rate,
14         d_v00, d_v01, d_v10, d_v11, d_v02, d_v20, d_v12, d_v21,
               ↪ d_v22):
15         # calculate the symmetric strain rate tensor,
16         # which is the symmetric part of the velocity gradient
17         D00 = d_v00[d_idx]
18         D01 = d_v01[d_idx]+d_v10[d_idx] /2
19         D11 = d_v11[d_idx]
20         D02 = d_v02[d_idx]+d_v20[d_idx] /2
21         D12 = d_v12[d_idx]+d_v21[d_idx] /2
22         D22 = d_v22[d_idx]
23
24         # calculate the diagonal components of the spehric part of
               ↪ D
25         D_spheric = (D00+D11+D22) /3
26
27         # calculate changed components for deviatoric part of D
28         D_dev00 = D00 - D_spheric
29         D_dev11 = D11 - D_spheric
30         D_dev22 = D22 - D_spheric
31
32         # the shear rate is sqrt(2*D':D')
33         d_shear_rate[d_idx] = sqrt(2*
               ↪ double_contraction_sym3x3_with_self(
34             A00=D_dev00, A01=D01, A11=D_dev11, A02=D02, A12=D12,
                   ↪ A22=D_dev22))
```

A.2.2 fff/fff_inlet.py

```
1  """
2  FFF Inlet
3
4  modifiactions to the inlet that uses the modified IOEvaluate
       ↪ equation
5  """
6
7  from pysph.sph.bc.inlet_outlet_manager import InletBase, IOEvaluate
       ↪ , InletInfo
8  from pysph.sph.equation import Equation
9  from pysph.sph.integrator_step import IntegratorStep
10
```

```
11 class FFFInlet(InletBase):
12     """modification of Inlet to use FFFIOEvaluate instead of
           ↪ IOEvaluate"""
13     def _create_io_eval(self):
14         """Evaluator to assign ioid to particles leaving a domain
               ↪ """
15         if self.io_eval is None:
16             from pysph.sph.equation import Group
17             from pysph.tools.sph_evaluator import SPHEvaluator
18             i_name = self.inlet_pa.name
19             f_name = self.dest_pa.name
20             eqns = []
21             eqns.append(Group(equations=[
22                 FFFIOEvaluate(
23                     i_name, [], x=self.x, y=self.y, z=self.z,
24                     xn=self.xn, yn=self.yn, zn=self.zn,
25                     maxdist=self.length, flowspeed=self.inletinfo.
                           ↪ flowspeed)],
26                     real=False, update_nnps=False))
27
28             eqns.append(Group(equations=[
29                 IOEvaluate(
30                     f_name, [], x=self.x, y=self.y, z=self.z,
31                     xn=self.xn, yn=self.yn, zn=self.zn)],
32                     real=False, update_nnps=False))
33
34             if self.gpu:
35                 from pysph.base.gpu_nnps import ZOrderGPUNNPS as
                       ↪ NNPS
36             else:
37                 from pysph.base.nnps import LinkedListNNPS as NNPS
38
39             arrays = [self.inlet_pa] + [self.dest_pa]
40             io_eval = SPHEvaluator(
41                 arrays=arrays, equations=eqns, dim=self.dim,
42                 kernel=self.kernel, nnps_factory=NNPS)
43             return io_eval
44         else:
45             return self.io_eval
46
47 class FFFInletInfo(InletInfo):
48     """
49     This class has additional parameters to allow the inlet to move
           ↪  in the
50     direction of the outflow. Otherwise it's the same as "InletInfo
           ↪ "
51     """
52     def __init__(self, pa_name, normal, refpoint, has_ghost=True,
```

```
53                  update_cls=None, equations=None, umax=1.0,
54                  props_to_copy=None, flowspeed=0.0, timestep=0.0):
55
56          super().__init__(pa_name=pa_name, normal=normal,
57              refpoint=refpoint, has_ghost=has_ghost, update_cls=
                    ↪ update_cls,
58              equations=equations, umax=umax, props_to_copy=
                    ↪ props_to_copy)
59
60          self.flowspeed = flowspeed
61          self.timestep = timestep
62
63  class FFFIOEvaluate(IOEvaluate):
64      """
65      a modification of IOEvaluate to move the refpoint of the inlet
66      """
67      # adding an additional argument for the flowspeed
68      def __init__(self, dest, sources, x, y, z, xn, yn, zn, maxdist
            ↪ =1000, flowspeed=0.0):
69          super().__init__(dest, sources, x, y, z, xn, yn, zn,
                ↪ maxdist)
70
71          self.flowspeed = flowspeed
72
73      # adding reduce function to calculate new refpoint
74      # for some reason the parameters cannot be overridden in
            ↪ py_initialize
75      def reduce(self, dst, t, dt):
76          # components of the flowspeed
77          flow_x = self.flowspeed*self.xn
78          flow_y = self.flowspeed*self.yn
79          flow_z = self.flowspeed*self.zn
80
81          # speed of the inlet particles
82          u = dst.u[0]
83          v = dst.v[0]
84          w = dst.w[0]
85
86          # calculate new inlet refpoint
87          self.x += (u+flow_x)*dt
88          self.y += (v+flow_y)*dt
89          self.z += (w+flow_z)*dt
90
91          # write the refpoint to the arrays for use in other
                ↪ equations
92          dst.ref_x[:] = self.x
93          dst.ref_y[:] = self.y
94          dst.ref_z[:] = self.z
```

```
95
96  class FFFMoveNozzle(Equation):
97      """**moves nozzle for fff pattern**"""
98      def __init__(self, dest, sources, t_wait=0.1,
99          x_length=1.0, y_diff=1.0, z_diff=1.0,
100         y_steps=0, z_steps=0, h_start=1.0,
101         nozzlespeed=0.0, extrusionspeed=0.0):
102         self.t_wait=t_wait
103         self.x_length = x_length
104         self.y_diff = y_diff
105         self.z_diff = z_diff
106         self.y_diff = y_diff
107         self.y_steps = y_steps
108         self.z_steps = z_steps
109         self.h_start = h_start
110         self.nozzlespeed = nozzlespeed
111         self.extrusionspeed = extrusionspeed
112
113         self.move = 'wait'
114         self.x_dir = 1
115         self.z_dir = 1
116         self.row = 0
117         self.level = 0
118         self.u = 0.0
119         self.v = -self.extrusionspeed
120         self.w = 0.0
121
122         super().__init__(dest, sources)
123
124     def reduce(self, dst, t, dt):
125         """calculate velocity components and write them to the
                ↪ inlet array
126         """
127         # dont move for specified amount of time
128         if self.move == 'wait':
129             if t < self.t_wait:
130                 self.u = 0.0
131             else:
132                 # start in x-direction
133                 self.move = 'x'
134                 self.u = self.nozzlespeed*self.x_dir
135
136         elif self.move == 'x':
137             # go in x-direction until either end is reached
138             if 0.0 < dst.ref_x[0] < self.x_length:
139                 self.u = self.nozzlespeed*self.x_dir
140             # on the last row:
```

```
141             elif (self.row==0 and self.z_dir==-1) or (self.row==
                    ↪ self.z_steps and self.z_dir==1):
142                 # stop x-movement, go in y-direction
143                 self.u = 0.0
144                 self.move = 'y'
145                 self.v = self.nozzlespeed - self.extrusionspeed
146                 # invert x_dir and y_dir
147                 self.x_dir *= -1
148                 self.z_dir *= -1
149             # on any other row:
150             else:
151                 # stop x-movement, go in z-direction
152                 self.u = 0.0
153                 self.move = 'z'
154                 self.w = self.nozzlespeed*self.z_dir
155                 # invert x_dir
156                 self.x_dir *= -1
157 
158 
159         elif self.move == 'y':
160             # go in y-direction until reaching the next level
161             if dst.ref_y[0] < (self.level+1)*self.y_diff+self.
                    ↪ h_start:
162                 self.v = self.nozzlespeed - self.extrusionspeed
163             # if there is another level to print
164             elif self.level<self.y_steps:
165                 # stop y-movement, go in x-direction
166                 self.v = -self.extrusionspeed
167                 self.move = 'x'
168                 self.u = self.nozzlespeed*self.x_dir
169                 # change level-counter
170                 self.level += 1
171             # stop when done
172             else:
173                 self.v = 0
174 
175         elif self.move == 'z':
176             # go in z-direction until reaching the next row
177             if (self.row < dst.ref_z[0]/self.z_diff < (self.row+
                    ↪ self.z_dir) and self.z_dir==1
178             or (self.row+self.z_dir) < dst.ref_z[0]/self.z_diff <
                    ↪ self.row and self.z_dir==-1):
179                 self.w = self.nozzlespeed*self.z_dir
180             else:
181                 # change row-counter
182                 self.row += self.z_dir
183                 # stop z-movement, go in x-direction
184                 self.w = 0.0
```

```
185                 self.move = 'x'
186                 self.u = self.nozzlespeed*self.x_dir
187 
188         # set velocities to determined values
189         # print(f'u: {self.u}, v: {self.v}, w: {self.w}')
190         dst.u[:] = self.u
191         dst.v[:] = self.v
192         dst.w[:] = self.w
193 
194 class InletOutletStepOneStage(IntegratorStep):
195     """A trivial integrator for the inlet/outlet particles that
        ↪ uses only one stage
196     """
197 
198     def stage1(self, d_idx, d_x, d_y, d_z, d_u, d_v, d_w, dt):
199         d_x[d_idx] += dt * d_u[d_idx]
200         d_y[d_idx] += dt * d_v[d_idx]
201         d_z[d_idx] += dt * d_w[d_idx]
```

A.2.3 fff/fff_solver.py

```
1  """ a modified solver that does not use adaptive timestep until the
   ↪  wait time is over
2  """
3  # PySPH imports
4  from pysph.solver.solver import Solver
5 
6  class FFFSolver(Solver):
7      """Base class for all PySPH Solvers
8      """
9      def __init__(self, dim=2, integrator=None, kernel=None, n_damp
       ↪ =0, tf=1.0, dt=1e-3,
10                  adaptive_timestep=False, cfl=0.3, output_at_times
                    ↪ =(), fixed_h=False,
11                  adaptive_timestep_critical_time=0.0, **kwargs):
12         """ modified constructer to pass on the adaptive timestep
           ↪ waittime
13         """
14         self.adaptive_timestep_critical_time =
           ↪ adaptive_timestep_critical_time
15         super().__init__(dim=dim, integrator=integrator, kernel=
           ↪ kernel, n_damp=n_damp, tf=tf, dt=dt,
16         adaptive_timestep=adaptive_timestep, cfl=cfl,
           ↪ output_at_times=output_at_times,
17         fixed_h=fixed_h, **kwargs)
18 
```

