

■ fakultät für informatik

Projektgruppe 562 ChipSim

Modellierung und Simulation
plastoelastischer
Objektinteraktionen

Endbericht SoSe 2012


14. November 2012


Teilnehmer:

Jim Bergmann, Jan Bessai, Andre Droschinsky,
Oliver Jungeilges, Armin Kazmi, Daniel Neugebauer,
Ercan Özdemir, Andrew Quinn, Christoph Schikora,
Christian Stossno, Stefan Voss

Betreuer:

Prof. Dr. Heinrich Müller
Dipl.-Inf. Denis Fisseler
Jun.-Prof. Dr.-Ing. Dipl.-Inf. Petra Kersting
Dipl.-Inf. Raffael Joliet

 Lehrstuhl Informatik VII
Graphische Systeme

 ISF Institut für Spanende Fertigung

 fakultät für
informatik

Inhaltsverzeichnis

1. Einleitung	1
1.1. Aufgabenstellung	2
1.2. Gliederung	2
2. Ziele der Projektgruppe	3
2.1. Teilziele	3
2.2. Organisatorisches und Vorgehensweise	4
3. Grundlagen verwendeter Modelle	7
3.1. Spanbildung	7
3.1.1. Teile eines Werkzeugs	8
3.1.2. Bezugssysteme	9
3.1.3. Verfahrensvarianten	11
3.1.4. Spanbildung	12
3.1.5. Spanform	13
3.2. Kontaktmechanik	15
3.2.1. Kontakt zwischen Kugel und Ebene	16
3.2.2. Kontakt zwischen Kegel und Ebene	17
3.3. Smoothed Particle Hydrodynamics	18
3.3.1. Glättungsfunktion	19
3.3.2. Simulation mit SPH	21
3.3.3. PCISPH	24
3.3.4. Simulation von Festkörpern	26
3.3.5. Bewertung und Ausblick	30
3.4. Movable Cellular Automata	31
3.4.1. Zustände	31
3.4.2. Auftretende Kräfte	32
3.4.3. Kollisionsbehandlung	36
3.4.4. Bewertung	37
3.5. Feder-Masse-Systeme	37
3.5.1. Aufbau eines Feder-Masse-Systems	38
3.5.2. Anwendung	39
3.5.3. Bewertung	39

3.6.	Nachbarschaftssuche	40
3.6.1.	Abstandsberechnung	41
3.6.2.	Naive Methode	41
3.6.3.	Sortierungsverfahren	42
3.6.4.	Äquidistante Raumaufteilung	43
3.6.5.	Nicht äquidistante Raumaufteilung	45
3.6.6.	Hashing	46
4.	Konzeption	49
4.1.	Festkörpergeometrie	49
4.2.	Boundary-Partikel	50
4.3.	Modellierung eines Bodens	51
4.3.1.	Implizite Flächen	51
4.3.2.	Partikelbasierter Boden	52
4.4.	Konstruktion eines Werkzeugs	52
4.4.1.	Modellierung	52
4.4.2.	Werkzeugkonfiguration	53
4.5.	Fixieren des Werkzeugs	53
4.6.	Oberflächenrauheit des Werkstücks	54
5.	Projektbeschreibung	55
5.1.	Prototyp	55
5.2.	Endprodukt	56
6.	Implementierung	59
6.1.	Simulator	60
6.1.1.	Startphase	60
6.1.2.	Initialisierungsphase	60
6.1.3.	Simulationsphase	60
6.2.	Visualisierung	63
6.2.1.	Resource manager	63
6.2.2.	Renderwidget	63
6.2.3.	Shader	64
6.3.	Generator	65
6.3.1.	Geometrien	66
6.3.2.	Das Generator-Interface	67
6.3.3.	Projektdateiparser	70
6.4.	Nachbarschaftssuche	71
6.4.1.	Datenstruktur	72
6.4.2.	Initialisierung	72
6.4.3.	Aktualisierung der Zelleninhalte	73
6.4.4.	Aktualisierung der Nachbarschaften	73

6.4.5.	Aktualisierung der Datenstruktur	74
6.4.6.	Ausblick	74
6.5.	Smoothed Particle Hydrodynamics	77
6.5.1.	Initialisierung	77
6.5.2.	Berechnung der Zustandsvariablen	77
6.5.3.	Berechnung der Kräfte	78
6.5.4.	Optimierung durch Ausnutzen der Symmetrie	78
6.5.5.	PCISPH	79
6.6.	Movable Cellular Automata	79
6.7.	GPU-Solver	82
7.	Auswertung	85
7.1.	MCA-Testfälle	85
7.1.1.	Ruhelage eines Würfels unter Gravitation auf festem Boden	85
7.1.2.	Aufprall eines Würfels auf festem Boden unter Gravitation	87
7.1.3.	Fall eines Würfels unter Gravitation auf einen Keil	90
7.1.4.	Ruhelage eines Festkörperstabes	95
7.2.	SPH-Testfälle	99
7.2.1.	Auffangen einer Flüssigkeit	99
7.2.2.	Ruhelage eines Würfel unter Gravitation auf festem Boden	100
7.2.3.	Fall eines Würfel unter Gravitation auf einen Keil	102
7.3.	Vergleich mit der Mechanik	106
7.4.	Tests zur Spanbildung	107
7.4.1.	Spanbildungsverhalten beim MCA-Verfahren	107
7.4.2.	Spanbildungsverhalten beim SPH-Verfahren	111
7.5.	Diskussion der Ergebnisse	114
7.5.1.	Grundlegende Materialeigenschaften	114
7.5.2.	Schneid- und Spanbildungseigenschaften	115
7.5.3.	Gesamtbewertung	117
8.	Zusammenfassung und Ausblick	121
	Literaturverzeichnis	123
A.	Pflichtenheft	129
A.1.	Zielbestimmungen	129
A.1.1.	Muss-Kriterien	129
A.1.2.	Prototyp	129
A.1.3.	Optionale Kriterien	130
A.1.4.	Demonstrationsfälle / Minimalziele	130
A.1.5.	Anmerkungen zum Pflichtenheft	130

B. Zeitplan	133
C. Benutzerhandbuch	135

Notation in SI-Einheiten

Smoothed Particle Hydrodynamics

h	Glättungslänge
\vec{x}_{ij}	Abstandsvektor der Partikel i und j
$W(\vec{x}_{ij}, h)$	Glättungsfunktion
m_i	Masse des Partikels i
ρ_i	Dichte des Partikels i
p_i	Druck des Partikels i
k	Steifheitskonstante
ρ_0	Ruhedichte
γ	Gamma, konstant 7
μ_i	Viskositätskonstante des Partikels i
\vec{v}_i	Geschwindigkeit des Partikels i
U_i	Energiedichte des Partikels i
\bar{V}_i	Volumen des Partikels i
ϵ	Dehnungstensor
σ	Spannungstensor
C	Elastizitätstensor
\vec{u}_i	Partikelpositionsabweichung zur Anfangsposition von Partikel i
λ	Lamémodul
μ	Schubmodul
γ_Y	Obergrenze scherende Formabweichung
γ_c	Obergrenze plastische Verformung

Movable Cellular Automata

r_i	Radius des Automaten i
r_{ij}	Mittelpunktsabstand der Automaten i und j
\vec{p}_{ij}	Zentralkräfte der Automaten i und j
\vec{f}_{ij}	Tangentialkräfte der Automaten i und j
α_{ij}	Tiefe des Lennard-Jones-Potentials
σ_{ij}	Automatenabstand im Gleichgewichtszustand
$\vec{\omega}_i$	Winkelgeschwindigkeit des Automaten i
$\vec{\omega}_{ij}$	Winkelgeschwindigkeit der Rotation des Automatenpaars i und j
\vec{q}_{ij}	Vektor zwischen dem Mittelpunkt des Automaten i und dem Mittelpunkt der Kontaktfläche der Automaten i und j
\vec{q}_i	Abstandsvektor vom Mittelpunkt des Automaten i zum nächsten Kontaktpunkt mit einem anderen Automaten
\vec{v}_{ij}	Gesamtgleitgeschwindigkeit der Automaten i und j

η_{ij}	Reibungskoeffizienten zwischen den Automaten i und j
γ_i	Winkel der Verformungsverschiebung von Automat i
G_i	Schubmodul für den Automaten i
τ_{ij}	Experimentell ermittelte Spannungsfunktion der Scheerspannung zwischen den Automaten i und j
S_{ij}	Verformungsfläche zwischen den Automaten i und j
\vec{K}_{ij}	Drehmoment, der von der Kraft, die Automat j auf den Automaten i ausübt, entsteht
\hat{j}_i	Trägheitsmoment des Automaten i
μ_{ij}	Trockenreibungskonstante zwischen den Automaten i und j
\vec{v}_{c_i}	Kollisionsgeschwindigkeit des Automaten i
\vec{v}_{oc_i}	orthogonale Kollisionsgeschwindigkeit des Automaten i
\vec{v}_{oi}	orthogonale Geschwindigkeit des Automaten i
\vec{d}	Richtungsvektor zwischen den Automaten j und i

Feder-Masse-Systeme

ε	Amplitude Lennard-Jones-Potential
σ	Nullstelle Lennard-Jones-Potential

Nachbarschaftssuche

r	Suchradius
-----	------------

Kontaktmechanik

E_i	Elastizitätsmodul von Objekt i
ν_i	Querdehnungszahl/Poissonzahl von Objekt i
E^*	Gesamtelastizitätsmodul von zwei Objekten
a	Radius Kontaktfläche
d	Eindrucktiefe
R_i	Radius von Objekt i
r	Abstand vom Mittelpunkt der Kontaktfläche
p_0	Maximaler Druck
θ	Winkel zwischen Kegel und Boden

Spannbildung

\vec{v}_e	Wirkgeschwindigkeit
\vec{v}_c	Schnittgeschwindigkeit
\vec{v}_f	Vorschubgeschwindigkeit
β	Spanwinkel

1. Einleitung

Spanende Fertigungsverfahren werden beispielsweise in der Luft- und Raumfahrttechnik eingesetzt, um unterschiedliche Komponenten (Turbinenschaufeln o.ä.) herzustellen [HCL⁺05]. Um den Materialabtrag und die entstehenden Kräfte abbilden zu können, werden Simulationssysteme genutzt, die die Veränderungen der Geometrie akkurat simulieren können [WMK⁺02]. Zur besseren Analyse eines Fräsprozesses muss der Materialabtrag genauer betrachtet werden. Die sogenannte Spanbildung kann allgemein so beschrieben werden, dass ein fester Körper (beispielsweise eine Schneide) sich durch einen plastoelastischen Körper (beispielsweise ein Werkstück) bewegt (vgl. Abbildung 1.1). Bereits existierende Finite-Elemente-Simulationen, die diesen Prozess simulieren, sind jedoch sehr rechenaufwendig. Im Vergleich dazu bieten partikelbasierte Methoden die Möglichkeit, den Prozess einfacher zu beschreiben und eine Neuvernetzung des Spans zu vermeiden, was eine schnellere Simulation ermöglichen kann [WMK⁺02, Bic07].

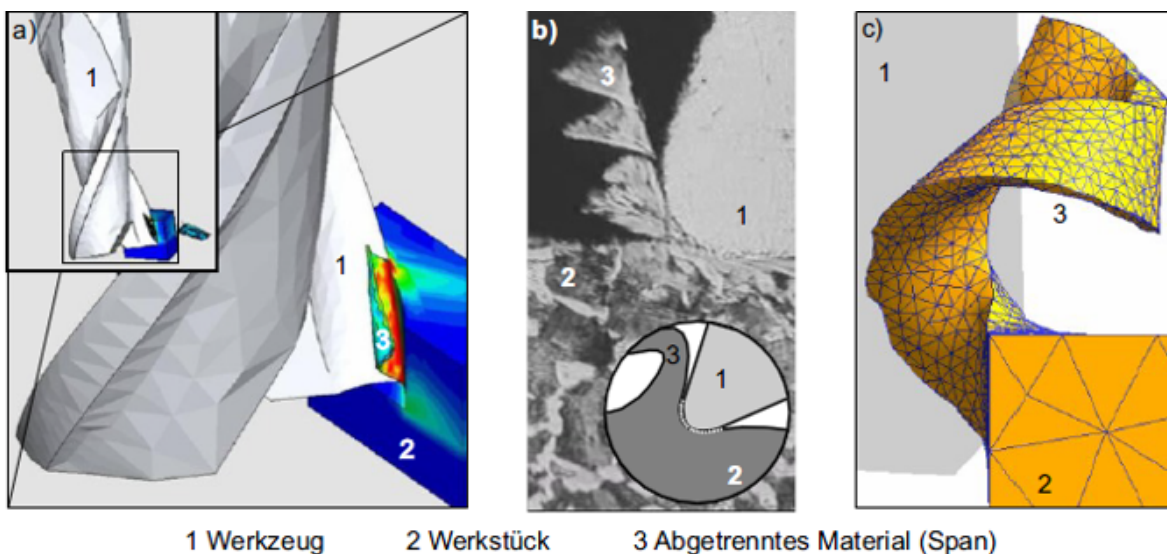


Abbildung 1.1.: a) Simulation eines Spanbildungsprozesses (DaimlerChrysler AG, wbk)
b) Reale Spanbildung. Es wird Material abgenommen (Span) und unter der Schneide hindurchgedrückt (iwf)
c) Vernetzte Modellierung eines Spans (Kennametal Technologies GmbH)[MFJK12]

1.1. Aufgabenstellung

Im Wintersemester 2011/2012 hat am Lehrstuhl 7 (Graphische Systeme) der Fakultät Informatik der TU Dortmund in Kooperation mit dem Institut für Spanende Fertigung die Projektgruppe ChipSim, mit der Aufgabe, ein Simulations- und Visualisierungssystem zu entwerfen und zu implementieren, um die Spanbildung untersuchen zu können, begonnen. Das entwickelte System soll plastoelastische Objektinteraktionen realistisch darstellen. Dabei ist es notwendig, dass die Kinematik der simulierten Objekte durch bestimmte externe und interne physikalische Kräfte sowie durch Kollisionen von Schneide und Werkstück beeinflusst wird. Als externe Kraft kann z.B. die Gravitationskraft, als interne Kräfte bzw. Eigenschaften z.B. der Druck und die Viskosität berücksichtigt werden.

Zur Realisierung wurden dafür verschiedene Methoden als Grundlage angedacht. Einen vielversprechenden Ansatz stellte hierbei die Smoothed Particle Hydrodynamics (SPH) Methode dar [AW09], die beim Simulieren von Flüssigkeiten und Gasen bereits gute Ergebnisse lieferte. Zusätzlich wurden auch strukturell andere Methoden, beispielsweise Feder-Masse-Systeme [NMK⁺05] und die Methode der beweglichen zellulären Automaten (MCA), betrachtet. Eine Kombination von mehreren Methoden wurde ebenfalls in Erwägung gezogen.

1.2. Gliederung

Dieser Endbericht fasst die gewählten Methoden sowie die während der zwei Semester erzielten Ergebnisse bei der Modellierung einer Spanbildungssimulation zusammen und bewertet die Güte dieser Ergebnisse. In Kapitel 2 werden die angestrebten Projektziele näher erläutert. Die allgemeinen organisatorischen Rahmenbedingungen und Vorgehensweisen werden ebenfalls dargestellt. Kapitel 3 beschreibt die von der Projektgruppe betrachteten Modelle und skizziert deren Grundlagen. Zusätzlich werden die vorgestellten Modelle bezüglich ihrer Verwendbarkeit für die Simulation der Spanbildung bewertet. Kapitel 4 beinhaltet modellunabhängige Konzepte zu allgemeinen Komponenten der Simulation, wie zum Beispiel die Modellierung eines Werkzeugs zur Spanbildung. Hier finden Entscheidungen Platz, die nicht konzeptionell zu einem grundlegenden Simulations-Modell gehört. Kapitel 5 beschreibt den Rahmen des Software-Projekts, wobei umgesetzte und nicht umgesetzte Funktionalitäten kurz beschrieben werden. In Kapitel 6 sind spezifischere Details der Implementierung der Simulation festgehalten. Jede vorhandene Komponente wird an dieser Stelle konkret vorgestellt und erläutert. In Kapitel 7 folgt eine Auswertung des möglichen Simulationsverhaltens. Weiterhin werden allgemeine und spezielle Fälle der Spanbildung getestet und ausgewertet. Das achte Kapitel resümiert die erzielten Ergebnisse und gibt einen Ausblick auf noch mögliche Verbesserungen und Erweiterungen der Simulation.

2. Ziele der Projektgruppe

Um für die Simulation der Bildung eines Spans eine Lösung zu finden, galt es, in der Projektgruppe Modellierungs- und Visualisierungsmethoden sowie mathematische Lösungsverfahren zu entwerfen und zu implementieren. Ziel war es, diese auf einfache geometrische Beispiele und den komplexen Spanbildungsvorgang anzuwenden.

2.1. Teilziele

Um diese Ziele umzusetzen, wurden folgende Teilziele zur Bearbeitung in Kleingruppen identifiziert:

1. Modellerstellung:

Es musste ein Modell erstellt werden, mit dem sowohl einfache, später auch komplexere geometrische Objekte samt ihrer Eigenschaften, die beispielsweise aus einer XML-Datei gelesen werden sollten, beschrieben werden können. Objektorientierung galt es als Designmerkmal möglichst ohne Leistungseinbußen umzusetzen. Bereits bekannte Methoden, wie z.B. SPH [Mon05] oder MCA [PHK⁺95], sollten zu diesem Zweck an die gegebene Aufgabenstellung angepasst werden.

2. Kollisionsdetektion:

Gegebenenfalls mussten Methoden zur Kollisionserkennung entworfen werden, um Interaktionen zwischen partikelbasierten und nicht-partikelbasierten Komponenten zu simulieren. Des Weiteren wurden auch Techniken wie Raumunterteilungsverfahren betrachtet (siehe Abschnitt 3.6), um die Simulation zu beschleunigen [AM91].

3. 3D-Visualisierung:

Um die verwendeten Modelle und Ergebnisse vergleichen zu können, galt es, eine Visualisierung zu implementieren, die zeitnah den Simulationsablauf darstellt. Physikalische Größen, wie z.B. Geschwindigkeiten, sollen einzeln betrachtet werden können.

4. GPGPU-Programmierung:

Grafische Programmierschnittstellen wie OpenGL [MB05] und GPGPU-Techniken (Cuda, OpenCL) wurden in Betracht gezogen, um eine interaktive Visualisierung zu implementieren und Berechnungen zu parallelisieren und zu beschleunigen.

5. Grundlegende Tests mit geometrischen Primitiven:

Um die grundsätzliche Funktionalität des entwickelten Simulationsframeworks zu demonstrieren, wurden Simulationen mit vorgegebenen geometrischen Primitiven, wie Quadern, ermöglicht. Dazu wurden Testfälle definiert, die zu Demonstrationszwecken sinnvolle Eigenschaften einer Partikelsimulation im Bereich der Spanbildung aufweisen (siehe Abschnitt 7.1). Eine Visualisierung, die es unter anderem erlaubt, die Kollisionseffekte darzustellen, war hierbei wünschenswert.

6. Realisieren einer Zerspansimulation:

Sobald mit dem entwickelten Framework erste gute Ergebnisse erzielt wurden, war ein weiterer Ausbau zu einer Simulation geplant, in der der Materialabtrag während eines Fräsprozesses abgebildet werden kann. Hierzu wurde ein Werkzeug auf festgelegten Bahnen bewegt, um durch die implementierten Partikelinteraktionen Zerspanungseffekte zu realisieren.

Bei der Umsetzung dieser Anforderungen in ein Softwaresystem war es von Vorteil, dass verschiedene Entwurfsphasen durchlaufen wurden [Bal96], welche u.a. die Erstellung von Lasten- und Pflichtenheft oder die Erstellung von Gantt Charts beinhalteten. Außerdem galt es, passende Werkzeuge, wie z.B. eine Quellcodeverwaltung und ein Wiki- und Bugtracking-System, einzusetzen, auf die im nächsten Abschnitt näher eingegangen wird.

Als Minimalziel galt es, sowohl einen dokumentierten Systementwurf als auch eine Implementierung eines Prototypens, mit dem plastoelastische Objektinteraktionen simuliert und visualisiert werden, zu erstellen. Einfache geometrische Objektinteraktionen sollten als Beispiele dienen, diese Grundfunktionalität zu demonstrieren.

2.2. Organisatorisches und Vorgehensweise

Die auf zwei Semester ausgelegte Projektgruppe hat mit einer Seminarphase begonnen, in der unter anderem Grundbegriffe der spanenden Fertigungsverfahren und die mathematischen Zusammenhänge einer Simulation mit der SPH-Methode oder einem Feder-Masse-Systeme erklärt wurden. Hierbei wurde der Projektgruppe aufgabenspezifisches Wissen vermittelt, welches den Projektgruppenteilnehmern eine Übersicht über die Komplexität der zu lösenden Aufgabe gab und das Projektgruppenziel konkreter zu formulieren erlaubte. Es fanden anfangs wöchentlich zwei Diskussionssitzungen statt, wovon eine später je nach Bedarf als Implementierungssitzung genutzt wurde. Nach anfänglich einseitiger Moderation der Gruppentreffen hat sich ein Rotationsprinzip etabliert, bei dem jeder Teilnehmer eine Sitzung protokolliert und die darauffolgende moderiert.

Inspiziert vom Prozessmodell SCRUM [SS12] wurden Kleingruppen gebildet, um die in Abschnitt 2.1 genannten Module zu implementieren, auf die im Kapitel 6 näher eingegangen wird. Diese Aufteilung in Untergruppen wurde stets dynamisch angepasst,

um bedarfsgerecht Kapazitäten in andere Aufgabenbereiche umzuschichten. Besonders in Hinblick auf den im zweiten Semester festgelegten Zeitplan (siehe Anhang B) ergab sich durch diese Umschichtung eine sinnvolle Verteilung von Aufgaben.

Darüber hinaus sind noch zwei wichtige online verfügbare Kommunikationsplattformen zu nennen, über die die Vorgehensweise der Projektgruppe zusätzlich koordiniert wurde:

- **Wiki mit Bugtracker:** Unter der Adresse

<https://ls7-www.cs.tu-dortmund.de/trac/chipsim>

wurde ein Wiki mit Bugtracker geführt, in dem unter anderem die oben genannten Protokolle¹ einsehbar sind. Darüber hinaus wurden die getroffenen Entscheidungen, benutzte Materialien, sowie offene Aufgaben und der Zeitplan (siehe Anhang B) dort gesammelt. Dieses mittels Projektmanagement-Werkzeug Trac[Edg12] realisierte Wiki und Bugtracker ist mit dem SVN-Repository verknüpft. Dadurch war es beim Melden eines Fehlers möglich, sich direkt auf eine Revision zu beziehen und Änderungen im Repository übersichtlich über das Wiki einzusehen.

- **IRC:** Der IRC-Channel #pgchipsim auf dem IRC-Netzwerk *irc.de.euirc.net* wurde als weitere Kommunikationsmöglichkeit eingerichtet. Benötigten die gewählten Aufgaben mehr Absprache, so wurden in den normalen Gruppentreffen weitere IRC-Treffen vereinbart.

¹Protokolle verfügbar unter <https://ls7-www.cs.tu-dortmund.de/trac/chipsim/wiki/Protokolle>

3. Grundlagen verwendeter Modelle

In diesem Kapitel werden die theoretischen Grundlagen behandelt, die während der Projektgruppendauer zur Modellierung einer Lösung verwendet wurden. Auf die wesentlichen Aspekte wird in den folgenden Abschnitten eingegangen und es erfolgt eine Erläuterung der wichtigsten Entscheidungen, die getroffen worden sind.

3.1. Spanbildung

Um die Behandlung der im Folgenden untersuchten Simulationsverfahren für spanende Fertigungsprozesse zu erleichtern, wird zunächst die Terminologie vorgestellt, die zur Beschreibung der einzelnen Komponenten eines Werkzeug-Werkstoff-Systems sowie deren räumlichen Verhältnisse zueinander dienen. Anschließend werden die verschiedenen Arten, in denen ein Werkstück bearbeitet werden kann, präsentiert.

Dabei werden die diversen Formen der Späne, die dabei entstehen können und die Ursachen dahinter betrachtet. Diese werden sowohl auf ihre groben, makroskopischen Eigenschaften als auch auf ihre feineren, teilweise mikroskopischen, Unterschiede untersucht.

Zur Orientierung werden vorab, wie in Abbildung 3.1 gezeigt, folgende Richtungen festgelegt:

- Schnittrichtung
- Vorschubrichtung.

Dabei handelt es sich bei der Schnittrichtung um die augenblickliche Vorwärtsbewegung des Werkzeugs im Schneidvorgang. Die Vorschubrichtung ist dann folglich definiert als die fortschreitende Bewegung des Werkzeugs durch den Werkstoff. Werden diese Richtungen als Vektoren betrachtet, so wird die Bewegung des Werkzeugs ebenfalls als Vektor \vec{v}_e angegeben, der sich als Summe der einzelnen Richtungen ergibt:

$$\vec{v}_e = \vec{v}_c + \vec{v}_f. \tag{3.1}$$

Dabei berechnet \vec{v}_e die sogenannte Wirkgeschwindigkeit, \vec{v}_c die Schnittgeschwindigkeit und \vec{v}_f die Vorschubgeschwindigkeit.

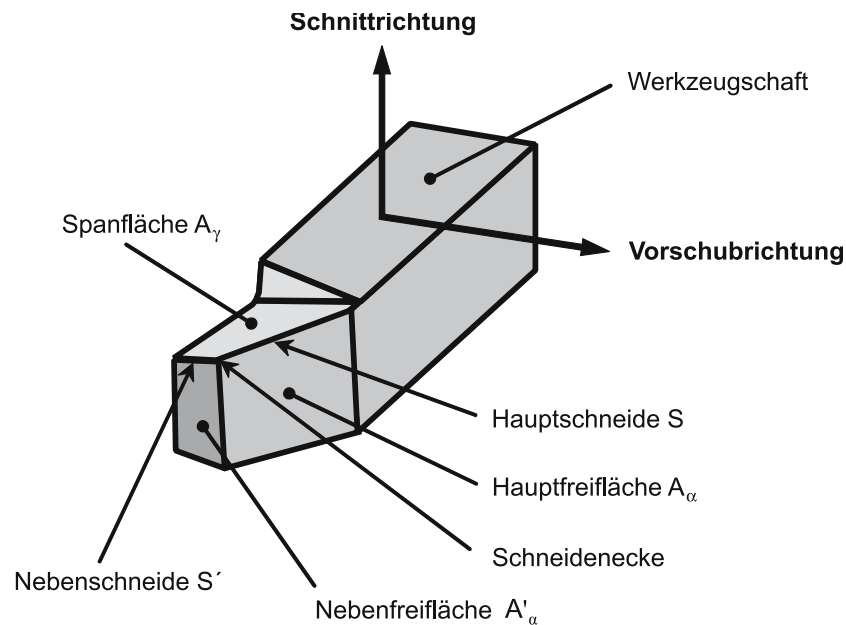


Abbildung 3.1.: Teile eines einfachen Schneidteils [KK08a]

3.1.1. Teile eines Werkzeugs

Mit der Festlegung dieser Richtungen können nun einige Aspekte eines Werkzeugs, die sich auf dessen Aufbau beziehen, betrachtet werden. Naheliegend ist die ‐Hauptschneide‐, die in Abbildung 3.1 mit S gekennzeichnet wird. Diese dient als Hauptwirkungspunkt beim Schneidvorgang, denn hier wird der gr‐o‐ste Teil des Werkstoffs getrennt.

Ein Werkzeug kann des weiteren  ber sekund re und terti re Schneiden verf gen, beispielsweise beim Bohren. Die unterste Kante des Bohrers dient als Hauptschneide und die Seiten des Bohrers wirken als sogenannte Nebenschneiden. In Abbildung 3.1 ist eine solche Nebenschneide mit der Kennzeichnung S' versehen.

Wenn diese Schneidkante etwas vom Werkstoff abtrennt, wird ein sogenannter ‐Span‐ gebildet. Dieser Span flie‐t idealerweise  ber die Spanfl che A_γ (Abb. 3.1) ab.

Die Fl chen des Werkzeugs, die mit den Haupt- bzw. Nebenschneiden verbunden sind, aber nicht bei der Abtragung des Spans mitwirken, werden als Haupt- bzw. Nebenfreifl chen bezeichnet. In Abbildung 3.1 werden diese als A_α bzw. A'_α markiert. An dieser Stelle sei erw hnt, dass per Konvention festgelegt wird, dass die Komponenten eines Werkzeugs, die Nebenrollen spielen, den gleichen Bezeichner wie die zugeh‐rige Hauptkomponente erhalten, allerdings mit einem bzw. mehreren (im Fall, dass die Komponente eine terti re Rolle spielt) zus tzlichen Apostrophen versehen werden.

Der Winkel zwischen A_γ und A_α wird β genannt (Abb. 3.2). Damit wird der Spanwinkel

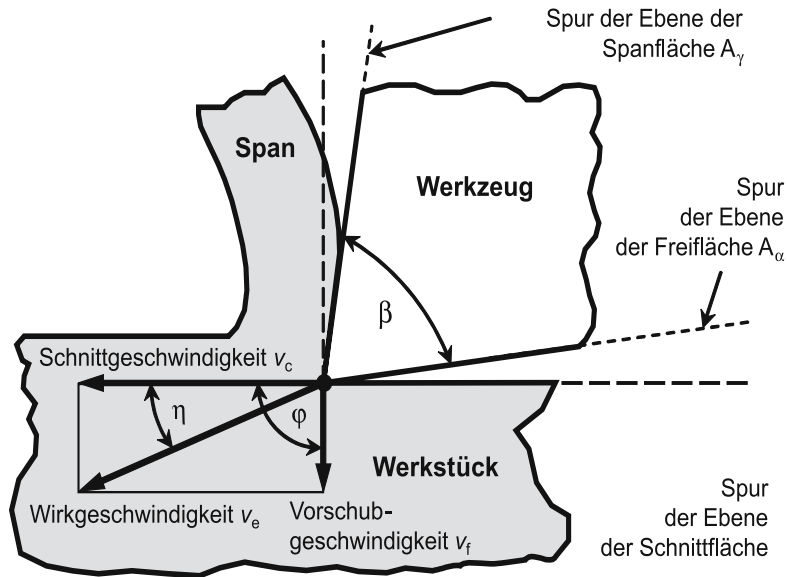


Abbildung 3.2.: Ein idealisierter Schneidkeil [KK08a]

der Hauptschneide angegeben. In der Abbildung ist auch zu erkennen, wie das Werkstück an der Kante dieser zwei Flächen getrennt wird. Dabei stellt die Abbildung einen idealisierten Fall dar. In der Realität wird diese Kante eine gewisse Krümmung haben. Diese Krümmung wird als Schneidkantenradius r_β bezeichnet und in Abbildung 3.3 dargestellt.

3.1.2. Bezugssysteme

Um die Ausrichtung eines Werkzeugs zu beschreiben, werden Bezugssysteme definiert. Je nach dem, ob das Werkzeug oder seine Bewegung über ein Werkstück beschrieben wird, wird entweder das Werkzeug-Bezugssystem oder das Wirk-Bezugssystem verwendet.

Werkzeug-Bezugssystem

Merkmal des Werkzeug-Bezugssystems ist, dass die dadurch festgelegten Ebenen orthogonal zum Werkzeug sind. Da die Verfahrenskinetik in diesem Bezugssystem nicht betrachtet wird, dient es nur zur Spezifikation eines Werkzeugs. Dabei wird die Werkzeug-Bezugsebene P_r nur von der Schnitttrichtung abhängig festgelegt und zwar senkrecht dazu. Die Arbeitsebene P_f und die Werkzeug-Rückebene P_p werden dann abhängig von P_r und dem Werkzeug festgelegt. Dabei wird P_f als senkrecht zu P_r und parallel zu \vec{v}_f definiert, P_p wird dann als senkrecht zu P_r und P_f festgelegt (Abb. 3.4).

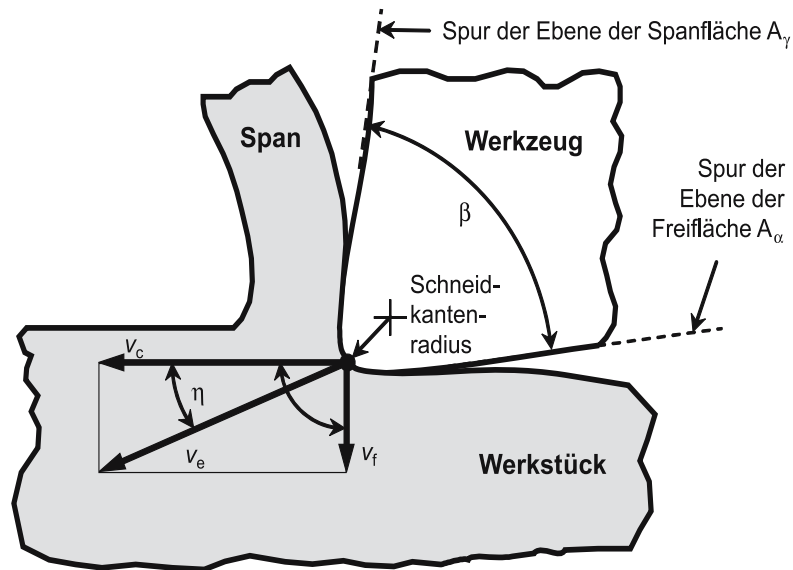


Abbildung 3.3.: Schneide mit Schneidkantenradius [KK08a]

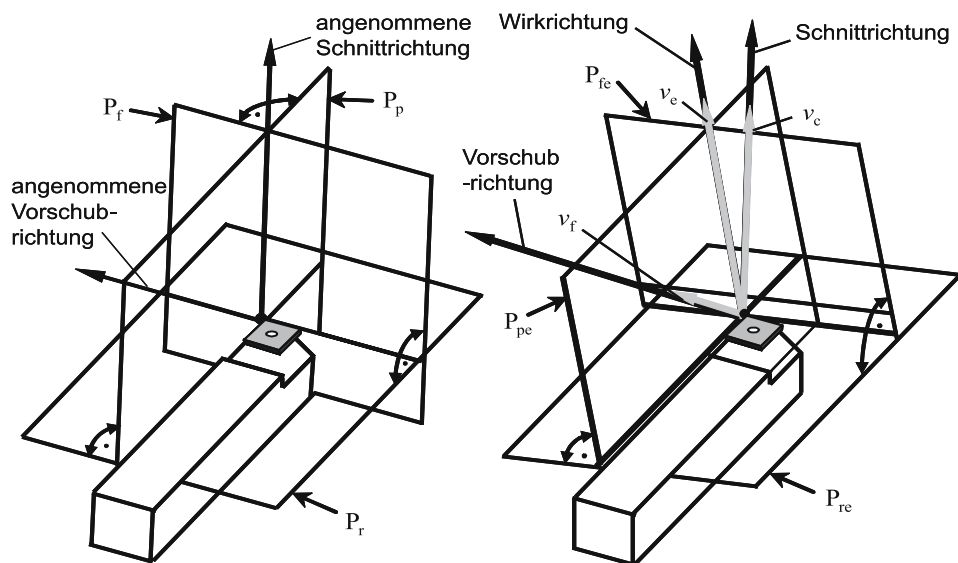


Abbildung 3.4.: a): Werkzeug-Bezugssystem, b): Wirk-Bezugssystem [KK08a]

Wirk-Bezugssystem

Im Wirk-Bezugssystem wird die Verfahrenskinetik mit einbezogen, jedoch gibt es zu den Ebenen im Werkzeug-Bezugssystem Analogien, die auch entsprechend genannt werden. Um die Mitwirkung der Verfahrenskinetik darzustellen, wird das Werkzeug-Bezugssystem um den Winkel η (s. Abb. 3.2) gedreht. Die Namensgebung wird auch analog durchgeführt indem die Indizes der Ebenen um den Buchstaben e erweitert werden, wobei das e aus dem englischen "effective" hergeleitet wird.

3.1.3. Verfahrensvarianten

Mit diesem Rahmenwerk von Begriffen können nun die verschiedenen Schnittverfahren beschrieben werden, die von der Ausrichtung des Werkzeugs in Relation zum Werkstück abhängig sind. Der Allgemeinfall eines Schnittvorgangs ist der sogenannte gebundene, schräge Schnitt (Abb. 3.5). In diesem Fall ist weder der Werkzeug-Einstellwinkel κ_r noch der Werkzeug-Neigungswinkel λ_s festgelegt und es dürfen auch Nebenschneiden eine Rolle bei der Trennung des Werkstoffs spielen. Diese Bezeichnung läßt sich dadurch ableiten, dass die Ausrichtung des Werkzeugs in Relation zum Werkstück uneingeschränkt, also schräg, ist und dass die Mitwirkung der Nebenschneiden die Bewegungsfreiheit des Werkzeugs in mindestens eine Richtung einschränkt, so dass das Werkzeug am Platz gebunden ist. Ein anschauliches Beispiel für den gebundenen, schrägen Schnitt ist der Schneeräumer, der mit einem zur Fahrtrichtung schrägen Hobel den Schnee zur Seite schiebt.