```
19      #
            ↪ ##########################################################################
            ↪
20      # Non-public interface.
21      #
            ↪ ##########################################################################
            ↪
22      def _compute_timestep(self):
23          """ modified to not use the adaptive timestep close to the
                ↪ specified time
24          """
25          undamped_dt = self._get_undamped_timestep()
26          if self.adaptive_timestep:
27              # locally stable time step
28              dt = self.integrator.compute_time_step(undamped_dt,
                    ↪ self.cfl)
29
30              # set the globally stable time step across all
                    ↪ processors
31              if self.in_parallel:
32                  if dt is None:
33                      # For some reason this processor does not have
                            ↪ an adaptive
34                      # timestep constraint so we set it to a large
                            ↪ number so the
35                      # timestep is determined by the other
                            ↪ processors.
36                      dt = 1e20
37                  dt = self.pm.update_time_steps(dt)
38              else:
39                  if dt is None:
40                      dt = undamped_dt
41
42              # if t is under the critical time, use the estimated
                    ↪ timestep
43              if self.t < 1.1*self.adaptive_timestep_critical_time:
44                  dt = undamped_dt
45
46          else:
47              dt = undamped_dt
48
49          return dt
```

A.2.4 fff/matrix_operations.py

```
1   """ helper functions to perform matrix operations
2   """
3
```

```
4  from math import sqrt, cos, acos, pi as M_PI
5
6  def double_contraction_sym3x3_with_self(A00=0.0, A01=0.0, A11=0.0,
       ↪ A02=0.0, A12=0.0, A22=0.0):
7      r""" a helper function that calculates the double contraction
           ↪ for a
8      symmetric 3x3 matrix with itself
9      """
10     return A00*A00 + 2*A01*A01 + A11*A11 + 2*A02*A02 + 2*A12*A12 +
           ↪ A22*A22
11
12 def smallest_eigenvalue_sym(
13                 m_00=0.0, m_01=0.0, m_02=0.0,
14                           m_11=0.0, m_12=0.0,
15                                     m_22=0.0):
16     r"""**a helper function that calculates the smallest eigenvalue
17      of 2x2 and 3x3 symmetric matrices**"""
18
19     # 2D case
20     if m_02==m_12==m_22==0.0:
21         # determinant and trace of 2x2 matrix
22         trace = m_00 + m_11
23         det = m_00*m_11 - m_01**2
24         # eigenvalues for 2x2 matrix
25         eig_0 = (trace + sqrt(trace**2 - 4*det))/2
26         eig_1 = (trace - sqrt(trace**2 - 4*det))/2
27
28         return min(eig_0, eig_1)
29
30     # 3D case
31     # determinant and trace of 3x3 matrix
32     det = m_00*(m_11*m_22-m_12**2)\
33         - m_01*(m_01*m_22-m_12*m_02)\
34         + m_02*(m_01*m_12-m_11*m_02)
35     trace = m_00 + m_11 + m_22
36
37     p_1 = m_01**2 + m_02**2 + m_12**2
38
39     if p_1 == 0.0: # check wether the matrix is diagonal
40         eig_0 = m_00
41         eig_1 = m_11
42         eig_2 = m_22
43     else: # code if it's not diagonal
44         q = trace/3
45         p_2 = (m_00 - q)**2 + (m_11 - q)**2 + (m_22 - q)**2 + 2*p_1
46         p = sqrt(p_2/6)
47         # B = (1/p) * (A - q*I)
48         b_00 = 1/p * (m_00 - q)
```

```
49          b_11 = 1/p * (m_11 - q)
50          b_22 = 1/p * (m_22 - q)
51          det_b = b_00*(b_11*b_22-m_12*m_12)\
52                  - m_01*(m_01*b_22-m_12*m_02)\
53                  + m_02*(m_01*m_12-b_11*m_02)
54          r = det_b/2
55          if r <= -1.0:
56              phi = M_PI/3
57          elif r >= 1.0:
58              phi = 0.0
59          else:
60              phi = acos(r)/3
61
62          eig_0 = q + 2*p*cos(phi)
63          eig_2 = q + 2*p*cos(phi + (2*M_PI/3))
64          eig_1 = trace - eig_0 - eig_2
65
66      return min(eig_0, eig_1, eig_2)
```

A.2.5 fff/thermal_aha.py

```
1  """modifications to the AHA scheme to use temperature dependent
       ↪ materials
2  """
3
4  from pysph.fff.thermal_edac import FFFMomentumEquationViscosity,
       ↪ FFFSolidWallNoSlipBC
5  from pysph.sph.basic_equations import XSPHCorrection
6  from pysph.sph.equation import Group
7  from pysph.sph.scheme import AdamiHuAdamsScheme
8  from pysph.sph.wc.basic import TaitEOS
9  from pysph.sph.wc.transport_velocity import (
10     ContinuityEquation, ContinuitySolid,
           ↪ MomentumEquationArtificialViscosity,
11     MomentumEquationPressureGradient, SetWallVelocity,
           ↪ SolidWallPressureBC, VolumeSummation)
12
13
14 class ThermalAHAScheme(AdamiHuAdamsScheme):
15     def get_equations(self):
16         equations = []
17         all = self.fluids + self.solids
18
19         g2 = []
20         for fluid in self.fluids:
21             g2.append(VolumeSummation(dest=fluid, sources=all))
22             g2.append(TaitEOS(
```

```
23                  dest=fluid, sources=None, rho0=self.rho0, c0=self.
                        ↪ c0,
24                  gamma=self.gamma, p0=self.p0
25              ))
26          for solid in self.solids:
27              g2.append(VolumeSummation(dest=solid, sources=all))
28              g2.append(SetWallVelocity(dest=solid, sources=self.
                    ↪ fluids))
29
30          equations.append(Group(equations=g2, real=False))
31
32          g3 = []
33          for solid in self.solids:
34              g3.append(SolidWallPressureBC(
35                  dest=solid, sources=self.fluids, b=1.0, rho0=self.
                        ↪ rho0,
36                  p0=self.B, gx=self.gx, gy=self.gy, gz=self.gz
37              ))
38
39          equations.append(Group(equations=g3, real=False))
40
41          g4 = []
42          for fluid in self.fluids:
43              g4.append(
44                  ContinuityEquation(dest=fluid, sources=self.fluids)
45              )
46              if self.solids:
47                  g4.append(
48                      ContinuitySolid(dest=fluid, sources=self.solids
                            ↪ )
49                  )
50              g4.append(
51                  MomentumEquationPressureGradient(
52                      dest=fluid, sources=all, pb=0.0, gx=self.gx,
53                      gy=self.gy, gz=self.gz, tdamp=self.tdamp
54                  )
55              )
56              if self.alpha > 0.0:
57                  g4.append(
58                      MomentumEquationArtificialViscosity(
59                          dest=fluid, sources=all, c0=self.c0,
60                          alpha=self.alpha
61                      )
62                  )
63              g4.append(
64                  FFFMomentumEquationViscosity(
65                      dest=fluid, sources=self.fluids
66                  )
```

```
            )
            if len(self.solids) > 0:
                g4.append(
                    FFFSolidWallNoSlipBC(
                        dest=fluid, sources=self.solids
                    )
                )
            g4.append(XSPHCorrection(dest=fluid, sources=[fluid]))

        equations.append(Group(equations=g4))
        return equations
```

A.2.6 fff/thermal_edac.py

```
"""modifications to the EDAC scheme to use temperature dependent
    ↪ materials
"""

from pysph.sph.basic_equations import XSPHCorrection
from pysph.sph.equation import Equation, Group
from pysph.sph.integrator_step import IntegratorStep
from pysph.sph.wc.edac import (ClampWallPressure, EDACScheme,
    ↪ MomentumEquation,
                               NoSlipVelocityExtrapolation,
                                   ↪ SetWallVelocity,
                               SolidWallPressureBC,
                                   ↪ SourceNumberDensity)
from pysph.sph.wc.transport_velocity import (
    MomentumEquationArtificialViscosity, SummationDensity,
        ↪ VolumeSummation)


class ThermalEDACScheme(EDACScheme):
    """ modified EDAC scheme that uses a variable viscosity
    """
    def get_equations(self):
        if self.use_tvf:
            print('The TVF Equations are not yet modified to use
                ↪ temperature dependent materials!')
            return self._get_internal_flow_equations()
        else:
            return self._get_external_flow_equations()

    def _get_external_flow_equations(self):
        iom = self.inlet_outlet_manager
        fluids_with_io = self.fluids
        all_solids = self.solids + self.inviscid_solids
        if iom is not None:
```

```
29              fluids_with_io = self.fluids + iom.get_io_names()
30          all = fluids_with_io + all_solids
31
32          equations = []
33          # inlet-outlet
34          if iom is not None:
35              io_eqns = iom.get_equations(self, self.use_tvf)
36              for grp in io_eqns:
37                  equations.append(grp)
38
39          group1 = []
40          for fluid in fluids_with_io:
41              group1.append(SummationDensity(dest=fluid, sources=all)
                    ↪ )
42          for solid in self.solids:
43              group1.extend([
44                  SourceNumberDensity(dest=solid, sources=
                        ↪ fluids_with_io),
45                  VolumeSummation(dest=solid, sources=all),
46                  SolidWallPressureBC(dest=solid, sources=
                        ↪ fluids_with_io,
47                                      gx=self.gx, gy=self.gy, gz=self
                                            ↪ .gz),
48                  SetWallVelocity(dest=solid, sources=fluids_with_io)
                        ↪ ,
49              ])
50              if self.clamp_p:
51                  group1.append(
52                      ClampWallPressure(dest=solid, sources=None)
53                  )
54
55          for solid in self.inviscid_solids:
56              group1.extend([
57                  SourceNumberDensity(dest=solid, sources=
                        ↪ fluids_with_io),
58                  NoSlipVelocityExtrapolation(
59                      dest=solid, sources=fluids_with_io),
60                  VolumeSummation(dest=solid, sources=all),
61                  SolidWallPressureBC(dest=solid, sources=
                        ↪ fluids_with_io,
62                                      gx=self.gx, gy=self.gy, gz=self
                                            ↪ .gz)
63                  ])
64
65          equations.append(Group(equations=group1, real=False))
66
67          group2 = []
68          for fluid in self.fluids:
```

```
69              group2.append(
70                  MomentumEquation(
71                      dest=fluid, sources=all, gx=self.gx, gy=self.gy
                            ↪ ,
72                      gz=self.gz, c0=self.c0, tdamp=self.tdamp
73                  )
74              )
75              if self.alpha > 0.0:
76                  sources = fluids_with_io + self.solids
77                  group2.append(
78                      MomentumEquationArtificialViscosity(
79                          dest=fluid, sources=sources, alpha=self.
                                ↪ alpha,
80                          c0=self.c0
81                      )
82                  )
83              if self.nu > 0.0:
84                  group2.append(
85                      FFFMomentumEquationViscosity(
86                          dest=fluid, sources=fluids_with_io
87                      )
88                  )
89              if len(self.solids) > 0 and self.nu > 0.0:
90                  group2.append(
91                      FFFSolidWallNoSlipBC(
92                          dest=fluid, sources=self.solids
93                      )
94                  )
95              group2.extend([
96                  FFFEDACEquation(
97                      dest=fluid, sources=all, cs=self.c0,
98                      rho0=self.rho0
99                  ),
100                 XSPHCorrection(dest=fluid, sources=[fluid],
101                                eps=self.eps)
102             ])
103         equations.append(Group(equations=group2))
104
105         # inlet-outlet
106         if iom is not None:
107             io_eqns = iom.get_equations_post_compute_acceleration()
108             for grp in io_eqns:
109                 equations.append(grp)
110
111         return equations
112 class TEDACStep(IntegratorStep):
113     """a modified stepper to integrate temperatures
114     """
```