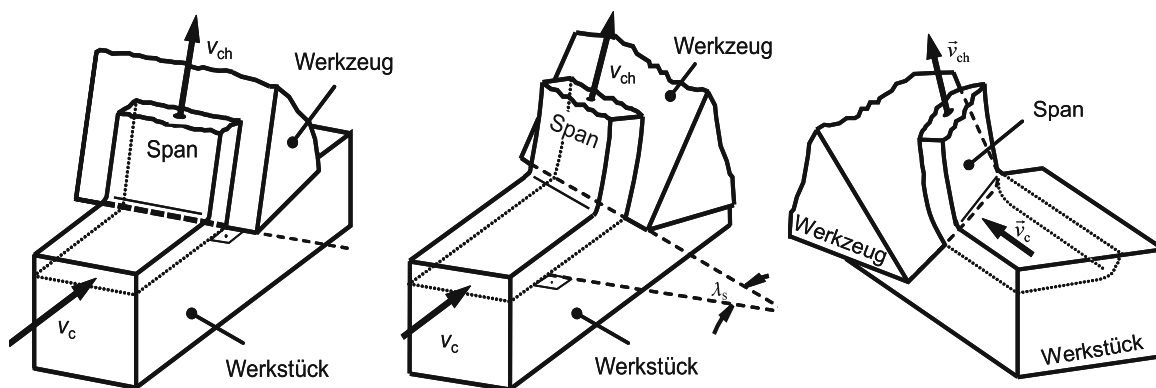


Abbildung 3.5.: a) Der freie, orthogonale Schnitt, b) Der freie, schräge Schnitt, c) Der gebundene, schräge Schnitt [KK08a]

Soll festgelegt werden, dass keine Nebenschneiden zum Einsatz kommen, so wird der Schnitt "frei" genannt. Dabei bleibt die sonstige Ausrichtung des Werkzeugs uneingeschränkt, also handelt es sich um einen freien, schrägen Schnitt.

Werden ferner die Winkel $\lambda_s = 0^\circ$ und $\kappa_r = 90^\circ$ festgelegt, so liegt das Werkzeug orthogonal zum Werkstück. Dieses wird als freier, orthogonaler Schnitt bezeichnet. Hier dient ein normaler Holzhobel als Beispiel.

Diese Namenskonvention kann um einen Parameter erweitert werden, nämlich ob der ganze Schneidvorgang in einem durchgehenden Schnitt erfolgt, z.B. beim Drehen eines Werkstücks, oder ob das Werkzeug für mehrere einzelne Schnitte eingesetzt wird, z.B. beim Gebrauch eines Holzhobels. Der erste Fall ist ein ununterbrochener Schnitt, der letzte hingegen ein unterbrochener Schnitt.

3.1.4. Spanbildung

Bei der Bearbeitung eines Werkstücks mit einem Werkzeug wird das Werkstück verformt. Falls die Bearbeitung unter richtigen Konditionen stattfindet, kann Material abgetrennt werden und über die Spanfläche des Werkzeugs fließen. In diesem Fall handelt es sich um Spanbildung.

Ob ein Schneidvorgang zur Spanbildung führt, hängt von diversen Faktoren ab. Dabei sind neben den Kräften, die im Prozess mitwirken, z. B. auch die Materialparameter und die Geometrie des Werkzeugs wichtig.

Materialabhängiges Verhalten

Abbildung 3.6 zeigt, wie der Werkstoff sich unter Bearbeitung mit einem Werkzeug verhalten kann. Zuerst werden zwei Fälle, die sich an den Materialeigenschaften des Werkstoffs unterscheiden, betrachtet.

Duktiler Werkstoff Im ersten Fall wird angenommen, dass das Material duktil ist. Somit lässt sich das Werkstück leicht plastisch verformen. Diese Verformung findet in Bereich (a) (Abb. 3.6) statt. Durch die Flexibilität des Werkstoffs wird das Material nicht abgetrennt. Das Werkzeug dringt weiter vorwärts durch den Werkstoff, der sich durch die entstehende Spannung in Bereich (e) trennen lässt. Durch diesen fortlaufenden Vorgang wird ein immer größeres Stück des Werkstoffs vom Werkstück getrennt, das dann über die Spanfläche des Werkzeugs fließt. Somit entsteht ein Span.

Spröder Werkstoff Falls der Werkstoff eher spröde ist, kann die Trennung des Materials auf eine etwas andere Art stattfinden. In diesem Fall kann eine leichte Verformung des Werkstoffs in (a) (s. Abb. 3.6) zu einer Bruchdehnung führen. Das getrennte Material bricht bereits vor der Hauptschneide ab und zerbröckelt. Dadurch entsteht kein Span, sondern lediglich Werkstückfragmente. Die Bruchdehnung gibt an wie weit sich das Material nach einem Bruch dehnt, im Vergleich zur ursprünglichen Länge des Materials.

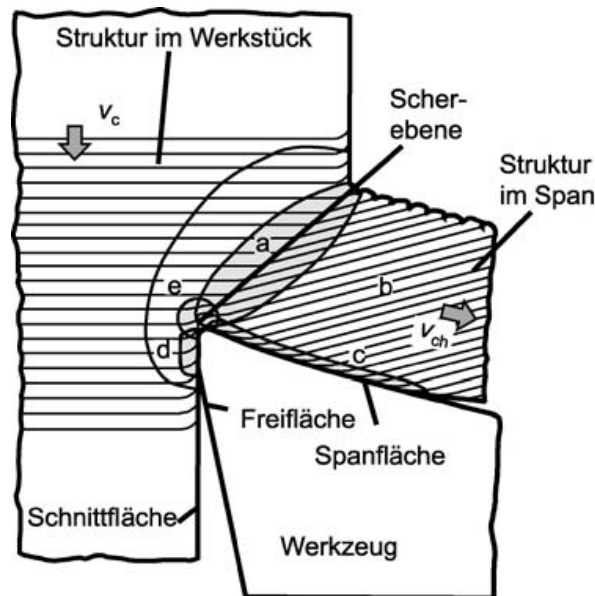


Abbildung 3.6.: Entstehung eines Spans [KK08a]

3.1.5. Spanform

Nicht nur die Entstehung eines Spans kann durch die Materialparameter des Werkstoffs beeinflusst werden, sondern auch die Form des entstehenden Spans. Dabei können andere Faktoren wie z. B. die Schnitt- und Vorschubgeschwindigkeit des Werkzeugs sowie der Werkzeug-Normalspanwinkel γ_n (s. Abb. 3.7), der Werkzeug-Einstellwinkel und der Werkzeug-Neigungswinkel eine entscheidende Rolle spielen.

Makroskopische Eigenschaften

Da manche Formen von Spänen günstiger sind als andere, ist es von Vorteil, dass deren Form durch Änderung der oben genannten Parameter beeinflusst werden kann. Eine Betrachtung einiger der in Abbildung 3.8 dargestellten Formen findet im Folgenden statt.

Für die maschinelle Bearbeitung eines Werkstücks sind die in (3) und (5) abgebildeten Spanformen besonders ungünstig. Bei (5), dem sogenannten langen Wendelspan, wird der automatisierte Abtransport erschwert. Dadurch wird ein häufiges Eingreifen eines Mitarbeiters benötigt, um den Schneidvorgang fortzusetzen. Dies kostet natürlich Zeit und erhöht dadurch die Fertigungskosten. Die in (3) dargestellten Flachwendelspäne können leicht zwischen Werkzeug und Werkstück rutschen und Schaden bei einem oder beiden verursachen. In (1) sind Bandspäne abgebildet. Diese Art von Spänen stellen zusammen mit den Wirrspänen (2) und den Bröckelspänen (10) eine besondere Gefahr

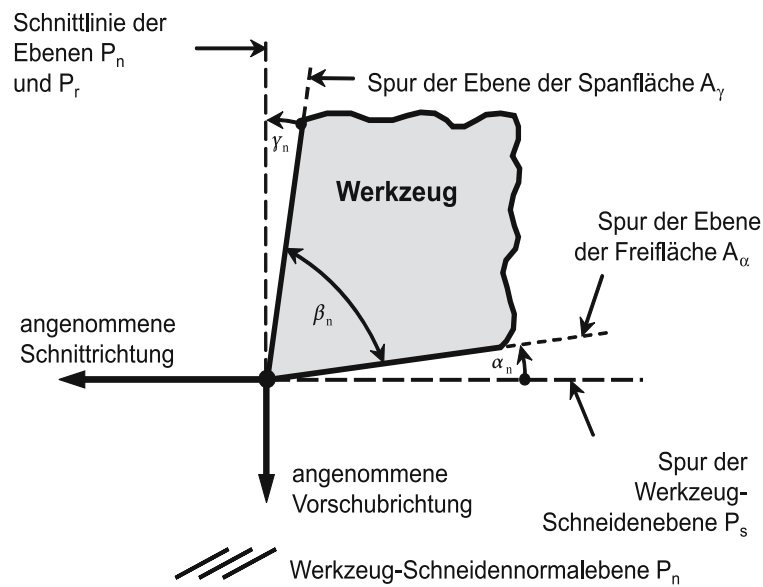


Abbildung 3.7.: Der Werkzeug-Normalspanwinkel [KK08a]

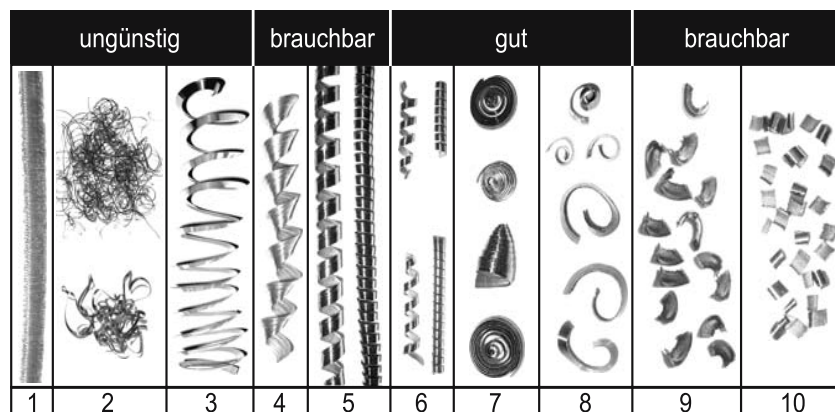


Abbildung 3.8.: Spanformen, [KK08a]

für das Personal dar.

Mikroskopische Eigenschaften

Nicht nur die grobe Form der Späne kann beeinflusst werden, sondern auch die mikroskopische Eigenschaften des abgetrennten Materials. Diese Eigenschaften haben natürlicherweise einen Einfluss auf die resultierende Oberfläche des bearbeiteten Werkstücks. Davon hängt auch die Menge der benötigten Nachbearbeitungsschnitte und die entsprechenden

Fertigungskosten ab. Die in Abbildung 3.9 dargestellten Formen werden betrachtet.

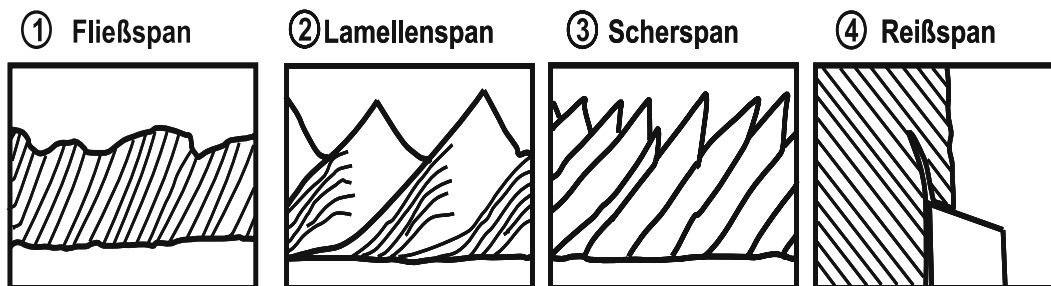


Abbildung 3.9.: Späne - strukturelle Eigenschaften, [KK08a]

Die mit (1) gekennzeichnete Spanform zeichnet sich durch ihre glatte, regelmäßige Struktur aus. Diese Art von Span entsteht, wenn gleichmäßige Reibungsverhältnisse zwischen Span und Werkzeug herrschen.

In Abbildung 3.9 (2) ist der sogenannte Lamellenspan abgebildet. Um so eine Spanform zu erzeugen, müssen zeitlich stark veränderliche Reibungsverhältnisse zwischen Werkzeug und Werkstück präsent sein. Die Frequenz der auftretenden Spitzen können im kHz Bereich relativ zur Schnittgeschwindigkeit liegen. Dabei ist die Amplitude der Spitzen eher gering.

Der Scherspan (3) ist ein Sonderfall der Lamellenspanbildung, wobei eine Rissbildung stattfindet. Durch die hohen Temperaturen und den hohen Druck, die im Fertigungsprozess herrschen, werden diese Risse aber wieder zusammengeschweißt und bleiben somit intakt.

Zuletzt wird der Reißspan betrachtet. In diesem Fall findet keine plastische Verformung statt, sondern es brechen Teile vom Werkstück ab. Dieses Verhalten ist bei sehr spröden Werkstoffen zu erwarten.

3.2. Kontaktmechanik

Die Kontaktmechanik wurde als Referenzlösung gewählt, mit der ein Vergleich zur SPH- und MCA-Lösung, zum Kontaktzeitpunkt möglich ist. Hierzu wurde [Pop09, Kapitel 2 und 5] als Quelle genutzt.

Bei den folgenden Berechnungen werden nicht alle zwischen zwei Körpern herrschenden Kräfte berücksichtigt. So wird die Auswirkung der Adhäsion genauso wenig betrachtet wie die der Kapillarkräfte. Reiner Normalkontakt zwischen zwei Objekten steht in den folgenden Abschnitten im Fokus, obwohl die in der Realität wirkenden Kräfte leichte Abweichungen liefern. Da das Ziel der PG keine mathematisch absolut korrekte, sondern

eine mit der Realität vergleichbare Lösung ist, sind die genannten Einschränkungen bei der Referenzlösung akzeptabel.

Bei der Erarbeitung einer Referenzlösung wurden einfache geometrische Objektkombinationen betrachtet. Hierzu zählen der einfachste Fall, bei dem eine Kugel auf einer Ebene liegt. Der für die Spansimulation interessanteste Fall ist ein Kegel der in eine Ebene drückt.

In den folgenden genauer vorgestellten Objektkombinationen werden immer zwei Objekte betrachtet: Objekt₁ und Objekt₂. Für diese sind die folgenden Materialparameter notwendig:

- Elastizitätsmodul (E_i)
- Querdehnungszahl oder auch Poissonzahl (ν_i)
- Dichte ρ_i .

	Elastizitätsmodul in $\frac{\text{kg}}{\text{m} \cdot \text{s}^2}$:	Querdehnungszahl:	Dichte in $\frac{\text{kg}}{\text{m}^3}$:
Stahl:	$2,10 \cdot 10^{11}$	0,285	7840
Gummi:	$3,05 \cdot 10^6$	0,5	3265

Tabelle 3.1.: Die Materialparameter sind folgender Materiparameterdatenbank entnommen: <http://www.matweb.com/>

Der Gesamtelastizitätsmodul der beiden Objekte berechnet sich durch [Pop09, 5.26]:

$$E^* = \frac{E_1 \cdot E_2}{(1 - \nu_1^2) \cdot E_2 + (1 - \nu_2^2) \cdot E_1}. \quad (3.2)$$

Eine einheitliche Formel zur Druck- oder Kraftberechnung gibt es nicht, da die zu verwendenden Formeln abhängig von der Geometrie der einzelnen Körper sind, weshalb die jeweilig verwendete Formel in dem zugehörigen Abschnitt zu finden ist.

3.2.1. Kontakt zwischen Kugel und Ebene

Befindet sich eine elastische Kugel auf einer elastischen Ebene (Abb. 3.10), so ist der Radius der Kontaktfläche beider Objekte $a = \sqrt{R \cdot d}$, wobei d die zu berechnende Eindringtiefe und R der vorher gewählte Kugelradius ist.

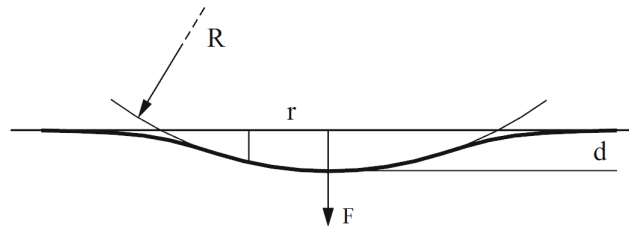


Abbildung 3.10.: Kontakt zwischen Kugel und Ebene
[Pop09, Abb. 5.3]

Der Druck an einzelnen Punkten wird mit

$$p(r) = p_0 \cdot \sqrt{1 - \frac{r^2}{a^2}} \quad (3.3)$$

bestimmt. Hierbei sei r der Abstand vom Mittelpunkt der Kontaktfläche mit $0 \leq r \leq a$ und p_0 ist der maximale Druck:

$$p_0 = \frac{2}{\pi} \cdot E^* \cdot \sqrt{\frac{d}{R}}. \quad (3.4)$$

Die Kraftgleichung:

$$F = \frac{4}{3} \cdot E^* \cdot \sqrt{R} \cdot d^{\frac{3}{2}} \quad (3.5)$$

wird gleich der Gravitationskraft der Kugel gesetzt und nach d aufgelöst.

3.2.2. Kontakt zwischen Kegel und Ebene

Das Eindringen einer Werkzeugschneide in ein Werkstück bei der Spanbildung lässt sich am besten mit einem Kegel, der auf eine ebene Fläche drückt, vergleichen (Abb. 3.11).

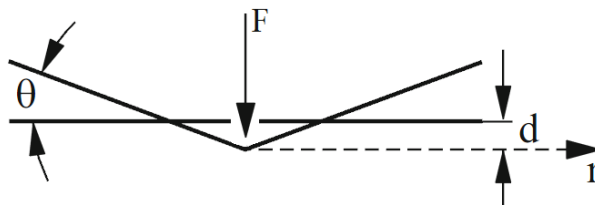


Abbildung 3.11.: Kegel auf Ebene [Pop09, Abb. 2.10]

Wird die Spitze eines Kegels durch sein Eigengewicht in eine Ebene gedrückt, so lässt

sich die Eindringtiefe d mit:

$$d = \frac{\pi \cdot a \cdot \tan(\theta)}{2} \quad (3.6)$$

berechnen, wobei θ der Winkel zwischen Kegel und Ebene ist. Für die Berechnung der Druckverteilung wird:

$$p(r) = -\frac{E \cdot d}{\pi \cdot a \cdot (1 - v^2)} \cdot \ln \left(\frac{a}{r} + \sqrt{\frac{a^2}{r^2} - 1} \right) \quad (3.7)$$

verwendet. Die einwirkende Kraft wird mit:

$$F = \frac{2 \cdot \tan(\theta) \cdot E \cdot d^2}{\pi} \quad (3.8)$$

berechnet.

3.3. Smoothed Particle Hydrodynamics

SPH ist eine Methode aus dem Umfeld der Hydrodynamik und wird dort erfolgreich für die Simulation verschiedener Flüssigkeiten verwendet. Ebenso ist dieses Verfahren zur Simulation von Gasen verwendbar, insbesondere in der Astrophysik ([GM77], [Luc77]).

Das Verhalten von zusammenhängender Materie wird oftmals als das Verhalten eines Kontinuums betrachtet und mit Differentialgleichungen beschrieben, die dieses Kontinuum modellieren [KK08b]. Mit dem SPH-Verfahren werden diese approximiert.

Anschaulich betrachtet ist die Grundidee von SPH, ein Kontinuum durch eine endliche Anzahl von Partikeln zu beschreiben. Diese sind nicht mit realen Teilchen wie Atomen oder Molekülen zu verwechseln, sondern stellen eher Stützstellen dar, die das Kontinuum näherungsweise beschreiben sollen. Dennoch ähneln die Partikel der SPH-Methode echten Teilchen insofern, als dass sie sich bewegen können und Eigenschaften aufweisen, die physikalischen Größen entsprechen und ihre Interaktion beeinflussen.

Jedes SPH-Partikel stellt einen Interpolationspunkt für die Berechnungen dar und hat Eigenschaften wie zum Beispiel Temperatur, Geschwindigkeit und Dichte. Wenn sich mehrere in einer gemeinsamen Nachbarschaft befinden, interagieren sie miteinander. Diese Nachbarschaft wird charakterisiert durch die Glättungslänge h , welche den Radius dieser Nachbarschaften darstellt. Interaktionen zwischen Partikeln i und j werden durch eine sogenannte Glättungsfunktion W berechnet, wobei der Abstandsvektor $\vec{x}_{ij} = \vec{x}_j - \vec{x}_i = \vec{r}$ zwischen ihnen eine Rolle spielt und wie im Folgenden beschrieben in die Berechnungen der Glättungsfunktionen mit eingeht.

3.3.1. Glättungsfunktion

Die Glättungsfunktion bestimmt den Grad der Interaktion zwischen Partikeln, indem sie die Werte über die Interpolationspunkte glättet, wie in Abbildung 3.12 veranschaulicht wird. Die Wahl der Glättungsfunktion W ist abhängig von dem zu lösenden Problem. Es

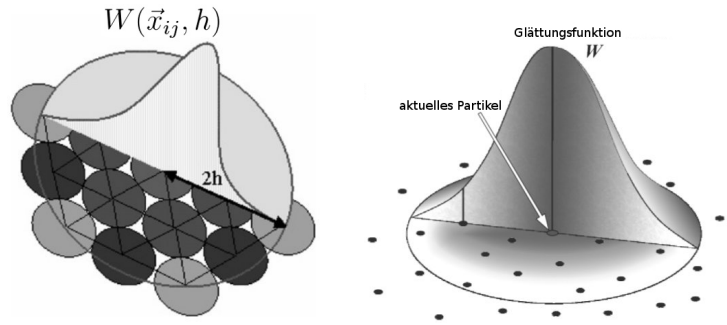


Abbildung 3.12.: Glättung mittels Glättungsfunktion (Quellen: [VF08] und [LESL07])

ist durchaus zulässig, für unterschiedliche Probleme innerhalb eines SPH-Modells verschiedene Glättungsfunktionen für verschiedene zu berechnende Funktionen zu verwenden. Es ist notwendig, Funktionen zu wählen, welche besonders geeignete Ableitungen, Gradienten oder Laplace-Operatoren besitzen. Was unter einer guten Eignung zu verstehen ist, hängt von dem jeweils zu lösenden Problem ab.

Ein Beispiel für eine Glättungsfunktion ist die folgende kubische B-Spline Funktion (siehe Abbildung 3.13):

$$W(\vec{x}_{ij}) = C \cdot \begin{cases} 1 - \frac{3}{2} \|\vec{x}_{ij}\|^2 + \frac{3}{4} \|\vec{x}_{ij}\|^3 & \text{für } \|\vec{x}_{ij}\| \leq 1 \\ \frac{1}{4} (2 - \|\vec{x}_{ij}\|)^3 & \text{für } 1 < \|\vec{x}_{ij}\| \leq 2 \\ 0 & \text{für } \|\vec{x}_{ij}\| > 2 \end{cases} \quad (3.9)$$

Hierbei ist die Glättungslänge h 2 und C ist eine dimensionsabhängige Normierungskonstante.

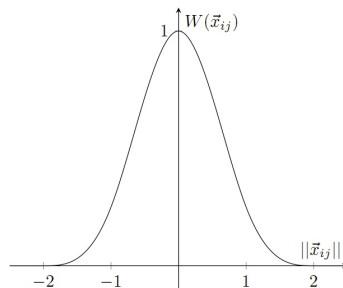


Abbildung 3.13.: Plot des kubischen B-Spline mit $h = 2$

Eigenschaften Eine Glättungsfunktion W weist die folgenden drei Eigenschaften auf:

1. Normierungseigenschaft:

$$\int_{\Omega} W(\vec{x}_{ij}, h) dx' = 1 \quad (3.10)$$

2. Delta-Eigenschaft:

$$\lim_{h \rightarrow 0} W(\vec{x}_{ij}, h) = \delta(\vec{x}_{ij}) \quad (\text{Dirac Funktion}) \quad (3.11)$$

3. Kompakter Träger:

$$W(\vec{x}_{ij}, h) = 0, \text{ falls } \|\vec{x}_{ij}\| > h. \quad (3.12)$$

Als Glättungsfunktion stehen unter anderem $W_{poly6}(\vec{x}_{ij}, h)$ (siehe Abbildung 3.14) als Standardglättungsfunktion [Sol10, 2.20] und die Glättungsfunktion $W_{spiky}(\vec{x}_{ij}, h)$ (siehe Abbildung 3.15) [Sol10, 2.21] für die Druckberechnung zur Verfügung. Die Standardglättungsfunktion wird bei der Berechnung der Dichte verwendet:

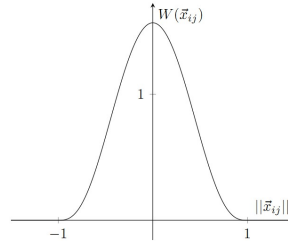


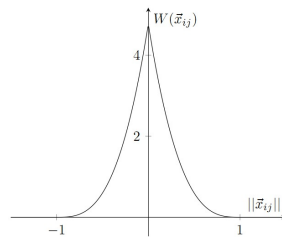
Abbildung 3.14.: Plot der Standardglättungsfunktion poly6 mit $h = 1$

$$W_{poly6}(\vec{x}_{ij}, h) = \frac{315}{64 \cdot \pi \cdot h^9} \cdot (h^2 - \|\vec{x}_{ij}\|^2)^3 \quad (3.13)$$

Die Spiky-Glättungsfunktion findet bei der Berechnung des Drucks Verwendung und wird mit

$$W_{spiky}(\vec{x}_{ij}, h) = \frac{15}{\pi \cdot h^6} \cdot (h - \|\vec{x}_{ij}\|)^3 \quad (3.14)$$

berechnet.

Abbildung 3.15.: Plot der Glättungsfunktion Spiky mit $h = 1$

3.3.2. Simulation mit SPH

Für eine Simulation mittels eines SPH-Modells sind im Allgemeinen folgende Schritte notwendig:

1. Initialisierung
 - Platzierung der Partikel gemäß Verteilung
 - Festlegung der initialen Partikeleigenschaften
 - Berechnung der abgeleiteten Eigenschaften
2. Nachbarschaftssuche
3. Berechnung der Zustandsgrößen
 - Berechnung der Massedichte
 - Berechnung anderer Zustandsgrößen (z.B. Druck)
4. Kraftberechnung
 - Berechnung der internen Kräfte
 - Berechnung der externen Kräfte
5. Zeitintegration
 - Berechnung der Beschleunigung/Geschwindigkeit der Partikel
 - Berechnung der Positionsveränderungen
 - (optional): Kollisionsbehandlung
6. Partikelupdate

Nachdem eine Simulation initialisiert ist, werden Schritte 2 bis 6 mit den aus dem vorherigen Durchgang erhaltenen Werten wiederholt. Die konkreten Vorgänge der einzelnen Simulationsschritte werden in Abschnitt 6.5 erläutert. Je nachdem welche genauen Größen auf welche Weise in den einzelnen Schritten berechnet werden, lassen sich mit dieser

Vorgehensweise verschiedene Einsatzgebiete von SPH abdecken (z.B. Fluidsimulation und Astrophysik).

Schritte, die nicht voneinander abhängen, können in ihrer Reihenfolge vertauscht oder parallel ausgeführt werden. Insbesondere die Berechnung der Zustandsgrößen sowie der internen und externen Kräfte muss nicht sequentiell erfolgen, jedoch sind voneinander abhängige Größen zwingend sequentiell zu berechnen. Beispielsweise muss die Masseberechnung vor der Gravitationskraftberechnung erfolgen, da die Gravitationskraft abhängig von der Masse ist. Zusammen bilden diese Berechnungen eine diskrete Approximation der Kontinuumsmechanik. Wie diese Diskretisierung und Berechnungen erfolgen, wird im Folgenden beschrieben.

Diskretisierung Werden die SPH-Partikel als Interpolationsstützpunkte aufgefasst, so lässt sich die SPH-Funktionsapproximation folgendermaßen formulieren:

$$\langle f(x_i) \rangle \approx \sum_{j=1}^N \frac{m_j}{\rho_j} \cdot f(x_j) \cdot W(\vec{x}_{ij}, h). \quad (3.15)$$

Hierbei wird jedes der N SPH-Partikel mit einem Gewicht versehen, was abhängig ist von seiner Masse m und seiner Dichte ρ .

Anhand einiger Beispiele wird verdeutlicht, wie verschiedene Größen und Kräfte mit dem SPH-Formalismus approximiert werden. Die angeführten Beispiele stellen keinen kompletten Berechnungszyklus dar, sondern sind lediglich Ausschnitte aus diesem.

Dichteberechnung Der erste Schritt, in dem eine SPH-Approximation angewandt wird, ist die Berechnung der Zustandsgrößen. In diesem Schritt wird zunächst die Dichte berechnet, da sie für die darauf folgende Druckberechnung benötigt wird.

Für die Dichteberechnung von Partikel i wird die bereits vorgestellte Glättungsfunktion $W_{poly6}(\vec{x}_{ij}, h)$ mit der folgenden masseabhängigen Formel verwendet [Sol10, 2.13]:

$$\rho_i = \sum_j m_j \cdot W_{poly6}(\vec{x}_{ij}, h). \quad (3.16)$$

Bei dieser Berechnung sind die Dichtewerte der Partikel am Rand eines Objektes geringer als die Dichtewerte von Partikeln im Inneren des Objektes. Dieses Phänomen kann verhindert werden, wenn die Dichte der Partikel mit

$$\frac{d\rho_i}{dt} = \sum_j m_j \cdot (\vec{v}_i - \vec{v}_j) \cdot W_{poly6}(\vec{x}_{ij}, h) \quad (3.17)$$

berechnet wird [Sol10, 2.14]. Allerdings können hierdurch bei längeren Simulationen Instabilitäten entstehen, weshalb die Dichte auch mit einer alternativen Formel berechnet werden kann (siehe [Sol10]).

Druckberechnung Für Gase lässt sich der Druck p an einem Partikel einfach berechnen:

$$p_i = k \cdot \rho_i. \quad (3.18)$$

Die Konstante k ist abhängig vom jeweiligen Gas und ρ_i ist die Dichte des Partikels i . Für eine Realisierung von anziehender und abstoßender Druckkraft muss die Gleichung modifiziert werden:

$$p_i = k \cdot (\rho_i - \rho_0). \quad (3.19)$$

Die Ruhedichte ρ_0 legt dabei fest, bei welcher Massedichte keine anziehenden oder abstoßenden Kräfte wirken. Veranschaulicht ist das Ganze in Abbildung 3.16 zu sehen.

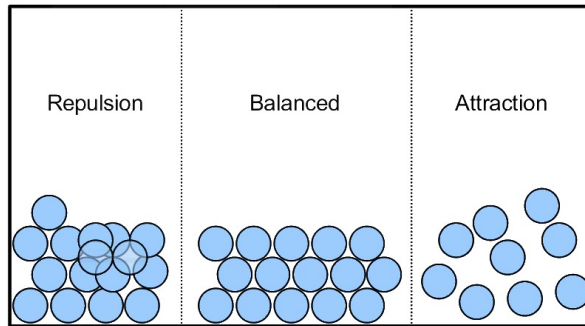


Abbildung 3.16.: Druck- und Abstoßungsverhältnisse (Quelle: [Sta10])

Eine alternative Formel für die Berechnung des Drucks, welche für bestimmte Anwendungsfälle bessere Ergebnisse liefern kann, ist:

$$p_i = \frac{k \cdot \rho_0}{\gamma} \cdot \left(\left(\frac{\rho_i}{\rho_0} \right)^\gamma - 1 \right). \quad (3.20)$$

Hierbei sind k die Steifigkeit und ρ_0 die Referenzdichte des Materials. Der Koeffizient γ hat üblicherweise den Wert 7 [Sol10, 2.15, Seite 16].

Druckkraftberechnung Nach der Berechnung der Zustandsgrößen folgt die Berechnung der internen Kräfte. Ein Beispiel für eine durch SPH zu errechnende interne Kraft ist die Kraft, die aus dem Druck innerhalb des Materials resultiert. Hoher Druck an einem Partikel führt zu einer Kraft, die umliegende Partikel abstoßt. Die Druckkraft eines Partikels wird, aufbauend auf dem berechneten Druck, mit folgender Formel berechnet:

$$\vec{F}_i^{Druck} = -\frac{m_i}{\rho_i} \cdot \sum_j \frac{m_j}{\rho_j} \cdot \frac{p_i + p_j}{2} \cdot \nabla W_{Spiky}(\vec{x}_{ij}, h). \quad (3.21)$$

Hierbei verwendet Solenthaler [Sol10] die Spiky-Glättungsfunktion aus Abbildung 3.15.

Viskositätskraftberechnung Ein weiteres Beispiel für eine bei der Simulation zu berechnende interne Kraft ist die Viskositätskraft. Bei der Berechnung der Viskositätskraft, die auf ein Partikel wirkt, wird die Formel:

$$\vec{F}_i^{Viskos} = \frac{m_i}{\rho_i} \cdot \sum_j \frac{\mu_i + \mu_j}{2} \cdot \frac{m_j}{\rho_j} \cdot (\vec{v}_j - \vec{v}_i) \cdot \nabla^2 W_{poly6}(\vec{x}_{ij}, h) \quad (3.22)$$

verwendet [Sol10, 2.19]. Hierbei sind μ_i und μ_j die Viskositätskonstanten und v_i und v_j die Geschwindigkeit der Partikel. Als Glättungsfunktion wird poly6 (s. Abb. 3.14) gewählt.

Gravitationskraftberechnung Nach der Berechnung der internen Kräfte werden die externen Kräfte berechnet. Ein Beispiel für eine derartige externe Kraft ist die Gravitationskraft.

Die Gravitationskraft, die auf ein Partikel wirkt, wird mit

$$\vec{F}_i^{Grav} = m_i \cdot \vec{g} \quad (3.23)$$

bestimmt und zu der Summe aller auf das Partikel wirkenden Kräfte addiert.

Wie an der Formel ersichtlich, wurde hierbei die SPH-Approximation nicht verwendet. Externe Kräfte wirken von außen auf die einzelnen Partikel und stehen in keiner Wechselwirkung zwischen einzelnen Partikeln. Der SPH-Formalismus findet deshalb bei externen Kräften keine Anwendung.

3.3.3. PCISPH

Problematisch bei der Simulation mit SPH sind Umgebungen mit sehr hohem Druck, die durch Kompression der Partikel entstehen, da sie künstliche Oszillationen und numerische Instabilität verursachen können. Um dieses Problem zu beseitigen, gibt es diverse Ansätze. Der PCISPH-Ansatz (Predictive-Corrective incompressible SPH) stellt eine Möglichkeit dar.

Die Grundidee der PCISPH-Methode ist, die mit den Positionsveränderungen der Partikel einhergehenden Dichteveränderungen vorauszusagen und die Abweichung von der Referenzdichte zu berechnen. Diese Abweichung wird benutzt, um den Druck neu zu berechnen. Dies wird solange wiederholt, bis die Abweichung von der Referenzdichte einen Schwellwert unterschreitet oder eine feste Anzahl an Iterationen durchlaufen wurde. Der Algorithmus zur Durchführung dieser Berechnungen ist im Folgenden aufgeführt.

```

while Simulation läuft do
  for alle Partikel i do
    Berechne Nachbarpartikel;
    Berechne Kräfte;
    Initialisiere den Druck mit 0;
    Initialisiere die Druckkraft mit dem Nullvektor;
    Setze  $k = 0$ 
  end
  while  $\max(\rho_{err_i}^*) > \eta$  oder  $k < maxIteration$  do
    for alle Partikel i do
      Berechne Geschwindigkeit  $v_i^*(t + \Delta t)$  voraus;
      Berechne Position  $x_i^*(t + \Delta t)$  voraus;
    end
    for alle Partikel i do
      Berechne Nachbarschaft neu;
      Berechne Dichte  $\rho_i^*(t + \Delta t)$  voraus;
      Berechne Dichteänderung  $\rho_{err_i}^*(t + \Delta t)$  ;
      Berechne Druck  $p_i(t) + = \delta\rho_{err_i}^*(t + \Delta t)$  neu;
    end
    for alle Partikel i do
      Berechne Druckkraft  $\mathbf{F}_i^p(t)$ ;
    end
    Setze  $k = k + 1$ ;
  end
  for alle Partikel i do
    Berechne neue Geschwindigkeit  $v_i(t + \Delta t)$ ;
    Berechne neue Position  $x_i(t + \Delta t)$ ;
  end
end

```

Algorithmus 1: Pseudocode für PCISPH

Üblicherweise werden nur wenige Prozent Abweichung von ρ_0 gewünscht. Als alternative Abbruchbedingung kann eine maximal Anzahl von Iterationen *maxIterations* angegeben werden, nach denen die Berechnung, unabhängig davon, ob die gewünschte Abweichung η unterschritten wurde, abgebrochen wird.