```
115     def initialize(self, d_idx, d_x0, d_y0, d_z0, d_x, d_y, d_z,
116                    d_u0, d_v0, d_w0, d_u, d_v, d_w, d_p0, d_p,
117                    d_T, d_T0):
118         d_x0[d_idx] = d_x[d_idx]
119         d_y0[d_idx] = d_y[d_idx]
120         d_z0[d_idx] = d_z[d_idx]
121 
122         d_u0[d_idx] = d_u[d_idx]
123         d_v0[d_idx] = d_v[d_idx]
124         d_w0[d_idx] = d_w[d_idx]
125 
126         d_p0[d_idx] = d_p[d_idx]
127         d_T0[d_idx] = d_T[d_idx]
128 
129     def stage1(self, d_idx, d_x0, d_y0, d_z0, d_x, d_y, d_z,
130                d_u0, d_v0, d_w0, d_u, d_v, d_w, d_p0, d_p,
131                d_au, d_av, d_aw, d_ax, d_ay, d_az, d_ap,
132                d_T, d_T0, d_aT, dt=0):
133         dtb2 = 0.5*dt
134         d_u[d_idx] = d_u0[d_idx] + dtb2*d_au[d_idx]
135         d_v[d_idx] = d_v0[d_idx] + dtb2*d_av[d_idx]
136         d_w[d_idx] = d_w0[d_idx] + dtb2*d_aw[d_idx]
137 
138         d_x[d_idx] = d_x0[d_idx] + dtb2 * d_ax[d_idx]
139         d_y[d_idx] = d_y0[d_idx] + dtb2 * d_ay[d_idx]
140         d_z[d_idx] = d_z0[d_idx] + dtb2 * d_az[d_idx]
141 
142         d_p[d_idx] = d_p0[d_idx] + dtb2 * d_ap[d_idx]
143 
144         d_T[d_idx] = d_T0[d_idx] + dtb2 * d_aT[d_idx]
145 
146     def stage2(self, d_idx, d_x0, d_y0, d_z0, d_x, d_y, d_z,
147                d_u0, d_v0, d_w0, d_u, d_v, d_w, d_p0, d_p,
148                d_au, d_av, d_aw, d_ax, d_ay, d_az, d_ap,
149                d_T, d_T0, d_aT, dt=0):
150         d_u[d_idx] = d_u0[d_idx] + dt*d_au[d_idx]
151         d_v[d_idx] = d_v0[d_idx] + dt*d_av[d_idx]
152         d_w[d_idx] = d_w0[d_idx] + dt*d_aw[d_idx]
153 
154         d_x[d_idx] = d_x0[d_idx] + dt * d_ax[d_idx]
155         d_y[d_idx] = d_y0[d_idx] + dt * d_ay[d_idx]
156         d_z[d_idx] = d_z0[d_idx] + dt * d_az[d_idx]
157 
158         d_p[d_idx] = d_p0[d_idx] + dt * d_ap[d_idx]
159 
160         d_T[d_idx] = d_T0[d_idx] + dt * d_aT[d_idx]
161 
162 class FFFMomentumEquationViscosity(Equation):
```

```
163     """ modified "MomentumEquationViscosity" from pysph.sph.wc.
            ↪ transport_velocity
164     to use variable viscosity
165     """
166
167     def initialize(self, d_idx, d_au, d_av, d_aw):
168         d_au[d_idx] = 0.0
169         d_av[d_idx] = 0.0
170         d_aw[d_idx] = 0.0
171
172     def loop(self, d_idx, s_idx, d_m, d_V, s_V,
173              d_au, d_av, d_aw, d_eta, s_eta,
174              R2IJ, EPS, DWIJ, VIJ, XIJ):
175
176         # averaged shear viscosity Eq. (6)
177         etai = d_eta[d_idx]
178         etaj = s_eta[s_idx]
179
180         etaij = 2 * (etai * etaj)/(etai + etaj)
181
182         # scalar part of the kernel gradient
183         Fij = DWIJ[0]*XIJ[0] + DWIJ[1]*XIJ[1] + DWIJ[2]*XIJ[2]
184
185         # particle volumes, d_V is inverse volume.
186         Vi = 1./d_V[d_idx]
187         Vj = 1./s_V[s_idx]
188         Vi2 = Vi * Vi
189         Vj2 = Vj * Vj
190
191         # accelerations 3rd term in Eq. (8)
192         tmp = 1./d_m[d_idx] * (Vi2 + Vj2) * etaij * Fij/(R2IJ + EPS
                ↪ )
193
194         d_au[d_idx] += tmp * VIJ[0]
195         d_av[d_idx] += tmp * VIJ[1]
196         d_aw[d_idx] += tmp * VIJ[2]
197
198 class FFFSolidWallNoSlipBC(Equation):
199     """modified "SolidWallNoSlipBC" from pysph.sph.wc.
            ↪ transport_velocity
200     to use variable viscosity
201     """
202
203     def initialize(self, d_idx, d_au, d_av, d_aw):
204         d_au[d_idx] = 0.0
205         d_av[d_idx] = 0.0
206         d_aw[d_idx] = 0.0
207
```

```
208     def loop(self, d_idx, s_idx, d_m, d_V, s_V,
209              d_u, d_v, d_w, d_eta, s_eta,
210              d_au, d_av, d_aw,
211              s_ug, s_vg, s_wg,
212              DWIJ, R2IJ, EPS, XIJ):
213
214         # averaged shear viscosity Eq. (6).
215         etai = d_eta[d_idx]
216         etaj = s_eta[s_idx]
217
218         etaij = 2 * (etai * etaj)/(etai + etaj)
219
220         # particle volumes; d_V inverse volume.
221         Vi = 1./d_V[d_idx]
222         Vj = 1./s_V[s_idx]
223         Vi2 = Vi * Vi
224         Vj2 = Vj * Vj
225
226         # scalar part of the kernel gradient
227         Fij = XIJ[0]*DWIJ[0] + XIJ[1]*DWIJ[1] + XIJ[2]*DWIJ[2]
228
229         # viscous contribution (third term) from Eq. (8), with VIJ
230         # defined appropriately using the ghost values
231         tmp = 1./d_m[d_idx] * (Vi2 + Vj2) * (etaij * Fij/(R2IJ +
               ↪ EPS))
232
233         d_au[d_idx] += tmp * (d_u[d_idx] - s_ug[s_idx])
234         d_av[d_idx] += tmp * (d_v[d_idx] - s_vg[s_idx])
235         d_aw[d_idx] += tmp * (d_w[d_idx] - s_wg[s_idx])
236
237 class FFFEDACEquation(Equation):
238     """modified "EDACEquation" from pysph.sph.wc.edac
239     to use variable viscosity
240     """
241
242     def __init__(self, dest, sources, cs, rho0):
243         self.cs = cs
244         self.rho0 = rho0
245
246         super().__init__(dest, sources)
247
248     def initialize(self, d_idx, d_ap):
249         d_ap[d_idx] = 0.0
250
251     def loop(self, d_idx, d_m, d_rho, d_ap, d_p, d_V, s_idx, s_m,
           ↪ s_rho, s_p,
252              s_V, d_eta, s_eta, DWIJ, VIJ, XIJ, R2IJ, EPS):
253
```

```
254             Vi = 1./d_V[d_idx]
255             Vj = 1./s_V[s_idx]
256             Vi2 = Vi * Vi
257             Vj2 = Vj * Vj
258 
259             etai = d_eta[d_idx]
260             etaj = s_eta[s_idx]
261             etaij = 2 * (etai * etaj)/(etai + etaj)
262 
263             # This is the same as continuity acceleration times cs^2
264             rhoi = d_rho[d_idx]
265             rhoj = s_rho[s_idx]
266             vijdotdwij = DWIJ[0]*VIJ[0] + DWIJ[1]*VIJ[1] + DWIJ[2]*VIJ
                    ↪ [2]
267             d_ap[d_idx] += rhoi/rhoj*self.cs*self.cs*s_m[s_idx]*
                    ↪ vijdotdwij
268 
269             # Viscous damping of pressure.
270             xijdotdwij = DWIJ[0]*XIJ[0] + DWIJ[1]*XIJ[1] + DWIJ[2]*XIJ
                    ↪ [2]
271             tmp = 1.0/d_m[d_idx]*(Vi2 + Vj2)*etaij*xijdotdwij/(R2IJ +
                    ↪ EPS)
272             d_ap[d_idx] += tmp*(d_p[d_idx] - s_p[s_idx])
```

A.2.7 fff/thermal_iisph.py

```
1  """ modifications to the IISPH scheme to simualte thermals
2  """
3 
4  from pysph.fff.fff_solver import FFFSolver
5  from pysph.sph.iisph import IISPHScheme, IISPHStep
6  from pysph.sph.integrator_step import IntegratorStep
7 
8 
9  class ThermalIISPHScheme(IISPHScheme):
10     """ modified IISPH Scheme that can use the FFFSolver
11     """
12     def configure_solver(self, kernel=None, integrator_cls=None,
13                          extra_steppers=None, **kw):
14         """Configure the solver to be generated.
15 
16         This is to be called before 'get_solver' is called.
17 
18         Parameters
19         ----------
20 
21         dim : int
22             Number of dimensions.
```

```
23          kernel : Kernel instance.
24              Kernel to use, if none is passed a default one is used.
25          integrator_cls : pysph.sph.integrator.Integrator
26              Integrator class to use, use sensible default if none
                    ↪ is
27              passed.
28          extra_steppers : dict
29              Additional integration stepper instances as a dict.
30          **kw : extra arguments
31              Any additional keyword args are passed to the solver
                    ↪ instance.
32          """
33          from pysph.base.kernels import CubicSpline
34          from pysph.sph.integrator import EulerIntegrator
35          if kernel is None:
36              kernel = CubicSpline(dim=self.dim)
37
38          steppers = {}
39          if extra_steppers is not None:
40              steppers.update(extra_steppers)
41
42          for fluid in self.fluids:
43              if fluid not in steppers:
44                  steppers[fluid] = IISPHStep()
45
46          cls = integrator_cls if integrator_cls is not None else
                ↪ EulerIntegrator
47          integrator = cls(**steppers)
48
49          self.solver = FFFSolver(
50              dim=self.dim, integrator=integrator, kernel=kernel, **
                    ↪ kw
51          )
52
53  class ThermalIISPHStep(IntegratorStep):
54      """A straightforward and simple integrator to be used for IISPH
            ↪  that also integrates the
55      Temperature.
56      """
57      def stage1(self, d_idx, d_x, d_y, d_z, d_u, d_v, d_w, d_uadv,
            ↪ d_vadv, d_wadv, d_au, d_av, d_aw,
58                 d_T, d_aT, dt):
59          d_u[d_idx] = d_uadv[d_idx] + dt * d_au[d_idx]
60          d_v[d_idx] = d_vadv[d_idx] + dt * d_av[d_idx]
61          d_w[d_idx] = d_wadv[d_idx] + dt * d_aw[d_idx]
62
63          d_x[d_idx] += dt * d_u[d_idx]
64          d_y[d_idx] += dt * d_v[d_idx]
```