Die vorausgesagte Dichte wird mit

$$\rho_i^*(t + \Delta t) = \sum_j m_j \cdot W(\vec{x}_{ij}, h)$$

berechnet. Die Abweichung von der Referenzdichte ρ_0 beträgt damit $\rho_{err_i}^*(t + \Delta t) = \rho_i^*(t + \Delta t) - \rho_0$.

Für die Aktualisierung des Drucks $p_i(t) := \delta \cdot \rho_{err_i}^*(t + \Delta t) + p_i(t)$ wird der Wert δ vorberechnet durch:

$$\delta = -(\beta \cdot (-\sum_j \nabla W_{ij}^0 \cdot \sum_j \nabla W_{ij}^0 - \sum_j (\nabla W_{ij}^0 \cdot \nabla W_{ij}^0)))^{-1}, \quad (3.24)$$

mit $\beta = 2 \cdot (\frac{m_i \cdot \Delta t}{\rho_0})^2$. Es wird dabei angenommen, dass die Masse m_i für alle Partikel gleich ist. Die Berechnung von $W_{ij}^0 = W(\mathbf{x}_{ij}^0, h)$ erfolgt auf Basis der initialen Position \mathbf{x}_i^0 eines prototypischen Partikels i mit Nachbarschaft j .

Die Formel mit der nach der Korrekturschleife die Druckkraft berechnet wird, lautet

$$\vec{F}_i^p(t) = -m_i \cdot \sum_j m_j \cdot \left(\frac{p_i(t)}{(\rho_i^*)^2} + \frac{p_j(t)}{(\rho_j^*)^2} \right) \cdot \nabla W(\vec{x}_{ij}, h). \quad (3.25)$$

3.3.4. Simulation von Festkörpern

Desbrun und Gascuel erweiterten in [DG96] die SPH-Methode, um inelastische kosmologische Festkörper durch eine Kombination von Viskosität und Steifigkeit zu simulieren. Diese Methoden wurden von Müller et al. in [MKN⁺04] aufgegriffen und um eine Kraft erweitert, die ein Objekt zu seiner ursprünglichen Form zurückstellt, die sogenannte Elastizität. Die ‘‘Moving Least Squares’’ Methode von Müller et al. hängt allerdings stark von der Matrixinvertierung ab und ist somit für gitterförmige Partikelnachbarschaften wegen der daraus entstehenden schlecht konditionierten Matrizen ungeeignet. Sohlerthaler entwickelte hierauf in [Sol10] eine rein SPH-basierte Methode, um Elastizität und Plastizität zu simulieren.

Elastizität Die Methode von Sohlerthaler beginnt mit der Energiedichte U_i eines Partikels, die durch

$$U_i = \bar{V}_i \cdot \frac{1}{2} \cdot (\epsilon_i \cdot \sigma_i) \quad (3.26)$$

angegeben wird. Dabei ist

$$\bar{V}_i = \bar{V}(\vec{x}_i) = \frac{m_i}{\sum_j m_j \cdot W(\vec{x}_{ij}, h)} \quad (3.27)$$

das Volumen, ϵ_i der Dehnungstensor und σ_i der Spannungstensor des Partikels i . Mit dem Elastizitätstensor C bilden diese Werte die verallgemeinerte Form des Hookeschen Gesetzes:

$$\sigma = C \otimes \epsilon. \quad (3.28)$$

Dabei sind σ und ϵ dreidimensionale Tensoren zweiten Ranges.

Der Dehnungstensor ϵ_i für ein Partikel hängt von der Verformung eines Objektes ab. Diese Verformung kann durch ein Deformationsfeld \vec{u} beschrieben werden. Für ein Partikel i und ein Nachbarpartikel j gilt nach [Sol10]:

$$\vec{u}_{ji} = \vec{u}_j - \vec{u}_i, \quad (3.29)$$

wobei \vec{u}_i die Abweichung der Partikel von ihrer Anfangsposition angibt.

Der Dehnungstensor wird in [BIT09] in zwei Formen angegeben: der nichtlineare Green-Saint-Venant Dehnungstensor

$$\epsilon = \frac{1}{2} \cdot (J^T \cdot J - I) \quad (3.30)$$

und der lineare Cauchy-Green Dehnungstensor

$$\epsilon = \frac{1}{2} \cdot (J + J^T - I). \quad (3.31)$$

Für die Jacobi-Matrix J der Abbildung $\vec{x}_0 \mapsto \vec{x}_0 + \vec{u}$ gilt:

$$J = \nabla \vec{x}_0 + (\nabla \vec{u})^T = I + (\nabla \vec{u})^T, \quad (3.32)$$

und somit ergibt sich

$$\epsilon = \frac{1}{2} \cdot (J^T \cdot J - I) = (\nabla \vec{u}) + (\nabla \vec{u})^T + (\nabla \vec{u}) \cdot (\nabla \vec{u})^T. \quad (3.33)$$

Dabei ist der Gradient des Dehnungsfeldes $\nabla \vec{u}$ eine 3×3 Matrix und wird in [Sol10] für eine Partikel i und ihre Nachbarn j durch

$$\nabla \vec{u}_i = \sum_j \bar{V}_j \cdot \nabla W(\vec{x}_{ij}, h) \cdot (\vec{u}_{ij})^T \quad (3.34)$$

ausgerechnet.

Durch die experimentell bestimmten Werte für

- E , der Elastizitätsmodul
- ν , die Poissonzahl

eines gegebenen Materials, können der Lamémodul (bzw. Lamés erster Parameter) λ und der Schubmodul (bzw. Lamés zweiter Parameter) μ bestimmt werden. Nach [Lur05] gilt

$$\lambda = \frac{E \cdot \nu}{(1 + \nu) \cdot (1 - 2 \cdot \nu)} \quad (3.35)$$

und

$$\mu = \frac{E}{2 \cdot (1 + \nu)}. \quad (3.36)$$

Die Komponente des Stresstensors σ kann folglich mit den Lamé- und Schubmodul durch

$$\sigma_{mn} = \lambda \cdot \epsilon_{kk} \cdot \delta_{mn} + 2\mu \cdot \epsilon_{mn}, \quad (3.37)$$

bestimmt werden, wobei

$$\delta_{mn} = \begin{cases} 1, & \text{falls } m = n \\ 0, & \text{sonst} \end{cases} \quad (3.38)$$

das Diracdreieck und ϵ_{kk} die Spur des Dehnungstensors darstellen.

Die Elastizität ergibt sich dann aus der Ableitung der Energiedichte U . Es gilt

$$\vec{F}_{ji}^{elastic} = -\nabla_{\vec{u}_j} U_i = -2 \cdot \bar{V}_i \cdot (I + \nabla \vec{u}_i^T) \cdot \sigma_i \cdot \vec{d}_{ij} \quad (3.39)$$

für die Elastizitätskraft, die von einem Partikel i auf eine Nachbarpartikel j wirkt. Der Operator $\nabla_{\vec{u}_j}$ ist die Richtungsableitung nach der Verformung von Partikel j . Es gilt

$$\vec{d}_{ij} = \frac{\partial \nabla \vec{u}_i}{\partial \vec{u}_j} = \bar{V}_j \cdot \nabla W(\vec{x}_{ij}, h) \quad (3.40)$$

für $j \neq i$.

Plastizität Falls ein Objekt so stark verformt wird, dass es dauerhaft verformt bleibt, nachdem die dafür verantwortliche Kraft nicht mehr vorhanden ist, so geht das Material in eine elastoplastische Verformung über. Dieser Übergang findet statt, nachdem das sogenannte Von-Mises-Spannungskriterium, das von O'Brien et al. in [OBH02] beschrieben wird, überschritten wurde.

Um festzustellen, ob das von-Mises-Spannungskriterium durch eine Verformung erfüllt ist, wird die gesamte Dehnung eines Objektes in eine elastische Dehnung ϵ_e und eine plastische Dehnung ϵ_p zerlegt. Die Summe

$$\epsilon = \epsilon_p + \epsilon_e \quad (3.41)$$

dieser beiden Teilspannungen bildet dann die gesamte Dehnung.

Die elastische Verformung eines Objektes kann in scherende und dehnende Effekte unterteilt werden. Um Volumenverfälschungen zu vermeiden, werden bei der Berechnung von plastischer Verformung nur die scherende Formabweichungen betrachtet. Damit dies gelingt, werden die dehnenden Anteile der vorhandenen elastischen Verformung folgendermaßen entfernt:

$$\epsilon' = \epsilon_e - \frac{\text{Spur}(\epsilon_e)}{3} I, \quad (3.42)$$

wobei ϵ' die scherende Formabweichung darstellt.

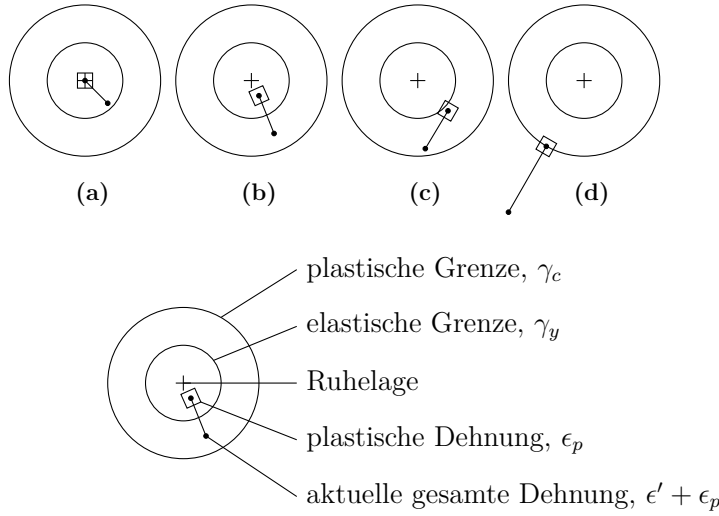


Abbildung 3.17.: Verhalten des Plastizitätsmodells. **(a)** Elastische Verformung. **(b)** und **(c)** plastische Verformung. **(d)** Materialbruch.[BIT09]

Solange diese scherende Formabweichung unter dem experimentell bestimmten Wert γ_y bleibt, also

$$\gamma_y < \|\epsilon'\|_F, \quad (3.43)$$

findet nur eine elastische Verformung statt (siehe Abbildung 3.17(a)). Mit dem Parameter γ_y wird also die Fließgrenze (engl. yield limit) eines Materials festgelegt. Dabei ist

$$\|\epsilon'\|_F = \sqrt{\text{Spur}(\epsilon'^* \epsilon')} \quad (3.44)$$

die Frobeniusnorm der scherenden Formabweichung und ϵ'^* die adjungierte Matrix zu ϵ' . Da aber ϵ' eine reellwertige Matrix ist, gilt $\epsilon'^* = \epsilon'^T$. Es ergibt sich also

$$\|\epsilon'\|_F = \sqrt{\text{Spur}(\epsilon'^T \epsilon')} = \sqrt{\sum_{i=1}^3 \sum_{j=1}^3 (\epsilon'_{ij})^2}. \quad (3.45)$$

In dem Fall, dass die Formabweichung eines Werkstücks über die Fließgrenze seines Materials steigt, findet eine plastische Verformung statt (siehe Abbildung 3.17(b) und (c)). Der plastische Anteil der gesamten Verformung wird mit

$$\Delta\epsilon_p = \frac{\|\epsilon'\|_F - \gamma_y}{\|\epsilon'\|_F} \epsilon' \quad (3.46)$$

angegeben. In einem iterativen Verfahren wie SPH muss die gesamte plastische Verformung in jedem Zeitschritt erneut berechnet werden. Dabei wird ein Parameter γ_c angegeben,

der eine obere Grenze für die gesamte plastische Verformung festlegt. Die Aktualisierung der plastischen Verformung geschieht mit

$$\epsilon_p := (\epsilon_p + \Delta\epsilon_p) \min \left(1, \frac{\gamma_c}{\|\epsilon_p + \Delta\epsilon_p\|_F} \right). \quad (3.47)$$

Falls $\|\epsilon_p + \Delta\epsilon_p\|_F > \gamma_c$ gilt, findet ein Materialbruch statt (siehe Abbildung 3.17(d)). In dieser Situation werden die Partikel, aus deren Auslenkung (Gl. 3.29) der Dehnungstensor bestimmt wurde, nicht mehr in den Elastizitätsberechnungen betrachtet und haben als Paar somit keinen Einfluß mehr auf die Kohärenz des Materials. Dabei bleibt natürlich die plastische Verformung, die bis zum Materialbruch stattgefunden hat, erhalten.

3.3.5. Bewertung und Ausblick

SPH wurde ursprünglich als Methode zur Simulation von Materialien in der fluiden Phase konzipiert und ist somit nicht ohne weitere Anpassungen für Festkörpermechanik geeignet. Solenthaler entwickelte in [Sol10] eine rein SPH-basierte Methode zur Simulation von Festkörpern. Allerdings weist dieser Ansatz besonders in der numerischen Stabilität einige Mängel auf, die sich im Laufe der Projektgruppe herausgestellt haben (siehe 7.5.3).

Die Rotation des Materials während der Span sich vom Werkstück wegbewegt stellt bei der Bildung eines Spans eine Schwierigkeit dar. Da der Cauchy-Saint-Venant Spannungstensor nicht rotationsinvariant ist, entstehen durch die Drehung Phantomkräfte. In [BIT09] wird das Dehnungsfeld in Rotations- und Dehnungskomponenten zerlegt, damit diese verfälschten Kräfte nicht mehr entstehen.

Ein weiteres Problem, das sich in der experimentellen Phase der Projektgruppe herausgestellt hat, war die von Monaghan in [Mon00] und Monaghan et al. in [MGS01] beschriebene tensile Instabilität, die besonders bei hohen Werten für den Elastizitätsmodul auftaucht. Diese Limitierung wird auch von Müller et al. in [MKN⁺04] erwähnt. Die in [Mon00] und [MGS01] vorgeschlagene Lösung ist eine weitere Kraft, die stabilisierend wirkt, wenn Teile des Werkstückes gedehnt werden.

Die Simulation von Festkörpermechanik mit SPH ist noch ein aktives Forschungsgebiet. Mit den oben genannten Erweiterungen des Modells könnten die Ergebnisse, die in 7.5.3 beschrieben werden, verbessert werden.

3.4. Movable Cellular Automata

Als mögliches Berechnungsverfahren für die Simulation spanender Fertigungsprozesse wurde auch die Movable-Cellular-Automata (MCA)-Methode untersucht. Wie bei SPH wird der zu simulierende Körper durch eine Menge diskreter Objekte dargestellt, die ein konfigurierbares Regelwerk für ihre Interaktion untereinander und mit ihrer Umgebung zugeteilt bekommen. Die für das Projekt relevanten Regeln werden von Psakhie et al. [PHK⁺95] für zwei Dimensionen erläutert. Die im Folgenden beschriebenen Eigenschaften sind dreidimensionale Verallgemeinerungen dieser Regeln.

Ein Automat i im Ruhezustand ist eine Kugel mit Radius r_i . Falls dieser Automat mit einem anderen Automaten j interagiert, wird der Abstand zwischen den beiden ausgehend von deren Mittelpunkten als r_{ij} bezeichnet.

3.4.1. Zustände

Zwei Automaten i und j können sich relativ zueinander in einem der folgenden Zustände befinden:

- nicht gekoppelt
- gekoppelt
- kontaktiert.

Dieser Zustand wird durch den Abstand zwischen den beiden Automaten und ihrem vorherigen Zustand entschieden. Hierbei sind folgende kritische Abstände ausschlaggebend:

- der maximale Kontaktabstand $r_{ij0} = r_i + r_j$,
- der maximale Abstand des gekoppelten Zustands r_{ijmax} ,
- der untere Schwellwert r_{ijmin} , bei dem ein Link erzeugt wird.

Im nicht gekoppelten Zustand haben Automaten keinen direkten Einfluss aufeinander. Schreitet der Abstand r_{ij} zwischen den Automaten i und j unter den unteren Schwellwert r_{ijmin} , so dass

$$r_{ij} \leq r_{ijmin} \quad (3.48)$$

gilt, so gehen i und j in einen gekoppelten Zustand über und es entstehen Kräfte zwischen den Automaten. Wenn die Automaten sich noch weiter nähern, so dass

$$r_{ij} \leq r_{ij0} \quad (3.49)$$

gilt, dann befinden sie sich in einem kontaktierten Zustand und Reibung kann stattfinden. Da der Kontaktabstand nicht größer als der Linking-Abstand sein kann, impliziert der kontaktierte Zustand den gekoppelten.

Bewegen Automaten sich wieder auseinander, so dass

$$r_{ij} > r_{ij_{max}} \quad (3.50)$$

gilt, dann wird der Link aufgelöst und sie gehen wieder in einen nicht gekoppelten Zustand über.

Falls zwei Automaten sich berühren wird deren Überlappung mit

$$h_{ij} = r_{ij} - r_{ij_0} \quad (3.51)$$

angegeben.

3.4.2. Auftretende Kräfte

Maßgeblich für die oben erwähnten Interaktionen sind die Kräfte, die in verschiedenen Situationen auf die Automaten wirken. Diese sind in Zentralkräfte $\vec{F}_{ij}^{zentral}$, die als Anziehung und Abstoßung zwischen den Automaten i und j auftauchen, und tangentiale Kräfte $\vec{F}_{ij}^{tangential}$, die die laterale Komponente der Interaktion darstellen, aufgeteilt. Die gesamte Kraft, die auf einen Automaten wirkt, ergibt sich als Summe dieser Kräfte,

$$\vec{F}_{ij} = \vec{F}_{ij}^{zentral} + \vec{F}_{ij}^{tangential}. \quad (3.52)$$

Im Folgenden wird aufgezeigt, wie aus dem Lennard-Jones-Potential ein Kraftgesetz abgeleitet werden kann.

3.4.2.1. Lennard-Jones-Potential

Das Lennard-Jones-Potential ist eine mathematisch einfache Approximation zur Beschreibung der Interaktion zwischen ungeladenen Teilchen (siehe [Kie11])

$$V_{LJ} = 4\varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right]. \quad (3.53)$$

Gleichung 3.53 stellt eine gebräuchliche Form des Lennard-Jones-Potentials V_{LJ} dar. Dabei stellt ε die Tiefe des Minimums und damit einen Parameter dar, mit dem sich die Amplitude des Potentials und damit auch der resultierenden Kraft modifizieren lässt. σ ist die Nullstelle des Potentials. Hauptbestandteil des Potentials sind der positive r^{-12} -Term, der einen repulsive Kraftanteil bewirkt, sowie der negative r^{-6} -Term, der für Anziehung sorgt. Der r^{-12} -Term stellt dabei eine Approximation eines Exponentialterms dar, welcher die quantenmechanische Austauschwechselwirkung zwischen ungeladenen Teilchen beschreibt, die stark abstoßend wirkt (Pauli-Repulsionskraft, siehe [BB07]). In Simulationen wird jedoch anstelle dessen ein polynomieller Term verwendet, da die

Berechnung der Exponentialfunktion eine höhere Rechenzeit benötigt. Der r^{-6} -Term approximiert die London-Kraft (siehe [Cas03]), welche einen Teil der van-der-Waals-Wechselwirkung ausmacht. Sie stellt eine kurzreichweitige anziehende Wechselwirkung zwischen Molekülen dar.