```
65          d_z[d_idx] += dt * d_w[d_idx]
66
67          d_T[d_idx] += dt * d_aT[d_idx]
```

A.2.8 fff/thermal_tvf.py

```
1  """modifications to the TVF scheme to use it with the thermal
       ↪ equation"""
2
3  from pysph.sph.integrator_step import IntegratorStep
4
5  #
       ↪ ##############################################################################
       ↪
6  # `TransportVelocityStep` class
7  #
       ↪ ##############################################################################
       ↪
8
9
10 class ThermalTransportVelocityStep(IntegratorStep):
11     """Integrator defined in 'A transport velocity formulation for
12     smoothed particle hydrodynamics', 2013, JCP, 241, pp 292--307
13
14     For a predictor-corrector style of integrator, this integrator
15     should operate only in PEC mode.
16
17     """
18     def initialize(self):
19         pass
20
21     def stage1(self, d_idx, d_u, d_v, d_w, d_au, d_av, d_aw, d_uhat
           ↪ , d_auhat, d_vhat,
22                d_avhat, d_what, d_awhat, d_x, d_y, d_z, d_T,
                     ↪ d_aT, dt):
23         dtb2 = 0.5*dt
24
25         # velocity update eqn (14)
26         d_u[d_idx] += dtb2*d_au[d_idx]
27         d_v[d_idx] += dtb2*d_av[d_idx]
28         d_w[d_idx] += dtb2*d_aw[d_idx]
29
30         # advection velocity update eqn (15)
31         d_uhat[d_idx] = d_u[d_idx] + dtb2*d_auhat[d_idx]
32         d_vhat[d_idx] = d_v[d_idx] + dtb2*d_avhat[d_idx]
33         d_what[d_idx] = d_w[d_idx] + dtb2*d_awhat[d_idx]
34
35         # position update eqn (16)
```

```
36          d_x[d_idx] += dt*d_uhat[d_idx]
37          d_y[d_idx] += dt*d_vhat[d_idx]
38          d_z[d_idx] += dt*d_what[d_idx]
39
40          # update Temperature
41          d_T[d_idx] += dtb2*d_aT[d_idx]
42
43      def stage2(self, d_idx, d_u, d_v, d_w, d_au, d_av, d_aw,
            ↪ d_vmag2, d_T, d_aT, dt):
44          dtb2 = 0.5*dt
45
46          # corrector update eqn (17)
47          d_u[d_idx] += dtb2*d_au[d_idx]
48          d_v[d_idx] += dtb2*d_av[d_idx]
49          d_w[d_idx] += dtb2*d_aw[d_idx]
50
51          # magnitude of velocity squared
52          d_vmag2[d_idx] = (d_u[d_idx]*d_u[d_idx] + d_v[d_idx]*d_v[
                ↪ d_idx] +
53                            d_w[d_idx]*d_w[d_idx])
54
55          # update Temperature
56          d_T[d_idx] += dtb2*d_aT[d_idx]
```

A.2.9 fff/thermal_wcsph.py

```
1  from pysph.sph.integrator import IntegratorStep
2
3  #
   ↪ ##############################################################################
   ↪
4  # `TWCSPHStep` class
5  #
   ↪ ##############################################################################
   ↪
6  class TWCSPHStep(IntegratorStep):
7      """Standard Predictor Corrector integrator for the WCSPH
           ↪ formulation
8
9      Use this integrator for WCSPH formulations. In the predictor
           ↪ step,
10     the particles are advanced to `t + dt/2`. The particles are
           ↪ then
11     advanced with the new force computed at this position.
12
13     This integrator can be used in PEC or EPEC mode.
14
15     The same integrator can be used for other problems. Like for
```

```
16      example solid mechanics (see SolidMechStep)
17
18      This version of the step supports a temperature field.
19      """
20      def initialize(self, d_idx, d_x0, d_y0, d_z0, d_x, d_y, d_z,
21                     d_u0, d_v0, d_w0, d_u, d_v, d_w, d_rho0, d_rho,
22                     d_T, d_T0):
23          d_x0[d_idx] = d_x[d_idx]
24          d_y0[d_idx] = d_y[d_idx]
25          d_z0[d_idx] = d_z[d_idx]
26
27          d_u0[d_idx] = d_u[d_idx]
28          d_v0[d_idx] = d_v[d_idx]
29          d_w0[d_idx] = d_w[d_idx]
30
31          d_rho0[d_idx] = d_rho[d_idx]
32
33          d_T0[d_idx] = d_T[d_idx]
34
35      def stage1(self, d_idx, d_x0, d_y0, d_z0, d_x, d_y, d_z,
36                     d_u0, d_v0, d_w0, d_u, d_v, d_w, d_rho0, d_rho,
                           ↪ d_au, d_av,
37                     d_aw, d_ax, d_ay, d_az, d_arho, dt, d_T, d_T0,
                           ↪ d_aT):
38          dtb2 = 0.5*dt
39          d_u[d_idx] = d_u0[d_idx] + dtb2*d_au[d_idx]
40          d_v[d_idx] = d_v0[d_idx] + dtb2*d_av[d_idx]
41          d_w[d_idx] = d_w0[d_idx] + dtb2*d_aw[d_idx]
42
43          d_x[d_idx] = d_x0[d_idx] + dtb2 * d_ax[d_idx]
44          d_y[d_idx] = d_y0[d_idx] + dtb2 * d_ay[d_idx]
45          d_z[d_idx] = d_z0[d_idx] + dtb2 * d_az[d_idx]
46
47          # Update densities and smoothing lengths from the
               ↪ accelerations
48          d_rho[d_idx] = d_rho0[d_idx] + dtb2 * d_arho[d_idx]
49
50          # Update Temperatures
51          d_T[d_idx] = d_T0[d_idx] + dtb2 * d_aT[d_idx]
52
53      def stage2(self, d_idx, d_x0, d_y0, d_z0, d_x, d_y, d_z,
54                     d_u0, d_v0, d_w0, d_u, d_v, d_w, d_rho0, d_rho,
                           ↪ d_au, d_av,
55                     d_aw, d_ax, d_ay, d_az, d_arho, dt, d_T, d_T0,
                           ↪ d_aT):
56
57          d_u[d_idx] = d_u0[d_idx] + dt*d_au[d_idx]
58          d_v[d_idx] = d_v0[d_idx] + dt*d_av[d_idx]
```

```
59          d_w[d_idx] = d_w0[d_idx] + dt*d_aw[d_idx]
60
61          d_x[d_idx] = d_x0[d_idx] + dt * d_ax[d_idx]
62          d_y[d_idx] = d_y0[d_idx] + dt * d_ay[d_idx]
63          d_z[d_idx] = d_z0[d_idx] + dt * d_az[d_idx]
64
65          # Update densities and smoothing lengths from the
                ↪ accelerations
66          d_rho[d_idx] = d_rho0[d_idx] + dt * d_arho[d_idx]
67
68          # Update Temperatures
69          d_T[d_idx] = d_T0[d_idx] + dt * d_aT[d_idx]
```

A.2.10 fff/thermal.py

```
1  """equations to calculate temperature field including surface
       ↪ detection"""
2
3  from math import pi as M_PI
4
5  from pysph.sph.equation import Equation
6  from pysph.fff.matrix_operations import smallest_eigenvalue_sym
7
8  class HeatEquation(Equation):
9      r"""**Heat equation including conduction, radiation and
           ↪ convection**
10
11     The SurfaceDetection equation must be run before this one if
           ↪ radiation
12      or convection are to be considered.
13
14     References
15     ----------
16     .. [Monaghan2005] J. Monaghan, "Smoothed particle hydrodynamics
           ↪ ",
17         Reports on Progress in Physics, 68 (2005), pp. 1703-1759.
18     .. [Afrasiabi2021] Afrasiabi et al., "Smoothed Particle
           ↪ Hydrodynamics
19         Simulation of Orthogonal Cutting with Enhanced Thermal
               ↪ Modeling",
20         Applied Sciences, 11 (2021), pp. 1020-1034.
21
22     """
23     def __init__(self, dest, sources, epsilon=0.0, sigma=5.67e-8,
24         hc=0.0, T_env=20.0, T_zero=-273.15, LambdaCutoff=0.75,
               ↪ shape='cube'):
25         r"""
26         Parameters
```

```
27          ----------
28          epsilon : float
29              emmisivity coefficient (usually between 0.0 and 1.0)
30          sigma : float
31              stefan-boltzman-constant (default is in SI)
32          hc : float
33              heat convection coefficient
34          T_env : float
35              temperature of the environment
36          T_zero : float
37              absolute zero temperature
38          LambdaCutoff : float
39              threshold under which particles are considered to be
                ↪ surface particles
40          shape : string
41              shape the particles represent to calculate their
                ↪ surface
42              Choose from the following:
43                  - 'cube'    (3D)
44                  - 'sphere'  (3D)
45                  - 'square'  (2D)
46                  - 'circle'  (2D)
47          """
48          self.epsilonsigma = epsilon*sigma       # product of
              ↪ emissivity and stefan-boltzmann-const.
49          self.hc = hc                            # heat convection
              ↪ coefficient
50          self.T_zero = T_zero                    # absoulte zero
              ↪ temperature
51          self.T_env = T_env                      # temperature of
              ↪ the environment
52          self.T_envabs4 = (T_env-self.T_zero)**4 # absolute
              ↪ temperature of the environment raised to
53                                                  # the power of 4
54          self.LambdaCutoff = LambdaCutoff        # maximum Lambda
              ↪ value of surface particles
55          self.shape = shape                      # shape of a
              ↪ particle to calculate the surface area
56                                                  # it represents
57
58          super(HeatEquation, self).__init__(dest, sources)
59
60      def initialize(self, d_idx, d_aT, d_T, d_Lambda, d_m, d_rho,
            ↪ d_cp):
61          """calculate surface using Lambda, radiation, convection
62          """
63          # set aT to zero for the new timestep
64          d_aT[d_idx] = 0.0
```

```
65          # V is volume of central particle
66          V = d_m[d_idx] / d_rho[d_idx]
67
68          # calculating the ratio of surface area to volume for a
               ↪ single particle
69          # The particles can represent different 2D and 3D shapes
               ↪ for radiation and convection.
70          if self.shape == 'cube':
71              A = 6.0*V**(2.0/3.0)
72          elif self.shape == 'square':
73              A = 4.0 * V**0.5
74          elif self.shape == 'sphere':
75              A = ((6.0*V)**2.0 * M_PI)**(1.0/3.0)
76          elif self.shape == 'circle':
77              A = 2.0*(M_PI*V)**0.5
78
79          # radiation and convection
80          # you need to run the SurfaceDetection Equation before this
81          if d_Lambda[d_idx] <= self.LambdaCutoff: # check wether
               ↪ particle is on the surface
82              # calculate surface area the particle represents
83              A_free = (1-d_Lambda[d_idx]/self.LambdaCutoff) * A
84
85              # heat loss due to radiation
86              d_aT[d_idx] += A_free*self.epsilonsigma*(self.T_envabs4
                   ↪ -(d_T[d_idx]-self.T_zero)**4) \
87                             /(d_m[d_idx]*d_cp[d_idx])
88              # heat loss due to convection
89              d_aT[d_idx] += A_free*self.hc*(self.T_env-d_T[d_idx]) \
90                             /(d_m[d_idx]*d_cp[d_idx])
91
92      def loop(self, R2IJ, XIJ, DWIJ, EPS, d_idx, s_idx, d_aT, d_cp,
           ↪ s_m, d_rho, s_rho, d_kappa,
93               s_kappa, d_T, s_T):
94          """calculate heat diffusion
95          """
96          xijdotdwij = XIJ[0]*DWIJ[0] + XIJ[1]*DWIJ[1] + XIJ[2]*DWIJ
               ↪ [2]
97
98          tmp = s_m[s_idx] * 4*d_kappa[d_idx]*s_kappa[s_idx] / \
99              (d_cp[d_idx] * d_rho[d_idx]*s_rho[s_idx] * (d_kappa[
                   ↪ d_idx]+s_kappa[s_idx]))
100
101         d_aT[d_idx] += tmp * (d_T[d_idx] - s_T[s_idx]) * xijdotdwij
               ↪  / (R2IJ + EPS)
102
103 class SurfaceDetection(Equation):
```