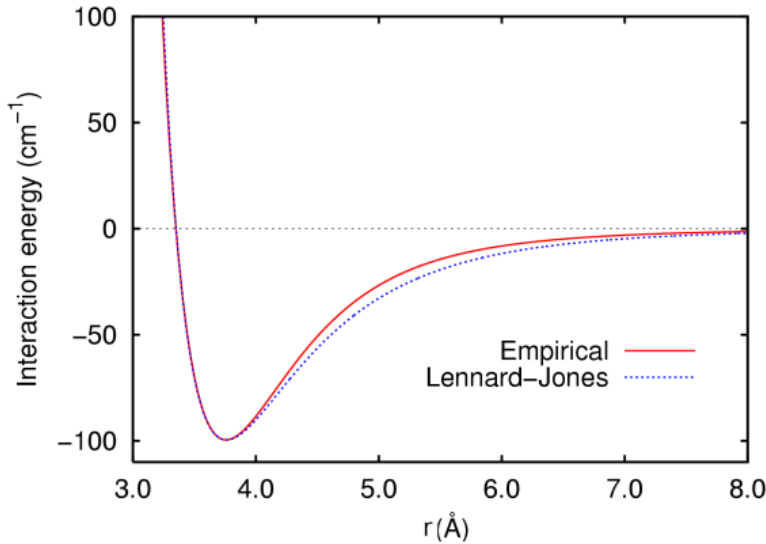


Abbildung 3.18.: Plot des Lennard-Jones-Potentials sowie Vergleich mit empirischer Wechselwirkungskurve [Jag08].

In Abbildung 3.18 ist ein Plot des Verlaufs des Lennard-Jones-Potentials (blau gestrichelter Graph) für Argon im Vergleich mit empirischen Werten (roter Graph) dargestellt. Die Unterschiede der Graphen sind sehr gering: Das Minimum liegt im gleichen Punkt. Da es sich hierbei um ein Extremum handelt, ist der Gradient an dieser Stelle 0, d.h. es wirkt keine Kraft. Dieses ist die Ruheposition der Teilchen. Für größere Abstände wirkt eine anziehende Kraft, da die Steigung hier positiv ist. Der Unterschied zwischen empirischer und Lennard-Jones-Kurve wird hier sichtbar, auch wenn es sich um wenige Prozent Abweichung handelt. Die Steigung des Lennard-Jones-Potentials sinkt an dieser Stelle zunächst stärker, woraus eine schwächer anziehende Kraft im Vergleich zu empirischen Werten resultiert. Für $r \approx 6,0 \text{ \AA}$ nähern sich die beiden Graphen jedoch wieder an. Die abstoßende Kraft verläuft nahezu identisch. Um mit Hilfe des Lennard-Jones-Potentials eine Simulation durchzuführen, ist die Ableitung eines Kraftgesetzes nötig. Im Folgenden wird aufgezeigt, wie dieses durchgeführt werden kann.

Eine Kraft wird als konservativ bezeichnet, wenn die Gleichung

$$\oint_C \vec{F}(\vec{r}) d\vec{r} = 0 \quad (3.54)$$

erfüllt ist, wobei $\vec{F}(\vec{r})$ eine Kraft ist. Mit Gleichung 3.54 berechnet sich dabei die Arbeit, die auf einem geschlossenen Weg im Kraftfeld berechnet wird. Für konservative Kraftfelder gilt demnach, dass die Arbeit für jeden beliebigen Weg im Kraftfeld gleich Null ist, sofern Anfangs- und Endpunkt des Weges identisch sind.

Aus der klassischen Mechanik ist weiterhin bekannt, dass für konservative Kräfte

$$\vec{F} = -k \cdot \vec{\nabla} \vec{U}(\vec{r}) \quad (3.55)$$

gilt, wobei \vec{U} ein Potential, k eine Ableitungskonstante, $\vec{\nabla}$ der Gradientenoperator und \vec{F} eine konservative Kraft ist. Durch Anwendung des Gradienten auf das Potential eines konservativen Kraftfeldes lässt sich die entsprechende Kraft berechnen, die in dem Kraftfeld ausgeübt wird.

Um auf Basis des Lennard-Jones-Potentials die Kraft \vec{F}^{LJ} zu berechnen, ist demnach die Anwendung des Gradienten auf das Potential nötig:

$$\begin{aligned} \vec{F}^{LJ} &= -\vec{\nabla} \cdot 4\varepsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \\ &= 24\varepsilon \left(\frac{2\sigma^{12}}{r^{13}} - \frac{\sigma^6}{r^7} \right). \end{aligned} \quad (3.56)$$

Gleichung 3.56 ergibt so die Kraft \vec{F}^{LJ} , die aus dem Lennard-Jones-Potential abgeleitet wurde. Die Parameter sind hier identisch zu denen in Gleichung 3.53.

3.4.2.2. Zentralkräfte

Zur Vermeidung der Überlagerung zweier Automaten wird das Lennard-Jones-Potenzial (Abb. 3.18) eingesetzt. Dieses wird von Psakhie et al.[PHK⁺95] für Automaten i und j folgendermaßen parametrisiert

$$\vec{F}_{ij}^{LJ}(r_{ij}) = \frac{\alpha_{ij}}{\sigma_{ij}} (n - m) \left\{ \left(\frac{r_{ij}}{\sigma_{ij}} \right)^{-(n+1)} - \left(\frac{r_{ij}}{\sigma_{ij}} \right)^{-(m+1)} \right\}. \quad (3.57)$$

Dabei ist α_{ij} die Tiefe des Potenzials und σ_{ij} der Abstand, bei dem die Automaten i und j im Gleichgewichtszustand ($\vec{p}_{ij} = \vec{0}$) sind. Typischerweise werden $m = 12$ und $n = 6$ gewählt.

3.4.2.3. Tangentiale Kräfte

Die tangentialen Kräfte zwischen zwei Automaten i und j werden in Reibung und Scherkraft unterteilt. Um die Reibung zwischen zwei Automaten zu berechnen, muss die Gleitgeschwindigkeit zwischen deren Oberflächen am Kontaktpunkt bzw. am Mittelpunkt

der Kontaktfläche ermittelt werden. Diese Geschwindigkeit setzt sich zusammen aus den Winkelgeschwindigkeiten $\vec{\omega}_i$ und $\vec{\omega}_j$ der Automaten und der Winkelgeschwindigkeit $\vec{\omega}_{ij}$, die die Rotation von j um i beschreibt. Um aus den Winkelgeschwindigkeiten der einzelnen Automaten eine Gleitgeschwindigkeit zu berechnen, muss der Vektor vom Mittelpunkt der Rotation, d.h. der Mittelpunkt des Automaten, zum Punkt, an dem die Reibung stattfindet, ermittelt werden. Dieser Vektor ist z.B. für Automat i

$$\vec{q}_i = \left(\frac{|\vec{r}_{ij}|}{2} + \frac{r_i^2 - r_j^2}{2 \cdot |\vec{r}_{ij}|} \right) \cdot \frac{\vec{r}_{ij}}{|\vec{r}_{ij}|}, \quad (3.58)$$

wobei \vec{r}_{ij} der Vektor vom Mittelpunkt von i zum Mittelpunkt von j und r_i bzw. r_j der Radius von i bzw. j sind. Mit anderen Worten ist \vec{q}_i der Vektor, der von dem Mittelpunkt des Automaten i zum Mittelpunkt der bei Überlappung zweier Automaten entstehenden Fläche läuft. Die gesamte Gleitgeschwindigkeit ist dann

$$\vec{v}_{ijS} = \vec{\omega}_{ij} \times \vec{r}_{ij} + \vec{\omega}_i \times \vec{q}_i + \vec{\omega}_j \times \vec{q}_j. \quad (3.59)$$

Die Kraft der Reibung richtet sich gegen die relative tangentielle Geschwindigkeit und wird mit dem Reibungskoeffizient η_{ij} skaliert:

$$\vec{F}_{ij}^{Viskos} = -\eta_{ij} \vec{v}_{ijS}. \quad (3.60)$$

Die Scherkraft \vec{F}_{ij}^{Scher} zwischen den Automaten i und j entsteht bei Verformung des gesamten Objekts. Entscheidend in diesem Fall ist der Winkel γ_i , der die Verschiebung eines Automaten i nach Verformung relativ zur Ausgangsposition angibt. In einem Zeitschritt Δt entsteht ein Verformungswinkel

$$\Delta\gamma_i = \frac{\vec{v}_{ij} \Delta t}{G_i} \cdot \left(\frac{|\vec{q}_i|}{G_i} + \frac{|\vec{q}_j|}{G_j} \right)^{-1}, \quad (3.61)$$

wobei \vec{v}_{ij} die relative tangentielle Geschwindigkeit und G_i bzw. G_j der Schubmodul für Automat i bzw. j sind.

Die dadurch entstehende Kraft ist

$$\vec{F}_{ij}^{Scher} = \tau_{ij} S_{ij}, \quad (3.62)$$

worin S_{ij} die Verformungsfläche zwischen i und j ist. Die Spannungsfunktion τ_{ij} wird von Psakhie et al.[PHK⁺95] in ihrer einfachsten Form als

$$\tau_{ij} = \begin{cases} G\gamma_{ij} & \gamma_{ij} < \gamma_{ijY} \\ G\gamma_{ijY} & \text{sonst,} \end{cases} \quad (3.63)$$

definiert, wobei γ_{ijY} der Verformungswinkel ist, bei dem das Material anfängt sich plastisch zu verformen. Dieser Winkel sowie der Schubmodul des Materials wird experimentell ermittelt.

3.4.2.4. Trockenreibungskraft

Für ein realistischeres Simulationsergebnis wird zusätzlich die Trockenreibungskraft berechnet:

$$\vec{F}_i^{\text{Reibung}} = \frac{\mu_{ij} \cdot (\vec{d}p \cdot \vec{F}_i) \cdot \vec{d}p}{r^2}. \quad (3.64)$$

Die Berechnung dieser Kraft erfolgt am Ende, da die Summe der auf i einwirkenden Kräfte \vec{F}_i bekannt sein muss. $\vec{d}p$ ist der Abstandsvektor zwischen i und j und r die Länge von diesem. μ_{ij} ist eine Trockenreibungskonstante zwischen i und j .

3.4.2.5. Auswirkung der Kräfte

Das Verhalten des gesamten Systems lässt sich als System zweier Differentialgleichungen beschreiben. Die Gleichung

$$\sum_j \vec{F}_{ij} = m_i \cdot \frac{d^2 \vec{x}_i}{dt^2} \quad (3.65)$$

beschreibt die gesamte Kraft, die auf einen Automaten i von allen seinen Nachbarn ausgeübt wird. Das effektive Drehmoment eines Automaten i wird mit

$$\sum_j \vec{K}_{ij} = \hat{j}_i \cdot \frac{d^2 \vec{\theta}_i}{dt^2} \quad (3.66)$$

beschrieben, wobei $\frac{d^2 \vec{\theta}_i}{dt^2}$ die Winkelbeschleunigung und \hat{j}_i das Trägheitsmoment eines Automaten i darstellen. Hier stellt \vec{K}_{ij} den Anteil vom gesamten Drehmoment des Automaten i , der aus der Kraft \vec{F}_{ij} entsteht, dar und lässt sich formulieren als

$$\vec{K}_{ij} = |\vec{q}_i| \left(\frac{\vec{r}_{ij}}{|\vec{r}_{ij}|} \times \vec{F}_{ij} \right). \quad (3.67)$$

3.4.3. Kollisionsbehandlung

Um die bei der Simulation aufgetretene fehlende Inkompressibilität der Partikel auszugleichen, wird eine Kollisionsbehandlung auf Grundlage des Impulsübertrages bei einem elastischen Stoß verwendet. Die Kollisionsgeschwindigkeit berechnet sich mit:

$$\vec{v}_{c_i} = \vec{v}_{oc_i} - \vec{v}_{o_i}. \quad (3.68)$$

Hier bei ist \vec{v}_{o_i} die orthogonale Geschwindigkeit von Automat i und \vec{v}_{oc_i} die orthogonale Kollisionsgeschwindigkeit von Automat i . Die orthogonale Geschwindigkeit lässt sich mit:

$$\vec{v}_{o_i} = (\vec{v}_i \cdot \vec{d}) \cdot \frac{\vec{d}}{r^2} \quad (3.69)$$

berechnen. \vec{d} ist der Richtungsvektor zwischen Automat j und Automat i , r ist der Abstand der beiden und \vec{v}_i die Geschwindigkeit von i . Die orthogonale Kollisionsgeschwindigkeit berechnet sich mit:

$$\vec{v}_{oc_i} = \frac{(m_i - m_j) \cdot \vec{v}_{o_i} + 2 \cdot m_j \cdot \vec{v}_{o_j}}{m}. \quad (3.70)$$

m_i und m_j sind die Massen von i und j und $m = m_i + m_j$. Die so berechnete Kollisionsgeschwindigkeit \vec{v}_{oc_i} wird zur Geschwindigkeit \vec{v}_i des Automaten hinzuaddiert. Die Kollisionsbehandlung erfolgt für j analog.

3.4.4. Bewertung

Da die MCA-Methode für die Beschreibung von Festkörperverhalten entwickelt wurde, bietet sie eine nachvollziehbare Möglichkeit, die Ziele der Projektgruppe zu realisieren. Obwohl die Methode noch relativ neu ist und die Menge der relevanten Literatur entsprechend klein ist, ist sie dennoch sehr adäquat, da die Gleichungen zur Beschreibung des Systemverhaltens aus bekannten physikalischen Regeln für Festkörper hergeleitet sind. In [PHK⁺95] werden weitere Systemeigenschaften beschrieben, die für das Projekt nicht relevant sind.

3.5. Feder-Masse-Systeme

Als dritte Alternative zur Simulation von Festkörpern wurden Feder-Masse-Systeme betrachtet. Zur Partikelinteraktion wurden sowohl Hooke'sche Federkräfte als auch das in Molekulardynamiksimulationen (vgl. [Kie11]) üblichere Lennard-Jones-Potential betrachtet. Details zur Methode der Feder-Masse-Systeme können dem Zwischenbericht dieser Projektgruppe entnommen werden. In diesem Kapitel werden Feder-Masse-Systeme nur grob beschrieben, ohne auf verwendete Kraftgesetze einzugehen. Abschließend wird darauf eingegangen, wieso diese Methode verworfen wurde.

3.5.1. Aufbau eines Feder-Masse-Systems

Feder-Masse-Systeme lassen sich mit gängigen Polygonnetz-Modellen [Smi06] vergleichen. Den größten Unterschied hierzu stellt die Unterteilung des ganzen Volumens statt nur der Unterteilung der außenliegenden Flächen dar. So wird nicht nur die Oberfläche aus z.B. kleinen Dreiecken in ein Netz unterteilt, sondern der gesamte Körper wird in kleinere, zumeist gleich geformte Elemente zerlegt. Optisch ähnelt ein Feder-Masse-System aus Hexaedern einem Voxel-Modell. Ein zusätzlicher Unterschied ist die Semantik der einzelnen Elemente. Die sonst in einem Polygonmodell als Knoten bezeichneten Verbindungen zwischen verschiedenen Kanten werden in einem Feder-Masse-System als unendlich kleine Punktmassen verstanden. Die Kanten zwischen den Knoten werden als elastische Federn modelliert, um so einen dynamischen Zusammenhalt zwischen den einzelnen Punktmassen zu simulieren.

Abbildung 3.19 zeigt eine exemplarische Darstellung eines hexaedrisch angeordneten Feder-Masse-Systems. Wie der Abbildung zu entnehmen ist, teilen sich die einzelnen Elemente, in diesem Fall Hexaeder, an den Berührungsflächen die selben Federn bzw. Punktmassen.

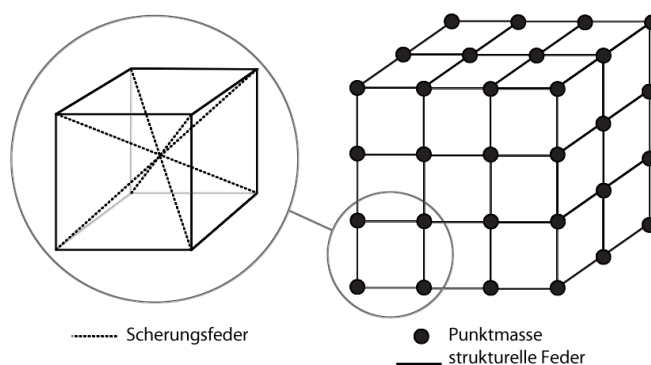


Abbildung 3.19.: Schematische Darstellung eines Feder-Masse-Systems mit Punkt-Massen und Federn [NMK⁺05].

Bezüglich der Federn eines Feder-Masse-Systems muss eine weitere Unterscheidung gemacht werden. Es existieren grundsätzlich zwei verschiedene Typen von Federn. Die Strukturfedern beschreiben die grundlegende Struktur einer Komponente eines Feder-Masse-Systems. Scherungsfedern fügen zusätzliche Stabilität in eine Komponente ein und gewährleisten eine sinnvolle Kraftverteilung bei scherartigen oder längsartige Verformungen. Dabei ist zu beachten, dass Strukturfedern und Scherungsfedern in Anlehnung an das Hooke'sche Gesetz verschiedene Federsteifigkeiten aufweisen, um, den verschiedenen Verformungen entsprechend, gute Widerstandsleistung zu gewährleisten. Für weitere Informationen zu Feder-Masse-Systemen, vor allem bezüglich der gängigen Integration, wird auf das Paper von Nealen et al [NMK⁺05] verwiesen.

3.5.2. Anwendung

Feder-Masse-Systeme wurden bisher hauptsächlich zur Animation von Kleidung benutzt oder auch im medizinischen Bereich zur Simulation von Haut. Vassilev et al. [VSC01] haben bereits 2002 einen Ansatz zur Animation von Kleidung bei sich bewegenden Menschen mittels Feder-Masse-Systemen vorgestellt. Wagner et al. haben den Simulator EyeSi entwickelt, der zur Ausbildung von Augenchirurgen eingesetzt wird. Die Simulation von Augengewebe innerhalb von EyeSi beruht auf Feder-Masse-Systemen. In ihrem Paper [WSM02] zeigen Wagner et al., dass Feder-Masse-Systeme in der Lage sind, Simulationen performant zu berechnen und trotzdem überzeugendes Gewebeverhalten nachzubilden. Sekercioglu und Duysak [SD09] haben 2009 mit Hilfe eines Feder-Masse-Systems einen Ansatz zur Simulation von molekularen Modellen vorgestellt. Dabei wurde die Haut eines Menschen modelliert und eine Nadelinjektion simuliert.

3.5.3. Bewertung

Die Simplizität des Aufbaus von Feder-Masse-Systemen stellt einen bestechenden Vorteil für die Konstruktion eines Modells dar. Gerade durch die Verwendung von Komponenten wie Hexaedern mit einer überschaubaren Anzahl von internen Federn lassen sich selbst komplexe Modelle in sehr intuitive Einheiten unterteilen. Es ist die Bestimmung von durchschnittlich nur sechs verschiedenen Federsteifigkeiten nötig, wodurch die Konstruktion eines Gesamtmodells stark vereinfacht wird. Diese Vereinfachung führt andererseits dazu, dass das gesamte Modell in seiner Genauigkeit stark beschränkt ist. Die Ersetzung der gesamten Fülle von physikalischen und mechanischen Kräften und Zusammenhängen durch drei fundamentale physikalische Eigenschaften suggeriert schon im Vorfeld, dass die bei einer Simulation erreichten Ergebnisse tendenziell Visualisierungscharakter haben.

Bis zum Zeitpunkt des Zwischenberichts wurde deshalb versucht ein Hybridmodell aus SPH und Feder-Masse-System zu realisieren, dass eine starke Bindung von Partikeln untereinander gewährleistet und trotzdem physikalische Gesetzmäßigkeiten soweit möglich befolgt. Während der Auswertung des Zwischenberichts wies das System aber derzeitige strukturelle Schwächen auf, da zum Beispiel Kompressionen zwar simuliert werden konnten, jedoch keine Scherungen von Objekten realistisch auftraten. Eine Alternative dazu bildeten die Elastizitäts- und Plastizitätskräfte des SPH-Zyklus (siehe Abschnitt 3.3.4). Zugunsten der SPH-nativen Kräfte wurde von einer weiteren Benutzung des SPH-FMS-Hybrids abgesehen.

Weiterhin wurden durch die Implementierung von SPH mit zusätzlichen Kräften und MCA zwei weitaus komplexere Modelle verwendet, weshalb die Methode der Feder-Masse-Systeme gänzlich verworfen wurde. Hooke'sche Federkräfte sind dabei als SPH-Kraft implementiert worden, bei MCA wird anstelle des Hooke'schen Federgesetzes die Ableitung des Lennard-Jones-Potentials zur Bestimmung der Partikelinteraktion verwendet.

3.6. Nachbarschaftssuche

Die Nachbarschaftssuche wird zur Optimierung der Laufzeit der Simulation benötigt, um effizient die Nachbarschaftspaare für die einzelnen MCA- oder SPH-Berechnungen zu bestimmen. Sie erlaubt virtuell im Rechner erzeugte Objekte auf ihre Nachbarschaft zu prüfen, also eine Nähe dieser festzustellen. Die Nachbarschaft besteht demzufolge aus einer Menge von Objektpaaren, die zueinander einen Abstand kleiner als den Suchradius aufweisen, der die Nachbarschaft definiert. Als Abstandsmaß wird die euklidische Distanz verwendet. Die Nachbarschaft ändert sich mit der Veränderung der Position von Objekten, womit sie innerhalb einer Implementierung auch immer wieder neu berechnet werden muss, falls es keine festen Nachbarschaften gibt. Für SPH sind die betrachteten Objekte die SPH-Partikel und für MCA die beweglichen zellulären Automaten. Im Folgenden werden die Begriffe Element und Partikel synonym für SPH-Partikel und MCA-Automaten verwendet, hingegen der Begriff Objekt im Allgemeinen für virtuelle Objekte jeglicher Art. Der Suchradius entspricht der Glättungslänge (engl. smoothing length) oder auch dem Glättungsradius der SPH-Formulierung, siehe Abschnitt 3.3, die im Glättungskern verwendet wird, oder dem Linkradius der MCA-Formulierung aus Abschnitt 3.4, der die Konnektivität der Automaten beschreibt. Das grundlegende Problem ist die Laufzeit der Nachbarschaftsberechnung, die sich aus der Anzahl der Objekte ergibt. Es zeigt sich, dass ein naiver Ansatz für eine wachsende Anzahl an Elementen sehr schnell ineffizient wird (Abschnitt 3.6.2).

Die Nachbarschaftssuche ist verwandt mit der Kollisionserkennung. Anstelle der Suche nach Objekten mit negativem Abstand (Kollision) werden Objekte innerhalb eines bestimmten Abstandes gesucht. Es können somit vor allem die gleichen Raumaufteilungsverfahren wie in der Kollisionserkennung verwendet werden. Die Kollisionserkennung wird in die Broad Phase, in der hauptsächlich die Anzahl der Kollisionspaare reduziert wird, und die Narrow Phase, in der Kandidaten für eine Kollision genau geprüft werden bzw. die eigentliche Schnittberechnung ausgeführt wird, eingeteilt ([Wei05] Kapitel 1). Entsprechendes gilt für die Nachbarschaftssuche, welche die gleiche Einteilung vornimmt. Dabei wird ebenfalls in einer Broad Phase die Anzahl der Nachbarschaftspaare reduziert, die übrigen müssen in der Narrow Phase auf eine tatsächliche Nachbarschaft geprüft werden. Beruhend auf der Verwandtschaft der Probleme werden in den folgenden Abschnitten Raumaufteilungsverfahren für die Kollisionserkennung nach citeericson vorgestellt. Diese werden für eine Verwendung als Nachbarschaftssuche für SPH und MCA entsprechend modifiziert als Suchdatenstrukturen und Suchverfahren vorgestellt.

Die Effizienz des Verfahrens zur Nachbarschaftssuche kann je nach Art der Problemstellung (Anzahl statischer und bewegter Partikel, Verteilung der Objekte im Raum, etc.) stark variieren. Als Beispiel ist bei Gasen oder Flüssigkeiten damit zu rechnen, dass ein großer Teil des Raumes meist keine Elemente enthält. Eine Suche auf leeren Räumen, die aufwendig verwaltet werden, kostet Rechenzeit, da diese Räume komplett durchsucht werden müssen. Hingegen besteht aufgrund der Aufgabenstellung, siehe Kapitel 1 und

2, der Schwerpunkt der Simulation auf Festkörpern, also auf Räumen, die sich durch geometrische Objekte gut eingrenzen bzw. beschreiben lassen. Die Partikel, die ein solches geometrisches Objekt beschreiben, füllen dabei die Räume gut aus, es gibt verhältnismäßig weniger leeren Raum. Weiterhin sind die Elemente, die eigentlich keine Größe haben, über ihren Glättungsradius oder Linkradius zu allgemeinen Kugelobjekten erweiterbar. Dadurch ergibt sich im Vergleich zur allgemeinen Nachbarschaftssuche, dass alle Objekte gleich groß sind bzw. eine gleich große Nachbarschaft aufweisen. Beide Besonderheiten sind ausschlaggebend für die Wahl der Implementierung der Nachbarschaftssuche, die in Kapitel 6.4 beschrieben wird.

3.6.1. Abstandsberechnung

Für den dreidimensionalen Raum ist der euklidische Abstand wie folgt definiert:

$$d(\vec{x}, \vec{y}) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2}. \quad (3.71)$$

Die Nachbarschaft eines Partikels sind somit die Partikel, deren Abstand $d(x, y) \leq r$ ist, wobei r der Suchradius ist. Für eine effizientere Implementierung ist anzumerken, dass die Nachbarn über das Quadrat des Abstandes berechnet werden können.

$$d^2(\vec{x}, \vec{y}) = (x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 \leq r^2. \quad (3.72)$$

Die Operation des Wurzelziehens, die zu den einfachen arithmetischen Operationen vergleichsmäßig rechenaufwändig ist, kann so eingespart werden.

3.6.2. Naive Methode

Zur Bewertung der Komplexität des Problems ist eine Betrachtung des naiven Ansatzes hinreichend. Der naive Ansatz basiert auf der Idee, für jedes Partikel die Nachbarschaft zu bestimmen, indem der Abstand zu allen anderen Partikel berechnet wird und dieser auf Einhaltung des Suchradius geprüft wird. Das entspricht der Berechnung aller Abstände zwischen allen Paaren von Partikeln. Die erste Formulierung bedeutet für n Partikel, dass $n \cdot (n - 1)$ Abstandsberechnungen und Vergleiche mit dem Suchradius durchgeführt werden müssen. Also wäre der Best-Case bereits $O(n^2)$ und damit gleich dem Worst-Case, da der Worst-Case genau dem Durchführen aller Abstandsberechnungen entspricht. Diese Anzahl an euklidischen Abstandsberechnungen ist nur für eine geringe Anzahl an Partikeln praktikabel. Anzumerken ist, dass dies ein reines Suchverfahren auf den Positionen der Partikel darstellt. Vor allem wird der Abstand jedes Partikelpaares zweimal berechnet, sofern nicht die Symmetrie der Abstände ausgenutzt wird. Es würden aber auch $n \cdot (n - 1)/2$ Abstände berechnet werden müssen.

Der Speicherverbrauch ist weniger ausschlaggebend, er orientiert sich naiv an der Anzahl

der Objektpaare mit $\Theta(n^2)$. Die einfachste Verbesserung stellt bereits das Speichern der Objektpaare dar, die einen Abstand kleiner als den Suchradius haben. Bestenfalls kann für jedes positive Ergebnis nur ein Eintrag für ein Nachbarschaftspaar durch die Ausnutzung der Symmetrie generiert werden. Somit ist die Ergebnismenge asymptotisch immer nach unten durch die Anzahl der Nachbarschaftspaare begrenzt.

3.6.3. Sortierungsverfahren

Aufgrund des Schwerpunkts der angestrebten Simulation von Festkörpern wurde die Idee, ein grundlegendes Sortierungsverfahren aus der Kollisionserkennung für die Zwecke der Simulation zu adaptieren, theoretisch betrachtet. Das grundlegende Sortierungsverfahren aus [CLMP95], was als „Sweep und Prune“ bezeichnet wird, ist ein Kollisionserkennungsverfahren für dreidimensionale Objekte im Raum. Die Grundidee ist eine Reduzierung der Dimension durch Projektion der Objekte auf die drei Raumachsen wie in Abbildung 3.20 dargestellt. Dazu werden die Objekte in je einer Liste für eine Raumachse entsprechend ihrer Anfangs- und Endposition gespeichert. Im Fall von den betrachteten Partikeln kann das durch die Position und den Suchradius umgesetzt werden. Die Einträge setzen sich dann aus der Position in der entsprechenden Raumrichtung plus und minus Suchradius zusammen. In jeder Liste, der so erzeugten Suchdatenstruktur, würden sich also die Elemente, die benachbart sind, in dieser Raumrichtung überschneiden. Ein Vergleich aller Listen würde als Suchverfahren dann die potenziellen Nachbarn ergeben. Eine mögliche Abwandlung ist die Reduzierung auf Raumebenen. Die Listen beinhalten dann weniger potenzielle Paare, aber das Berechnen der Listen ist etwas aufwendiger [CLMP95]. Anstelle von Listen (einfach, doppelt verkettet) können auch Arrays verwendet werden. Diese benötigen weniger Speicher im Vergleich zu doppelt verketteten Listen, sind aber weniger robust bei einer dynamischen Erhöhung der Anzahl der Objekte [Eri05]. Wegen der Sortierung ist es intuitiv, dass sobald in der Liste ein nicht potenzieller Nachbar gefunden wird, die Liste nicht weiter betrachtet werden muss. Für Kollisionsprobleme, somit auch für die Nachbarschaftssuche, mit vielen sich zwischen zwei Zeitschritten nur wenig bewegendenden Objekten, ist eine gut implementierte Liste effizient aktualisierbar [LMCG01]. Eine weitere Eigenschaft ist, dass der abgedeckte Raum somit die Größe des Darstellungsbereichs des Computersystems hat. Es handelt sich also um keine begrenzte Szene.

Für n Partikel würde sich ein zusätzlicher Speicherverbrauch von $\theta(3 \cdot n \cdot 2)$ ergeben. Dieser kann durch das Ausnutzen der Eigenschaft, dass die betrachteten Elemente gleich groß sind, auf $\theta(3 \cdot n)$ reduziert werden. Dazu wird lediglich die Position eines Partikels gespeichert. In dem Fall liegt keine Überschneidung vor. Es müssten nur eindimensionale Abstandsberechnung in beide Richtungen auf der Liste durchgeführt werden.

Die Nachbarschaft eines beliebigen Elementes zu bestimmen, bedeutet in allen Listen zu suchen. Im Worst-Case $O(n)$ für Listen, für Felder $O(\log n)$ bei binärer Suche und $\theta(1)$,

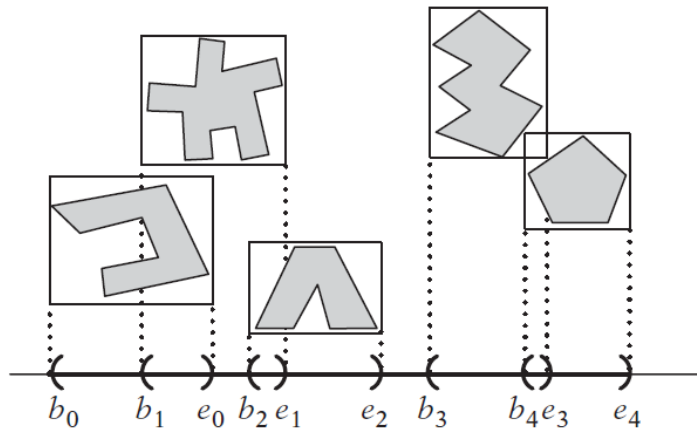


Abbildung 3.20.: Dimensionsreduktion aus [Eri05].

wenn für jedes Partikel seine Position in dem Feld abgespeichert ist (Direktzugriff). Für jede Liste kann für ein gesuchtes Partikel eine Liste der Kandidaten angelegt und diese dann verglichen werden. Alternativ kann direkt ein Vergleich ohne das Generieren einer Kandidatenliste stattfinden.

3.6.4. Äquidistante Raumaufteilung

Zur Reduzierung der Anzahl der Abstandsberechnung (Nachbarschaftskandidaten) bietet es sich an, den Raum in Teilräume zu unterteilen, da dann nur Abstände für Objekte in benachbarten oder gleichen Unterräumen berechnet werden müssen [Eri05]. Die einfachste Art der Raumzerlegung ist das Überdecken einer Szene mit einem regulären Gitter (äquidistante Raumaufteilung). Dabei wird der Raum in gleich große Intervalle aufgeteilt. Es reicht zum Erzeugen der Suchdatenstrukturen, die maximalen Raumkoordinaten der Szene durch die angestrebte Zellengröße zu teilen [Eri05]. Jede Zelle des mehrdimensionalen Feldes verwaltet die Objekte, die sie beinhaltet, mit linearen Listen oder doppelt verkettete Listen (oder alternativ Vektoren). Abbildung 3.21 stellt diesen Sachverhalt bildlich da.

Partikel, die nur eine Position haben, können mit dem Glättungsradius zu Kugeln erweitert werden, um die Nachbarschaft zu berechnen. Dabei muss ein so erweitertes Element jedoch in alle Zellen eingetragen werden, die es überdeckt oder berührt, wodurch sich mehrere Einträge ergeben. Das Auswerten würde bedeuten, dass alle Elemente aller Listen mit einem Eintrag eines Partikels auf Abstand mit diesem geprüft werden müssten. Ausgehend von einer Unterteilung des Raumes mit einem Abstand, der genau dem Suchradius entspricht, würde ein Partikel so bis zu 27 Zellen berühren oder überdecken. Bei einer Bewegung dieses Partikels müssten bis zu 27 Zellen aktualisiert werden. Im besten Fall liegt ein Partikel genau auf dem Berührungspunkt von 8 Zellen und somit

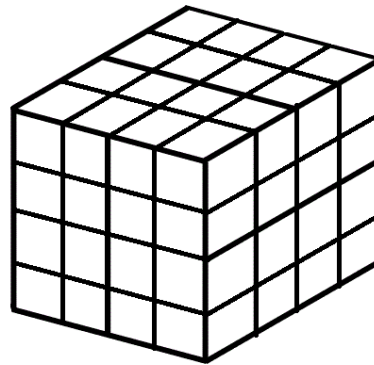


Abbildung 3.21.: Repräsentation der Raumaufteilung in drei Dimensionen.

müssten nur 8 Zellen durchsucht und aktualisiert werden. Voraussetzung für ein schnelles Finden aller Einträge eines Partikels ist das Abspeichern dieser, da sonst die ganze Struktur durchsucht werden muss. Alternativ kann auch von den Zellen aus gesucht werden, indem alle Zellen sequentiell geprüft werden. Ein Abspeichern der Zelle eines Elements entfällt dann. Für Partikel, die nur eine Position haben, bietet es sich an, diese nur in eine Zelle einzusortieren und eventuell nur die Zelle des Mittelpunktes zu speichern. Bei einer Bewegung muss so nur eine Zelle aktualisiert werden. Jedoch müssen für alle Partikel alle Nachbarzellen betrachtet werden, da ein Partikel mit Suchradius als Kugel größer als eine Zelle ist. Somit müssen bei einer Auswertung immer 27 Zellen durchsucht werden. In Abbildung 3.22 wird dieses für den zweidimensionalen Fall veranschaulicht. Die linke Abbildung stellt rot den Suchbereich aller möglichen Partikel dar, die sich in der blau dargestellten Zelle an beliebiger Position befinden. Zusätzlich zeigt der blaue Kreis den Suchradius eines Partikels, das sich genau in der Mitte der Zelle befindet. Die rechte Abbildung zeigt, dass für Partikel, die sich z.B. am Rand oder in die Ecke der mittleren Zelle befinden, einige Nachbarzellen nicht betrachtet werden müssen. Eine solche Positionierung eines Partikels, d.h. der Partikel liegt auf einer Kante, stellt aber eine Ausnahme dar. Eine zusätzliche Ausnutzung ist deswegen nicht rentabel. Eine Ausnutzung der Symmetrie der Abstände zweier Elemente erlaubt es, beim sequentiellen Auswerten der Zellen die Anzahl der zu betrachtenden Nachbarzellen zu reduzieren. Zellen, von denen aus bereits ausgewertet wurde, brauchen nicht mehr betrachtet zu werden, denn alle Partikelpaare, die von den anderen Zellen aus gefunden würden, wurden bereits beim Auswerten der Zelle gefunden.