```
104     r"""**Surface detection using eigenvalues of the
            ↪ renormalization matrix**
105
106     References
107     ----------
108     .. [Afrasiabi2021] Afrasiabi et al., "Smoothed Particle
            ↪ Hydrodynamics
109         Simulation of Orthogonal Cutting with Enhanced Thermal
                ↪ Modeling",
110         Applied Sciences, 11 (2021), pp. 1020-1034.
111     .. [Marrone2010] Marrone et al., "Fast free-surface detection
            ↪ and
112         level-set function definition in SPH solvers", Journal of
113         Computational Physics 229, pp. 3652-3663
114     """
115
116     def _get_helpers_(self):
117         return[smallest_eigenvalue_sym]
118
119     def initialize(self, d_idx, d_Ainv00, d_Ainv01, d_Ainv02,
120                                           d_Ainv11, d_Ainv12,
121                                                     d_Ainv22):
122         """initialize components of matrix A inverted
123         """
124         d_Ainv00[d_idx] = 0.0
125         d_Ainv01[d_idx] = 0.0
126         d_Ainv02[d_idx] = 0.0
127         d_Ainv11[d_idx] = 0.0
128         d_Ainv12[d_idx] = 0.0
129         d_Ainv22[d_idx] = 0.0
130
131
132     def loop(self, d_idx, s_idx, XIJ, DWIJ, s_m, s_rho,
133             d_Ainv00, d_Ainv01, d_Ainv02,
134                       d_Ainv11, d_Ainv12,
135                                 d_Ainv22):
136         """calculate inverted renormalization matrix A_inv
137         """
138         # V_j is volume of neighbor
139         V_j = s_m[s_idx]/s_rho[s_idx]
140         # calculate components of matrix A inverted
141         d_Ainv00[d_idx] += -XIJ[0] * DWIJ[0] * V_j
142         d_Ainv01[d_idx] += -XIJ[0] * DWIJ[1] * V_j
143         d_Ainv02[d_idx] += -XIJ[0] * DWIJ[2] * V_j
144         d_Ainv11[d_idx] += -XIJ[1] * DWIJ[1] * V_j
145         d_Ainv12[d_idx] += -XIJ[1] * DWIJ[2] * V_j
146         d_Ainv22[d_idx] += -XIJ[2] * DWIJ[2] * V_j
147
```

```
148     def post_loop(self, d_idx, d_Lambda,
149             d_Ainv00, d_Ainv01, d_Ainv02,
150                       d_Ainv11, d_Ainv12,
151                                 d_Ainv22):
152         """calculate Lambda from A and saving it
153         """
154         d_Lambda[d_idx] = smallest_eigenvalue_sym(
155             d_Ainv00[d_idx], d_Ainv01[d_idx], d_Ainv02[d_idx],
156                              d_Ainv11[d_idx], d_Ainv12[d_idx],
157                                               d_Ainv22[d_idx])
```

A.2.11 fff/viscosity_models.py

```
1  from math import exp
2
3  from pysph.sph.equation import Equation
4
5  class WLF_Viscosity(Equation):
6      r"""calculates the kinematic viskosity using the WLF modell and
       ↪  rho0
7      this does not take the calculated density or pressure into
       ↪ account
8      with the corrector the viscosity can be adjusted for stability
9      """
10
11     def __init__(self, dest, sources, T_g, eta_g, rho0, nu_min=0.1,
       ↪  nu_max=10, corrector=1.0):
12         super().__init__(dest, sources)
13         self.T_g = T_g
14         self.nu_g = corrector*eta_g/rho0
15         self.nu_min = nu_min
16         self.nu_max = nu_max
17
18     def initialize(self, d_idx, d_T, d_nu):
19         # calculating kinematic viscosity
20         T_diff = d_T[d_idx] - self.T_g
21         d_nu[d_idx] = self.nu_g*10** (-17.44*(T_diff)/(51.6+T_diff)
           ↪ )
22
23         # limits to prevent instability
24         if d_nu[d_idx] < self.nu_min:
25             d_nu[d_idx] = self.nu_min
26         elif d_nu[d_idx] > self.nu_max:
27             d_nu[d_idx] = self.nu_max
28
29 class Cross_WLF_Viscosity(Equation):
30     r"""calculates the kinematic viskosity using the WLF modell and
       ↪  rho0
```

```
31      this does not take the calculated density or pressure into
           ↪ account yet
32      with the corrector the viscosity can be adjusted for stability
33      """
34      def __init__(self, dest, sources, rho0, n, tau_star, D1, D2, D3
           ↪ , A1, A2,
35              use_limits=False, nu_min=0.1, nu_max=10, corrector=1.0)
                   ↪ :
36          super().__init__(dest, sources)
37          self.rho0 = rho0
38          self.T_g = D2 # must move to initialize() and add +D3*p if
               ↪ needed
39          self.exp = 1-n
40          self.tau_star = tau_star
41          self.D1 = D1
42          self.A1 = A1
43          self.A2 = A2
44
45          self.use_limits = use_limits
46          self.corrector = corrector
47          self.nu_min = nu_min
48          self.nu_max = nu_max
49
50      def initialize(self, d_idx, d_T, d_shear_rate, d_nu, d_eta):
51          T_diff = d_T[d_idx] - self.T_g
52
53          # calculating the zero-shear viscosity
54          eta0 = self.D1*exp(-self.A1*T_diff / (self.A2 + T_diff))
55
56          # calculating dynamic viscosity
57          d_eta[d_idx] = self.corrector * eta0 / \
58              (1 + (eta0/self.tau_star*d_shear_rate[d_idx])**self.exp
                   ↪ )
59
60          # calculating kinematic viscosity
61          d_nu[d_idx] = d_eta[d_idx] / self.rho0
62
63          # limits to prevent instability
64          if d_nu[d_idx] < self.nu_min and self.use_limits:
65              d_nu[d_idx] = self.nu_min
66              d_eta[d_idx] = d_nu[d_idx] * self.rho0
67          elif d_nu[d_idx] > self.nu_max and self.use_limits:
68              d_nu[d_idx] = self.nu_max
69              d_eta[d_idx] = d_nu[d_idx] * self.rho0
```

Literaturverzeichnis

[1] Andreas Gebhardt. *Additive Fertigungsverfahren: additive Manufacturing und 3D-Drucken für Prototyping - Tooling - Produktion*. Hanser, München, 5., neu bearbeitete und erweiterte auflage Auflage, 2016. ISBN 978-3-446-44539-0 978-3-446-44401-0.

[2] Ian Gibson, David Rosen und Brent Stucker. *Additive Manufacturing Technologies*. Springer New York, New York, NY, 2015. ISBN 978-1-4939-2112-6. doi: 10.1007/978-1-4939-2113-3.

[3] Yuchu Qin, Qunfen Qi, Paul J. Scott und Xiangqian Jiang. Status, comparison, and future of the representations of additive manufacturing data. *Computer-Aided Design*, 111:44–64, June 2019. ISSN 0010-4485. doi: 10.1016/j.cad.2019.02.004. URL https://www.sciencedirect.com/science/article/pii/S0010448518304202.

[4] Stephan Richter und Steffen Wischmann. Additive Fertigungsmethoden – Entwicklungsstand, Marktperspektiven für den industriellen Einsatz und IKT-spezifische Herausforderungen bei Forschung und Entwicklung, 2016. URL https://vdivde-it.de/de/publikation/additive-fertigungsmethoden-0.

[5] Saad Saleh Alghamdi, Sabu John, Namita Roy Choudhury und Naba K. Dutta. Additive Manufacturing of Polymer Materials: Progress, Promise and Challenges. *Polymers*, 13(5): 753, January 2021. ISSN 2073-4360. doi: 10.3390/polym13050753. URL https://www.mdpi.com/2073-4360/13/5/753. Number: 5 Publisher: Multidisciplinary Digital Publishing Institute.

[6] Ioannis Ntintakis, Georgios E. Stavroulakis, Georgios Sfakianakis und Nikolaos Fiotodimitrakis. Utilizing Generative Design for Additive Manufacturing. In Harshit K. Dave, Uday Shanker Dixit und Dumitru Nedelcu, editors, *Recent Advances in Manufacturing Processes and Systems*, Lecture Notes in Mechanical Engineering, pages 977–989, Singapore, 2022. Springer Nature. ISBN 9789811677878. doi: 10.1007/978-981-16-7787-8_78.

[7] ISO/TC 261 und CEN/TC 438. DIN EN ISO/ASTM 52900: Additive Fertigung - Grundlagen - Terminologie. Technical report, DIN e.V., 2021. URL www.din.de.

[8] J. Martínez, J. L. Diéguez, E. Ares, A. Pereira, P. Hernández und J. A. Pérez. Comparative between FEM Models for FDM Parts and their Approach to a Real Mechanical Behaviour. *Procedia Engineering*, 63:878–884, January 2013. ISSN 1877-7058. doi: 10.1016/j.proeng.2013.08.230. URL https://www.sciencedirect.com/science/article/pii/S1877705813014434.

[9] Ashu Garg und Anirban Bhattacharya. An insight to the failure of FDM parts under tensile loading: finite element analysis and experimental study. *International Journal of Mechanical Sciences*, 120:225–236, January 2017. ISSN 0020-7403. doi: 10.1016/j.ijmecsci.2016.11.032. URL https://www.sciencedirect.com/science/article/pii/S0020740316309596.

[10] Madhukar Somireddy und Aleksander Czekanski. Mechanical Characterization of Additively Manufactured Parts by FE Modeling of Mesostructure. *Journal of Manufacturing and Materials Processing*, 1(2):18, December 2017. ISSN 2504-4494. doi: 10.3390/jmmp1020018. URL https://www.mdpi.com/2504-4494/1/2/18. Number: 2 Publisher: Multidisciplinary Digital Publishing Institute.

[11] Nectarios Vidakis, Markos Petousis, Achilles Vairis, Konstantinos Savvakis und Athena Maniadi. On the compressive behavior of an FDM Steward Platform part. *Journal of Computational Design and Engineering*, 4(4):339–346, October 2017. ISSN 2288-5048. doi: 10.1016/j.jcde.2017.06.001. URL https://doi.org/10.1016/j.jcde.2017.06.001.

[12] Harald Völkl, Johannes Mayer und Sandro Wartzack. Strukturmechanische Simulation additiv im FFF-Verfahren gefertigter Bauteile. In Roland Lachmayer, Katharina Rettschlag und Stefan Kaierle, editors, *Konstruktion für die Additive Fertigung 2019*, pages 143–157, Berlin, Heidelberg, 2020. Springer. ISBN 978-3-662-61149-4. doi: 10.1007/978-3-662-61149-4_10.

[13] Amir Hossein Ehsani, Sadegh Rahmati, Mohammad Nikkhoo, Shahram Etemadi Haghighi und Mohammad Haghpanahi. Using different unit-cell geometries to generate bone tissue scaffolds by additive manufacturing technology. *Proceedings of the Institution of Mechanical Engineers, Part H: Journal of Engineering in Medicine*, 236(6):896–908, June 2022. ISSN 0954-4119. doi: 10.1177/09544119221099786. URL https://doi.org/10.1177/09544119221099786. Publisher: IMECHE.

[14] Andrea Tessarin, Mirco Zaccariotto, Ugo Galvanetto und Domenico Stocchi. A multiscale numerical homogenization-based method for the prediction of elastic properties of components produced with the fused deposition modelling process. *Results in Engineering*,

14:100409, June 2022. ISSN 2590-1230. doi: 10.1016/j.rineng.2022.100409. URL https://www.sciencedirect.com/science/article/pii/S2590123022000792.

[15] Anthony J Favaloro, Bastian Brenken, Eduardo Barocio und R Byron Pipes. Simulation of Polymeric Composites Additive Manufacturing using Abaqus. *Science in the Age of Experience*, page 13, May 2017.

[16] Bastian Brenken, Eduardo Barocio, Anthony Favaloro, Vlastimil Kunc und R. Byron Pipes. Development and validation of extrusion deposition additive manufacturing process simulations. *Additive Manufacturing*, 25:218–226, January 2019. ISSN 22148604. doi: 10.1016/j.addma.2018.10.041. URL https://linkinghub.elsevier.com/retrieve/pii/S2214860418304251.

[17] Yong Zhou, Han Lu, Gongxian Wang, Junfeng Wang und Weidong Li. Voxelization modelling based finite element simulation and process parameter optimization for Fused Filament Fabrication. *Materials & Design*, 187:108409, May 2019. ISSN 0264-1275. doi: 10.1016/j.matdes.2019.108409. URL https://www.sciencedirect.com/science/article/pii/S0264127519308470.

[18] John Kechagias, D. Chaidas, N. Vidakis, K. Salonitis und N.M. Vaxevanidis. Key parameters controlling surface quality and dimensional accuracy: a critical review of FFF process. *Materials and Manufacturing Processes*, 37(9):963–984, July 2022. ISSN 1042-6914, 1532-2475. doi: 10.1080/10426914.2022.2032144. URL https://www.tandfonline.com/doi/full/10.1080/10426914.2022.2032144.

[19] Arup Dey und Nita Yodo. A Systematic Survey of FDM Process Parameter Optimization and Their Influence on Part Characteristics. *Journal of Manufacturing and Materials Processing*, 3(3):64, September 2019. ISSN 2504-4494. doi: 10.3390/jmmp3030064. URL https://www.mdpi.com/2504-4494/3/3/64. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

[20] Sara Garzon-Hernandez, Daniel Garcia-Gonzalez, Antoine Jérusalem und Angel Arias. Design of FDM 3D printed polymers: An experimental-modelling methodology for the prediction of mechanical properties. *Materials and Design*, 188, December 2019. ISSN 18734197. doi: 10.1016/j.matdes.2019.108414. Publisher: Elsevier Ltd.

[21] Jafar Ghorbani, Pratik Koirala, Yu-Lin Shen und Mehran Tehrani. Eliminating voids and reducing mechanical anisotropy in fused filament fabrication parts by adjusting the filament extrusion rate. *Journal of Manufacturing Processes*, 80:651–658, August 2022. ISSN 1526-6125. doi: 10.1016/j.jmapro.2022.06.026. URL https://www.sciencedirect.com/science/article/pii/S1526612522004108.

[22] J. D. Kechagias, N. Vidakis, M. Petousis und N. Mountakis. A multi-parametric process evaluation of the mechanical response of PLA in FFF 3D printing. *Materials and Manufacturing Processes*, 0(0):1–13, June 2022. ISSN 1042-6914. doi: 10.1080/10426914.2022.2089895. URL https://doi.org/10.1080/10426914.2022.2089895. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/10426914.2022.2089895.

[23] L. Li, Q. Sun, C. Bellehumeur und P. Gu. Composite Modeling and Analysis for Fabrication of FDM Prototypes with Locally Controlled Properties. *Journal of Manufacturing Processes*, 4(2):129–141, January 2002. ISSN 1526-6125. doi: 10.1016/S1526-6125(02)70139-4. URL https://www.sciencedirect.com/science/article/pii/S1526612502701394.

[24] Bastian Brenken, Anthony Favaloro, Eduardo Barocio, Nicholas M DeNardo und R Byron Pipes. Development of a Model to Predict Temperature History and Crystallization Behavior of 3D-Printed Parts Made from Fiber-Reinforced Thermoplastic Polymers. *Proceedings of the SAMPE Conference*, page 8, May 2016.

[25] Ke Wu. *Computational Modelling of Fluid-Solid Interaction Problems by Coupling Smoothed Particles Hydrodynamics and the Discrete Element Method*. PhD thesis, The University of Leeds, 2017.

[26] Dongmin Yang, Ke Wu, Lei Wan und Yong Sheng. A Particle Element Approach for Modelling the 3D Printing Process of Fibre Reinforced Polymer Composites. *Journal of Manufacturing and Materials Processing*, 1(1):10, August 2017. doi: 10.3390/jmmp1010010.

[27] Huanxiong Xia, Jiacai Lu, Sadegh Dabiri und Gretar Tryggvason. Fully resolved numerical simulations of fused deposition modeling. Part I: fluid flow. *Rapid Prototyping Journal*, 24 (2):463–476, November 2017. ISSN 1355-2546. doi: 10.1108/RPJ-12-2016-0217.

[28] Huanxiong Xia, Jiacai Lu und Gretar Tryggvason. Fully resolved numerical simulations of fused deposition modeling. Part II – solidification, residual stresses and modeling of the nozzle. *Rapid Prototyping Journal*, 24(6):973–987, November 2017. ISSN 1355-2546. doi: 10.1108/RPJ-11-2017-0233.

[29] Huanxiong Xia, Jiacai Lu und Gretar Tryggvason. A numerical study of the effect of viscoelastic stresses in fused filament fabrication. *Computer Methods in Applied Mechanics and Engineering*, 346:242–259, April 2019. ISSN 00457825. doi: 10.1016/j.cma.2018.11.031. URL https://linkinghub.elsevier.com/retrieve/pii/S0045782518305905.

[30] Huanxiong Xia, Jiacai Lu und Gretar Tryggvason. Simulations of fused filament fabrication using a front tracking method. *International Journal of Heat and Mass Transfer*, 138:1310–

1319, August 2019. ISSN 00179310. doi: 10.1016/j.ijheatmasstransfer.2019.04.132. URL https://linkinghub.elsevier.com/retrieve/pii/S0017931019303254.

[31] Marcin P Serdeczny, Raphaël Comminal, David B Pedersen und Jon Spangenberg. Numerical simulations of the mesostructure formation in material extrusion additive manufacturing. *Additive Manufacturing*, 28:419–429, 2019. ISSN 22148604. doi: 10.1016/j.addma.2019.05.024.

[32] Daniel Hesse, Patrick Hohenberg, Markus Stommel und Zeitschrift Kunststofftechnik. Entwicklung eines rechnergestützten Ansatzes zur Abbildung des Strangablegens in der extrusionsbasierten additiven Fertigung. *Zeitschrift Kunststofftechnik / Journal of Plastics Technology*, 17:3, 2021. URL www.kunststofftech.com.

[33] Ases Akas Mishra, Affaf Momin, Matteo Strano und Kedarnath Rane. Implementation of viscosity and density models for improved numerical analysis of melt flow dynamics in the nozzle during extrusion-based additive manufacturing. *Progress in Additive Manufacturing*, 7(1):41–54, July 2021. ISSN 23639520. doi: 10.1007/s40964-021-00208-z. Publisher: Springer Science and Business Media Deutschland GmbH.

[34] Masato Makino, Daisuke Fukuzawa, Takahiro Murashima, Masaru Kawakami und Hidemitsu Furukawa. Analysis of deposition modeling by particle method simulation. *Microsystem Technologies*, 23(5):1177–1181, June 2016. ISSN 0946-7076. doi: 10.1007/s00542-016-3047-4. URL http://link.springer.com/10.1007/s00542-016-3047-4.

[35] Erwan Bertevas, Julien Férec, Boo Cheong Khoo, Gilles Ausias und Nhan Phan-Thien. Smoothed particle hydrodynamics (SPH) modeling of fiber orientation in a 3D printing process. *Physics of Fluids*, 30(10):103103, October 2018. ISSN 1070-6631. doi: 10.1063/1.5047088.

[36] Zhenyu Ouyang, Erwan Bertevas, Di Wang, Boo Cheong Khoo, Julien Férec, Gilles Ausias und Nhan Phan-Thien. A smoothed particle hydrodynamics study of a non-isothermal and thermally anisotropic fused deposition modeling process for a fiber-filled composite. *Physics of Fluids*, 32(5), May 2020. ISSN 10897666. doi: 10.1063/5.0004527. Publisher: American Institute of Physics Inc.

[37] Christoph Vehring. *Numerische Simulation und experimentelle Validierung der Temperatur- und Druckverhältnisse eines mehrkanaligen Thermoplastdruckkopfes*. PhD thesis, Technische Universität Braunschweig, Braunschweig, 2018. URL https://elib.dlr.de/125074/.

[38] Nils Meyer, Oleg Saburow, Martin Hohberg, Andrew N Hrymak, Frank Henning und Luise Kärger. Parameter Identification of Fiber Orientation Models Based on Direct Fiber Simulation with Smoothed Particle Hydrodynamics. *Journal of Composites Science*, 4(2):77, 2020. doi: 10.3390/jcs4020077.

[39] Hao Zhang, Lixing Zhang, Haoqi Zhang, Jiang Wu, Xizhong An und Dongmin Yang. Fibre bridging and nozzle clogging in 3D printing of discontinuous carbon fibre-reinforced polymer composites: coupled CFD-DEM modelling. *The International Journal of Advanced Manufacturing Technology*, 117(11):3549–3562, December 2021. ISSN 1433-3015. doi: 10.1007/s00170-021-07913-7. URL https://doi.org/10.1007/s00170-021-07913-7.

[40] A. H. Nickel, D. M. Barnett und F. B. Prinz. Thermal stresses and deposition patterns in layered manufacturing. *Materials Science and Engineering: A*, 317(1):59–64, October 2001. ISSN 0921-5093. doi: 10.1016/S0921-5093(01)01179-0. URL https://www.sciencedirect.com/science/article/pii/S0921509301011790.

[41] Y Zhang und Y K Chou. Three-dimensional finite element analysis simulations of the fused deposition modelling process. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 220(10):1663–1671, October 2006. ISSN 0954-4054. doi: 10.1243/09544054JEM572. URL https://doi.org/10.1243/09544054JEM572. Publisher: IMECHE.

[42] Y Zhang und K Chou. A parametric study of part distortions in fused deposition modelling using three-dimensional finite element analysis. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 222(8):959–968, August 2008. ISSN 0954-4054, 2041-2975. doi: 10.1243/09544054JEM990. URL http://journals.sagepub.com/doi/10.1243/09544054JEM990.

[43] Eduardo Barocio, Bastian Brenken, Anthony Favaloro, Miguel Ramirez, Jorge Ramirez und R Byron Pipes. Prediction of the Degree of Bonding in the Extrusion Deposition Additive Manufacturing Process of Semi-Crystalline Polymer Composites. *Science in the Age of Experience*, 2018.

[44] Bastian Brenken, Eduardo Barocio, Anthony Favaloro, Vlastimil Kunc und R. Byron Pipes. Fused filament fabrication of fiber-reinforced polymers: A review. *Additive Manufacturing*, 21:1–16, May 2018. ISSN 2214-8604. doi: 10.1016/j.addma.2018.01.002. URL https://www.sciencedirect.com/science/article/pii/S2214860417304475.

[45] Eduardo Barocio, Bastian Brenken, Anthony Favaloro, Michael Bogdanor und R. Byron Pipes. Extrusion deposition additive manufacturing with fiber-reinforced ther-

moplastic polymers. In *Structure and Properties of Additive Manufactured Polymer Components*, pages 191–219. Elsevier, 2020. ISBN 978-0-12-819535-2. doi: 10.1016/B978-0-12-819535-2.00007-7. URL https://linkinghub.elsevier.com/retrieve/pii/B9780128195352000077.

[46] R. A. Gingold und Joseph J. Monaghan. Smoothed particle hydrodynamics: theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181(3): 375–389, 1977. ISSN 0035-8711. doi: 10.1093/MNRAS/181.3.375.

[47] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *The Astronomical Journal*, 82:1013, December 1977. ISSN 00046256. doi: 10.1086/112164. URL http://adsabs.harvard.edu/cgi-bin/bib_query?1977AJ.....82.1013L.

[48] Joseph J. Monaghan. Smoothed particle hydrodynamics. *Reports on Progress in Physics*, 68(8):1703–1759, 2005. ISSN 0034-4885. doi: 10.1088/0034-4885/68/8/R01. URL https://iopscience.iop.org/article/10.1088/0034-4885/68/8/R01.

[49] Steven J. Lind, Benedict D. Rogers und Peter K. Stansby. Review of smoothed particle hydrodynamics: Towards converged Lagrangian flow modelling: Smoothed Particle Hydrodynamics review. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2241), September 2020. ISSN 14712946. doi: 10.1098/rspa.2019.0801. Publisher: Royal Society Publishing.

[50] Harshit K. Dave und J. Paulo Davim. *Fused Deposition Modeling Based 3D Printing*. Springer International Publishing, Cham, 2021. ISBN 978-3-030-68023-7. doi: 10.1007/978-3-030-68024-4.

[51] Aubrey L. Woern und Joshua M. Pearce. 3-D Printable Polymer Pelletizer Chopper for Fused Granular Fabrication-Based Additive Manufacturing. *Inventions*, 3(4):78, December 2018. ISSN 2411-5134. doi: 10.3390/inventions3040078. URL https://www.mdpi.com/2411-5134/3/4/78. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[52] Ahmed Elkaseer, Stella Schneider und Steffen G. Scholz. Experiment-Based Process Modeling and Optimization for High-Quality and Resource-Efficient FFF 3D Printing. *Applied Sciences*, 10(8):2899, January 2020. ISSN 2076-3417. doi: 10.3390/app10082899. URL https://www.mdpi.com/2076-3417/10/8/2899. Number: 8 Publisher: Multidisciplinary Digital Publishing Institute.

[53] Jingchao Jiang, Xun Xu und Jonathan Stringer. Support Structures for Additive Manufacturing: A Review. *Journal of Manufacturing and Materials Processing*, 2(4):64, December

2018. ISSN 2504-4494. doi: 10.3390/jmmp2040064. URL https://www.mdpi.com/2504-4494/2/4/64. Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[54] Ognjan Luzanin, Dejan Movrin, Vassilis Stathopoulos, Pavlos Pandis, Tanja Radusin und Vera Guduric. Impact of processing parameters on tensile strength, in-process crystallinity and mesostructure in FDM-fabricated PLA specimens. *Rapid Prototyping Journal*, 25 (8):1398–1410, January 2019. ISSN 1355-2546. doi: 10.1108/RPJ-12-2018-0316. URL https://doi.org/10.1108/RPJ-12-2018-0316. Publisher: Emerald Publishing Limited.

[55] Sung-Hoon Ahn, Michael Montero, Dan Odell, Shad Roundy und Paul K. Wright. Anisotropic material properties of fused deposition modeling ABS. *Rapid Prototyping Journal*, 8(4):248–257, January 2002. ISSN 1355-2546. doi: 10.1108/13552540210441166. URL https://doi.org/10.1108/13552540210441166. Publisher: MCB UP Ltd.

[56] José F. Rodríguez, James P. Thomas und John E. Renaud. Mechanical behavior of acrylonitrile butadiene styrene (ABS) fused deposition materials. Experimental investigation. *Rapid Prototyping Journal*, 7(3):148–158, January 2001. ISSN 1355-2546. doi: 10.1108/13552540110395547. URL https://doi.org/10.1108/13552540110395547. Publisher: MCB UP Ltd.

[57] Anna Bellini und Selçuk Güçeri. Mechanical characterization of parts fabricated using fused deposition modeling. *Rapid Prototyping Journal*, 9(4):252–264, January 2003. ISSN 1355-2546. doi: 10.1108/13552540310489631. URL https://doi.org/10.1108/13552540310489631. Publisher: MCB UP Ltd.

[58] Maggie Baechle-Clayton, Elizabeth Loos, Mohammad Taheri und Hossein Taheri. Failures and Flaws in Fused Deposition Modeling (FDM) Additively Manufactured Polymers and Composites. *Journal of Composites Science*, 6(7):202, July 2022. ISSN 2504-477X. doi: 10.3390/jcs6070202. URL https://www.mdpi.com/2504-477X/6/7/202. Number: 7 Publisher: Multidisciplinary Digital Publishing Institute.

[59] Douglas A. J. Brion, Matthew Shen und Sebastian W. Pattinson. Automated recognition and correction of warp deformation in extrusion additive manufacturing. *Additive Manufacturing*, 56:102838, August 2022. ISSN 2214-8604. doi: 10.1016/j.addma.2022.102838. URL https://www.sciencedirect.com/science/article/pii/S2214860422002378.

[60] Dassault Systèmes Simulia Corp. Thermomechanical Simulation of Additive Manufacturing Processes - SIMULIA User Assistance 2022, 2022.

URL https://help.3ds.com/2022/english/DSSIMULIA_Established/SIMACAEANLRefMap/simaanl-c-amthermomechanical.htm?contextscope=cloud#simaanl-c-amthermomechanical.

[61] Bastian Brenken, Eduardo Barocio, Anthony J Favaloro und R. Byron Pipes. Simulation of Semi-Crystalline Composites in the Extrusion Deposition Additive Manufacturing Process. In *Science in the Age of Experience - Proceedings*, pages 90–102, Chicago, Illinois, May 2017. URL https://info.simuleon.com/hubfs/SIMULIA%20Proceedings%202017/SIMULIA%20Science%20in%20the%20age%20of%20Experience%20-%20Proceedings%202017.pdf?hsCtaTracking=bac2c152-91e8-47ce-ad88-beed3d9731d4%7C03618504-5651-48b6-a895-6b136f2f70ac.

[62] Autodesk Inc. Viskositätsmodell nach dem Cross-WLF-Ansatz | Moldflow Adviser, 2022. URL https://knowledge.autodesk.com/de/support/moldflow-adviser/learn-explore/caas/CloudHelp/cloudhelp/2014/DEU/MoldflowAdvisor/files/GUID-7BC3A8F0-8B41-4FCB-BDF1-F1159E4DD175-htm.html.

[63] G. K. Batchelor. *An introduction to fluid dynamics*. Cambridge mathematical library. Cambridge Univ. Press, Cambridge [u.a.], 9. print. Auflage, 2007. ISBN 978-0-521-66396-0.

[64] Eduardo Lozano, Tariq Aslam, Vilem Petr und Gregory S. Jackson. Comparing different water equations of state for aquarium tests. In *AIP Conference Proceedings 2272*, page 070030, Portland, OR, USA, 2020. doi: 10.1063/12.0000807. URL http://aip.scitation.org/doi/abs/10.1063/12.0000807.

[65] Python Particle methods Research. SPH equations - TaitEOS — PySPH 1.0b2.dev0 documentation, 2022. URL https://pysph.readthedocs.io/en/1.0a1/reference/equations.html#pysph.sph.wc.basic.TaitEOS.

[66] Joseph J. Monaghan. Smoothed Particle Hydrodynamics and Its Diverse Applications. *Annual Review of Fluid Mechanics*, 44(1):323–346, January 2012. ISSN 0066-4189, 1545-4479. doi: 10.1146/annurev-fluid-120710-101220. URL https://www.annualreviews.org/doi/10.1146/annurev-fluid-120710-101220.

[67] Corina Höfler. *Entwicklung eines Smoothed Particle Hydrodynamics (SPH) Codes zur numerischen Vorhersage des Primärzerfalls an Brennstoffeinspritzdüsen*. PhD thesis, Karlsruher Institut für Technologie (KIT), 2013.

[68] M. B. Liu, G. R. Liu und K. Y. Lam. Constructing smoothing functions in smoothed particle hydrodynamics with applications. *Journal of Computational and Applied Mathematics*, 155(2):

263–284, June 2003. ISSN 0377-0427. doi: 10.1016/S0377-0427(02)00869-5. URL https://www.sciencedirect.com/science/article/pii/S0377042702008695.

[69] Joseph J. Monaghan. Particle methods for hydrodynamics. *Computer Physics Reports*, 3(2): 71–124, October 1985. ISSN 0167-7977. doi: 10.1016/0167-7977(85)90010-3. URL https://www.sciencedirect.com/science/article/pii/0167797785900103.

[70] L. Brookshaw. A Method of Calculating Radiative Heat Diffusion in Particle Simulations. *Publications of the Astronomical Society of Australia*, 6(2):207–210, 1985. ISSN 1323-3580. doi: 10.1017/S1323358000018117.

[71] Hans F. Schwaiger. An implicit corrected SPH formulation for thermal diffusion with linear free surface boundary conditions. *International Journal for Numerical Methods in Engineering*, 75(6):647–671, 2008. ISSN 1097-0207. doi: 10.1002/nme.2266. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.2266. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nme.2266.

[72] R. Fatehi und M. T. Manzari. Error estimation in smoothed particle hydrodynamics and a new scheme for second derivatives. *Computers & Mathematics with Applications*, 61(2): 482–498, January 2011. ISSN 0898-1221. doi: 10.1016/j.camwa.2010.11.028. URL https://www.sciencedirect.com/science/article/pii/S0898122110009004.

[73] Prabhu Ramachandran, Aditya Bhosale, Kunal Puri, Pawan Negi, Abhinav Muta, A Dinesh, Dileep Menon, Rahul Govind, Suraj Sanka, Amal S Sebastian, Ananyo Sen, Rohan Kaushik, Anshuman Kumar, Vikas Kurapati, Mrinalgouda Patil, Deep Tavker, Pankaj Pandey, Chandrashekhar Kaushik, Arkopal Dutt und Arpit Agarwal. PySPH: A Python-based Framework for Smoothed Particle Hydrodynamics. *ACM Transactions on Mathematical Software*, 47(4):1–38, 2021. ISSN 0098-3500. doi: 10.1145/3460773.

[74] Joseph J. Monaghan. Simulating Free Surface Flows with SPH. *Journal of Computational Physics*, 110(2):399–406, February 1994. ISSN 0021-9991. doi: 10.1006/jcph.1994.1034. URL https://www.sciencedirect.com/science/article/pii/S0021999184710345.

[75] Joseph J. Monaghan. SPH without a Tensile Instability. *Journal of Computational Physics*, 159(2):290–311, April 2000. ISSN 0021-9991. doi: 10.1006/jcph.2000.6439. URL https://www.sciencedirect.com/science/article/pii/S0021999100964398.

[76] S. Marrone, M. Antuono, A. Colagrossi, G. Colicchio, D. Le Touzé und G. Graziani. δ-SPH model for simulating violent impact flows. *Computer Methods in Applied Mechanics and Engineering*, 200(13):1526–1542, March 2011. ISSN 0045-7825. doi: 10.1016/

j.cma.2010.12.016. URL https://www.sciencedirect.com/science/article/pii/S0045782510003725.

[77] Jason P. Hughes und David I. Graham. Comparison of incompressible and weakly-compressible SPH models for free-surface water flows. *Journal of Hydraulic Research*, 48 (sup1):105–117, January 2010. ISSN 0022-1686, 1814-2079. doi: 10.1080/00221686.2010.9641251. URL https://www.tandfonline.com/doi/full/10.1080/00221686.2010.9641251.

[78] S. Adami, X. Y. Hu und N. A. Adams. A transport-velocity formulation for smoothed particle hydrodynamics. *Journal of Computational Physics*, 241:292–307, May 2013. ISSN 10902716. doi: 10.1016/j.jcp.2013.01.043. Publisher: Academic Press Inc.

[79] Chi Zhang, Xiangyu Y. Hu und Nikolaus A. Adams. A generalized transport-velocity formulation for smoothed particle hydrodynamics. *Journal of Computational Physics*, 337: 216–232, May 2017. ISSN 0021-9991. doi: 10.1016/j.jcp.2017.02.016. URL https://www.sciencedirect.com/science/article/pii/S0021999117301092.

[80] S. Adami, X. Y. Hu und N. A. Adams. A generalized wall boundary condition for smoothed particle hydrodynamics. *Journal of Computational Physics*, 231(21):7057–7075, August 2012. ISSN 10902716. doi: 10.1016/j.jcp.2012.05.005. Publisher: Academic Press Inc.

[81] Jonathan R. Clausen. Entropically damped form of artificial compressibility for explicit simulation of incompressible flow. *Physical Review E*, 87(1):013309, January 2013. ISSN 1539-3755, 1550-2376. doi: 10.1103/PhysRevE.87.013309. URL https://link.aps.org/doi/10.1103/PhysRevE.87.013309.

[82] Prabhu Ramachandran und Kunal Puri. Entropically damped artificial compressibility for SPH. *Computers and Fluids*, 179:579–594, January 2019. ISSN 00457930. doi: 10.1016/j.compfluid.2018.11.023. arXiv: 1612.05901 Publisher: Elsevier Ltd.

[83] Dassault Systèmes Simulia Corp. Smoothed Particle Hydrodynamics - SIMULIA User Assistance 2022, 2022. URL https://help.3ds.com/2022/english/dssimulia_established/simacaeanlrefmap/simaanl-c-sphanalysis.htm?contextscope=all.

[84] Dassault Systèmes Simulia Corp. Defining an equation of state - SIMULIA User Assistance 2022, 2022. URL https://help.3ds.com/2022/english/dssimulia_established/simacaecaerefmap/simacae-t-prpmechanicalothereosdefine.htm?contextscope=all.

[85] Dassault Systèmes Simulia Corp. Abaqus User Subroutines Guide - SIMULIA User Assistance 2022, 2022. URL https://help.3ds.com/2022/english/dssimulia_established/SIMACAESUBRefMap/simasub-c-ov.htm?contextscope=all&id=9f5af922e7b5418fb3741e170ad1b1ff.

[86] Larry D. Libersky, Phil W. Randles, Ted C. Carney und David L. Dickinson. Recent improvements in SPH modeling of hypervelocity impact. *International Journal of Impact Engineering*, 20(6):525–532, January 1997. ISSN 0734-743X. doi: 10.1016/S0734-743X(97)87441-6. URL https://www.sciencedirect.com/science/article/pii/S0734743X97874416.

[87] Larry D. Libersky, Albert G. Petschek, Theodore C. Carney, Jim R. Hipp und Firooz A. Allahdadi. High Strain Lagrangian Hydrodynamics: A Three-Dimensional SPH Code for Dynamic Material Response. *Journal of Computational Physics*, 109(1):67–75, November 1993. ISSN 0021-9991. doi: 10.1006/jcph.1993.1199. URL https://www.sciencedirect.com/science/article/pii/S002199918371199X.

[88] Gordon R. Johnson, Robert A. Stryk und Stephen R. Beissel. SPH for high velocity impact computations. *Computer Methods in Applied Mechanics and Engineering*, 139(1):347–373, December 1996. ISSN 0045-7825. doi: 10.1016/S0045-7825(96)01089-4. URL https://www.sciencedirect.com/science/article/pii/S0045782596010894.

[89] Andrea Colagrossi und Maurizio Landrini. Numerical simulation of interfacial flows by smoothed particle hydrodynamics. *Journal of Computational Physics*, 191(2):448–475, November 2003. ISSN 0021-9991. doi: 10.1016/S0021-9991(03)00324-3. URL https://www.sciencedirect.com/science/article/pii/S0021999103003243.

[90] Autodesk Inc. Moldflow Insight Help | About material databases, 2021. URL https://help.autodesk.com/view/MFIA/2021/ENU/?guid=GUID-9C852155-1ECC-4A68-A45D-F3FC5E79E057.

[91] Roland Gomeringer, Max Heinzler, Roland Kilgus, Volker Menges, Friedrich Näher, Stefan Oesterle, Claudius Scholer, Andreas Stephan und Falko Wieneke. *Tabellenbuch Metall*. Verlag Europa Lehrmittel, Haan-Gruiten, 46 Auflage, 2014. ISBN 978-3-8085-1726-0.

[92] Dassault Systèmes Simulia Corp. Smoothed particle hydrodynamic analysis - SIMULIA User Assistance 2017, 2017. URL https://help.3ds.com/2017/english/dssimulia_established/simacaeverrefmap/simaver-c-sphanalysis.htm?contextscope=all.

[93] Dassault Systèmes Simulia Corp. Continuum Particle Element Library - SIMULIA User Assistance 2022, 2022. URL https://help.3ds.com/2022/english/dssimulia_established/simacaeelmrefmap/simaelm-r-sphelemlib.htm?contextscope=all.

[94] Dassault Systèmes Simulia Corp. Particle Generator - SIMULIA User Assistance 2022, 2022. URL https://help.3ds.com/2022/english/dssimulia_established/SIMACAEANLRefMap/simaanl-c-particlegenerator.htm?contextscope=all.

[95] Dassault Systèmes Simulia Corp. Equation of State - SIMULIA User Assistance 2022, 2022. URL https://help.3ds.com/2022/english/dssimulia_established/simacaematrefmap/simamat-c-eos.htm?contextscope=all#simamat-c-eos-mg.

[96] Python Particle methods Research. Welcome to the PySPH documentation! — PySPH 1.0b2.dev0 documentation, 2022. URL https://pysph.readthedocs.io/en/latest/index.html#.

[97] Python Particle methods Research. PySPH, September 2022. URL https://github.com/pypr/pysph. original-date: 2017-05-25T17:17:00Z.

[98] Python Particle methods Research. Module solver - InletBase — PySPH 1.0b2.dev0 documentation, 2022. URL https://pysph.readthedocs.io/en/latest/reference/solver.html?highlight=inletbase#pysph.sph.bc.inlet_outlet_manager.InletBase.

[99] Paul W Cleary und Joseph J. Monaghan. Conduction Modelling Using Smoothed Particle Hydrodynamics. *Journal of Computational Physics*, 148(1):227–264, January 1999. ISSN 0021-9991. doi: 10.1006/jcph.1998.6118. URL https://www.sciencedirect.com/science/article/pii/S0021999198961186.

[100] Mathieu Doring. *Développement d'une méthode SPH pour les applications à surface libre en hydrodynamique*. These de doctorat, Nantes, 2005. URL https://www.theses.fr/en/2005NANT2116.

[101] S Marrone, A Colagrossi, D Le Touzé und G Graziani. Fast free-surface detection and level-set function definition in SPH solvers. *Journal of Computational Physics*, 229(10): 3652–3663, 2010. ISSN 00219991. doi: 10.1016/j.jcp.2010.01.019.

[102] Mohamadreza Afrasiabi, Hagen Klippel, Matthias Roethlin und Konrad Wegener. Smoothed Particle Hydrodynamics Simulation of Orthogonal Cutting with Enhanced Thermal Modeling. *Applied Sciences*, 11(3):1020–1034, 2021. doi: 10.3390/app11031020.

[103] Phil W. Randles und Larry D. Libersky. Smoothed Particle Hydrodynamics: Some recent improvements and applications. *Computer Methods in Applied Mechanics and Engineering*, 139(1-4):375–408, 1996. ISSN 00457825. doi: 10.1016/S0045-7825(96)01090-0.

[104] Markus Rütten. Theoretische Grundlagen. In Markus Rütten, editor, *Verallgemeinerte newtonsche Fluide: Thermische und viskose Strömungseigenschaften*, pages 7–81. Springer, Berlin, Heidelberg, 2019. ISBN 978-3-662-56226-0. doi: 10.1007/978-3-662-56226-0_2. URL https://doi.org/10.1007/978-3-662-56226-0_2.

[105] Dassault Systèmes Simulia Corp. Three-Dimensional Solid Element Library - SIMULIA User Assistance 2022, 2022. URL https://help.3ds.com/2022/english/dssimulia_established/SIMACAEELMRefMap/simaelm-r-3delem.htm?contextscope=all&id=5915478e847e4112b8993f746418ee66#simaelm-r-3delem-t-elemtypes1.

[106] Dassault Systèmes Simulia Corp. Thermal Contact Properties - SIMULIA User Assistance 2022, 2022. URL https://help.3ds.com/2022/english/dssimulia_established/SIMACAEITNRefMap/simaitn-c-thermalinteraction.htm?contextscope=all.