Eine reguläre (äquidistante) Raumaufteilung ist auf den Szenenbereich, der unterteilt wird, beschränkt. Weiterhin ist je nach verwendetem Suchradius für die Unterteilung die Anzahl der Nachbarzellen, die durchsucht werden müssen, und der tatsächliche Suchraum, d.h. die Zahl der Kandidaten, die für eine Nachbarschaft in Betracht gezogen werden müssen, verschieden. In den betrachteten Quellen, wie zum Beispiel [Sol10] und [Kel06], wird der Glättungsradius für die Zellengröße verwendet. Baumstrukturen, die

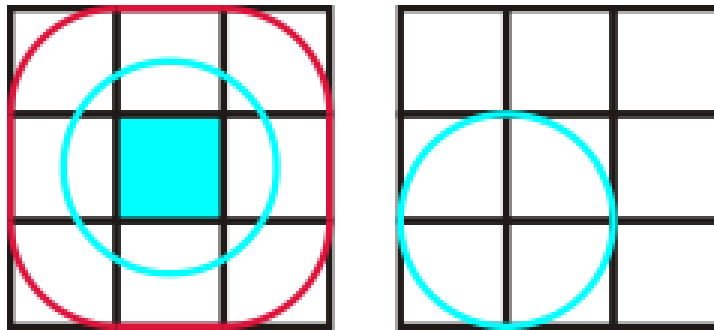


Abbildung 3.22.: Radius große Zelle in zwei Dimensionen.

eine äquidistante Raumaufteilung darstellen, werden hier grundsätzlich ausgeschlossen und nicht betrachtet. Die Begründung liegt in der Gegebenheit, dass die hier betrachteten Objekte alle die gleiche Größe aufweisen. Somit würde die Suche im Baum genauso viele Blätter ergeben wie Zellen, diese müssten aber erst im Baum gesucht werden. Die Intervallschachtelung der regulären Gitter hingegen gibt diese direkt. Ein Aufbau einer Baumsuchdatenstruktur wird im folgenden Abschnitt anhand eines k-d-Trees beschrieben.

3.6.5. Nicht äquidistante Raumaufteilung

Eine weitere Idee für die Nachbarschaftssuche stellen die bereits erwähnten Baumstrukturen dar, die den Raum rekursiv (äquidistant oder nicht gleichmäßig) unterteilen. Dabei entsteht eine Baumstruktur, in der die Blätter dann die Listen der Objekte beinhalten. Insbesondere sollen k-d-Trees für die Nachbarschaftssuche geeignet sein [Eri05]. Im Unterschied zu den meisten anderen Baumstrukturen wird bei k-d-Trees der Raum nicht gleichmäßig unterteilt. Zusätzlich wird bei k-d-Trees in jedem Schritt immer nur in einer Dimension unterteilt. Die Abbildung 3.23 stellt im zweidimensionalen Fall dar, dass ein Binärbaum entsteht. Weiterhin kann Abbildung 3.23 entnommen werden, dass der Raum nicht äquidistant unterteilt wird. Ein mögliches intuitives Vorgehen bei der Erzeugung ist das Unterteilen immer im Wechsel über verschiedene Raumachsen.

In Abbildung 3.24 ist das Verfahren in 3D dargestellt. Die rote Ebene unterteilt zuerst anhand der x-Achse den gesamten Würfel. Darauf unterteilt die grüne Ebene über die y-Achse eines der entstandenen Rechtecke. Das andere Rechteck wird nicht unterteilt, da eine Abbruchbedingung bereits erfüllt ist. Zuletzt unterteilen zwei gelbe Ebenen die im vorherigen Schritt entstandenen Rechtecke entlang der z-Achse.

Ohne genauer auf die Einteilung einzugehen, kann gesagt werden, dass hier verschiedene Strategien, wie zum Beispiel keine Objekte zu schneiden oder eine gleichmäßige Anzahl von Elementen pro Rechteck zu erreichen, verfolgt werden können. Im Vergleich zu regulären Gittern oder regulären Bäumen mit Intervallschachtelung als Raumaufteilung kann eine Neuberechnung mehr Probleme bereiten, da nicht nur die Zelleninhalte, sondern

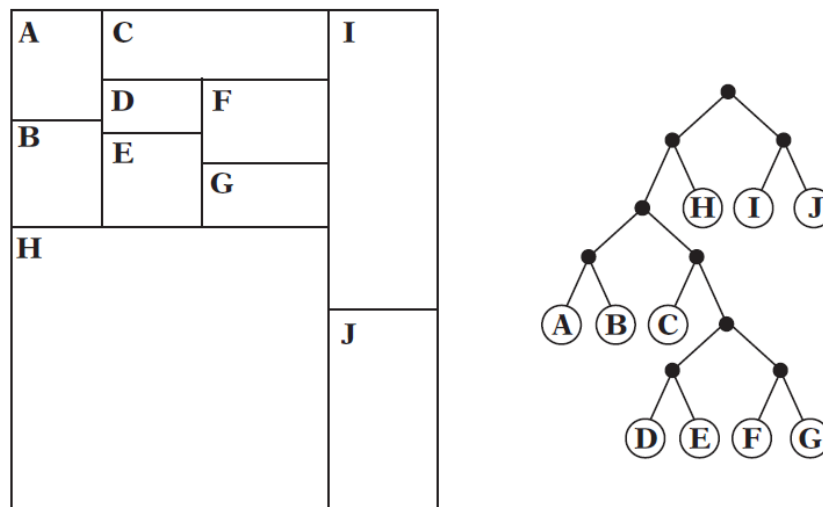


Abbildung 3.23.: k-d-Tree mit entsprechendem Baum [Eri05].

eventuell auch die Struktur aktualisiert werden muss.

Für Elemente, die zur Kugel erweitert sind, gibt es mehrere Einträge, die gefunden oder gespeichert werden müssen, um alle Listen zu finden, in denen potenzielle Nachbarn einsortiert sind. Wenn nur die Position eines Partikels gespeichert wird, muss in der Baumstruktur nach allen benachbarten Raumbereichen gesucht werden. Die k-d-Trees können als Vorteil große leere Bereiche zusammen fassen, die bei einer regulären äquidistanten Unterteilung mehrere leere Zellen umfassen würden. Jedoch benötigt das Finden der Zellen mehr Rechenzeit.

3.6.6. Hashing

Es ist zum Zweck der Reduzierung des Speicherverbrauchs möglich, über Hashtabellen die Anzahl der Listen zu reduzieren, die gespeichert werden müssen. Für ein Partikel muss jedoch immer noch eine Zelle berechnet werden, d.h. eine Art verallgemeinerte Raumposition, die eine Bestimmung der Nachbarkandidaten ermöglicht. Aus den Zellenkoordinaten lassen sich dann die Hashwerte sowie die Zellenkoordinaten der Nachbarzellen und deren Hashwert bestimmen. Entsprechend dem Hashingverfahren gibt es nur noch eine begrenzte Anzahl an Listen. Somit gibt es im Unterschied nicht mehr eine Liste pro Zelle. Durch das Hashing liegen in diesen Listen aber eventuell Elemente, die im Verfahren ohne Hashing bereits von einer möglichen Nachbarschaft ausgeschlossen wären. Die Anzahl der zu durchsuchenden Listen entspricht der Raumaufteilung ohne Hashing. Die Raumaufteilung mit Hashing ist durch den Darstellungsraum des Systems begrenzt. Nachteilig ist jedoch die Berechnung des Hashwerts, die rechenintensiv sein kann. Für jede Zelle kann der Hashwert bestimmt und abgespeichert werden, um die Rechenzeit durch

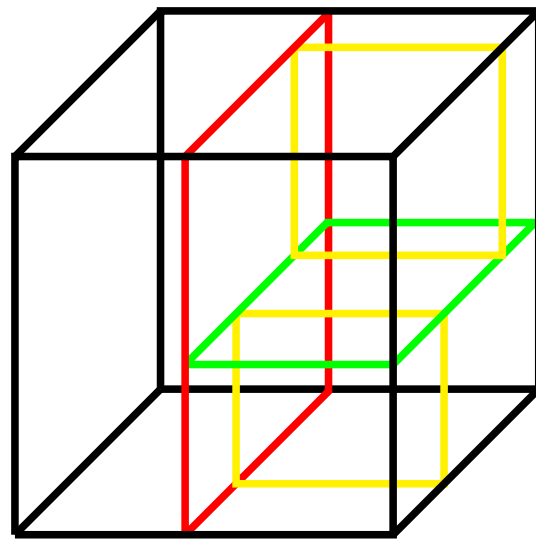


Abbildung 3.24.: k-d-Tree in drei Dimensionen.

weniger Hashingberechnungen zu verbessern, jedoch ist das Raumaufteilungsverfahren dann wieder durch die unterteilte Szene begrenzt.

4. Konzeption

In diesem Kapitel wird die Konzeption ausgewählter Komponenten der Simulation dargestellt. Durch die verschiedenartigen Anforderungen an die zu simulierenden Szenen und die unabhängigen Simulationsverfahren SPH und MCA wurden Szenetypen eingeführt, um eine Unterscheidung in ihrer Behandlung zu vereinfachen. Zum Einen gibt es **BLOCK**-Szenen, deren grundlegender Aufbau darin besteht, dass zwei Szeneobjekte z.B. unter Einfluss von Gravitation kollidieren (d.h. die Kollision eines plasto-elastisch verformbaren Objektes mit einem Boden). Dabei ist es möglich, durch bestimmte Parameter (siehe Benutzerhandbuch in Anhang C), die Eigenschaften der kollidierenden Objekte stark zu verändern. Einige Beispiele für **BLOCK**-Szenen sind in Abschnitt 7 zu finden. Als zweiten wichtigen Szenetypen gibt es **CUTTING**-Szenen, die auf Spanbildungssimulation spezialisiert sind. Dabei ist es möglich einen Schneidkeil zu definieren und das Werkstück zu fixieren. Auch für derartige Szenen lassen sich Beispiele in Abschnitt 7 finden. Da die Simulationsverfahren SPH und MCA mit beiden Szenetypen benutzbar sind, ergeben sich daraus vier Szenetypen **SPH_BLOCK**, **MCA_BLOCK**, **SPH_CUTTING** und **MCA_CUTTING**. Im folgenden Abschnitt wird auf die konzeptionellen Details eingegangen, die bei der Erstellung der Szenetypen eine Rolle spielen. Dazu zählt die Modellierung des Bodens, der in verschiedenen Ausprägungen getestet wurde. Weiterhin stellten die Konstruktion von Werkzeugen und die Fixierung von Werkstücken einen wichtigen Teil der Simulationsplanung dar.

4.1. Festkörpergeometrie

Die (konstruktive) Festkörpergeometrie ist eine Technik zur Modellierung von dreidimensionalen Körpern. Sie ermöglicht die Konstruktion durch logische Vereinigung, Differenz und Schnitt mehrerer dreidimensionaler primitiver Körper (z.B. Würfel, Kugel, Zylinder, Tori)[Mor99]. Zur Konstruktion von Werkzeugen, die zumindest zu Anfang der Planungsphase in mehreren Charakteristiken, wie Freiwinkel und Spanwinkel, variierbar sein sollten, ist die Verwendung der Festkörpergeometrie ein geeignetes Mittel. Nähere Details zur Nutzung der Festkörpergeometrie in diesem Projekt sind im Implementierungsteil in Abschnitt 6.3.1 („Geometrien“) zu finden.

4.2. Boundary-Partikel

In [IAGT10] wird vorgestellt, wie Begrenzungen mit Hilfe sogenannter Boundary-Partikel effizient durchgeführt werden können. Die Boundary-Partikel b besitzen dabei einen zu Beginn der Simulation berechneten Normalenvektor \vec{n}_b , der sich im Lauf der Simulation nicht mehr verändert. Der Normalenvektor gibt dabei an, in welche Richtung eine Abstoßung ausgeübt wird. Im Gegensatz zur Interaktion zweier SPH-Partikel ist die Richtung der Kraft also nicht vom Verhältnis der Positionen zweier Partikel abhängig, sondern allein vom Normalenvektor des Boundary-Partikels. Der Glättungsradius r_{Coll} wurde dabei halb so groß wie der SPH-Glättungsradius gewählt (siehe [IAGT10]). Da ein Partikel i im Einflussbereich mehrerer Boundary-Partikel sein kann, wird aus den Normalenvektoren ein durchschnittlicher Wert \vec{n}_i gebildet. Folgende Formeln geben die Berechnung an. $||\vec{x}_{ib}||$ beschreibt dabei den Abstand zwischen Partikel i und Boundary-Partikel b .

$$\vec{n}_i = \sum_b w_{ib} \vec{n}_b \quad (4.1)$$

$$w_{ib} = \frac{r_{Coll} - ||\vec{x}_{ib}||}{r_{Coll}}. \quad (4.2)$$

Offensichtlich ist der Einfluss eines Boundary-Partikels desto größer, je näher das Partikel i ist (sichtbar in Formel 4.3). Die Position \vec{x}_i und Geschwindigkeit \vec{v}_i der Partikel wird nach folgenden Formeln korrigiert. Hierfür sei $n_i := \frac{\vec{n}_i}{||\vec{n}_i||}$.

$$\vec{x}_i := \vec{x}_i - \frac{\sum_b (w_{ib} \cdot ||\vec{x}_{ib}||)}{\sum_b w_{ib} \cdot n_i} \quad (4.3)$$

$$\vec{v}_i := \vec{v}_i - (\vec{v}_i \cdot n_i) \cdot n_i \quad (4.4)$$

Zunächst wurden die Normalenvektoren der Boundary-Partikel manuell gesetzt. Bei einfachen, glatten Flächen, wie in dem im folgenden Abschnitt beschriebenen Kelchversuch, war diese Methode ausreichend und führte zu den gewünschten Ergebnissen. Da der Generator in der Lage ist, komplizierte geometrische Formen zu generieren, wurde entschieden, die Normalenberechnung bei der Erstellung der Boundary-Objekte automatisch durchzuführen. Dazu wird die Kovarianzmatrix mit Hilfe der Scattermatrix aus den Koordinaten der Nachbar-Boundary-Partikel geschätzt. Von dieser Matrix werden dann die Eigenvektoren und Eigenwerte ermittelt. Die zu den zwei größten Eigenwerten gehörenden Eigenvektoren ergeben die Hauptachsen entlang der Oberfläche, die aus den zuvor ausgewählten Nachbarn definiert wird. Der dritte und verbleibende Eigenvektor

steht senkrecht auf den zwei anderen Eigenvektoren und zeigt somit aus der Fläche heraus. Der Eigenvektor kann allerdings in beide Richtungen zeigen, nach innen und nach außen. Deshalb werden bei der Berechnung der Vektoren \mathbf{n}_i die Normalenvektoren \mathbf{n}_b so gerichtet, dass sie zum Partikel i zeigen. Realisiert wurde das über `glm::faceforward`. Eine Fläche aus Boundary-Partikeln wirkt dementsprechend in beide Richtungen abstoßend. Der Vorteil von dieser Realisierung ist, dass Boundary-Objekte, beispielsweise ein Schneidkeil, durch Boundary-Partikel realisiert werden können, die nur die Oberfläche des Boundary-Objekts umfassen.

In Verbindung mit dem MCA-Verfahren finden Boundary-Partikel keine Verwendung, da es hierbei eine eigene Kollisionsbehandlung gibt.

4.3. Modellierung eines Bodens

Zur Realisierung des Bodens wurden zwei verschiedene Vorgehensweisen in Betracht gezogen. Zum einen mittels Repräsentation durch implizite Flächen, zum anderen durch Umsetzung über partikelbasierte Objekte. Besonders mit Blick auf die Testfälle des Prototyps (vgl. Zwischenbericht), die einen Boden zur Reflexion eines fallenden Körpers benötigten, war diese Simulationskomponente immer wieder Teil der Planungsphase. Dieser Abschnitt skizziert die Überlegungen zur Modellierung eines Bodens, die durchlaufen wurden.

4.3.1. Implizite Flächen

Unter impliziten Flächen ist in diesem Zusammenhang ein fester Boden zu verstehen, an dem ein aufprallendes Objekt reflektiert wird. Dazu wird eine Ebene definiert, an der die Partikel abprallen sollen. Dies wird unter anderem mit einer Invertierung der Geschwindigkeit der am Boden auftreffenden Partikel erreicht. Das Auftreffen der Partikel wird durch eine einfache Abfrage der y -Koordinate der Partikel realisiert. Sobald ein Partikel eine gewisse Höhe unterschreitet, wird seine Geschwindigkeit in der entsprechenden Komponente invertiert.

Die Verwendung impliziter Flächen zeigte in ersten Versuchen trotz der starken physikalischen Vereinfachung verwertbare Ergebnisse zur Abschätzung des internen Verhaltens eines SPH-Blocks. Zusätzlicher Vorteil dieser Methode besteht darin, dass eine Reflexion nicht an sehr viele Parameter gebunden ist, wie zum Beispiel Druck oder Dichte. So verringerte sich anfangs die Parametersuche auf ein Objekt, statt auf Boden und Material. Die Implementierung eines partikelbasierten Bodens machte jedoch die Verwendung von impliziten Flächen obsolet, was im folgenden Abschnitt näher erläutert wird.

4.3.2. Partikelbasierter Boden

Ein partikelbasierter Boden lässt sich prinzipiell als weiteres Werkstück betrachten, das einen Boden aufgrund seiner flachen Ausprägung simuliert. Die Partikel des Bodens sind dabei wie die Partikel eines Objekts (Block oder Werkstück) ebenfalls abhängig von den gewählten Simulationsparametern der aktivierten Kräfte, da die Kräfte unmittelbar das Reflexionsverhalten beeinflussen. Einzige Besonderheit eines Partikelbodens stellt die lokale Fixierung der Partikel dar. Keines der Partikel wird durch externe oder interne Kräfte von seiner initialen Position bewegt.

So spielen Kräfte wie Druck, Dichte und Viskosität ebenfalls eine Rolle bei der Findung eines geeigneten Reflexionsverhaltens. Dieser Umstand stellt auch gleichzeitig das entscheidende Problem partikelbasierter Böden dar. Zu Beginn der Reflexionstests ergab sich die Schwierigkeit, zwei verschiedene Objekte so zu modellieren, dass eine wechselseitig nachvollziehbare Interaktion stattfinden kann. Insbesondere die zu Beginn unklaren Kraftverhältnisse führten zum Zeitpunkt des Prototyps zu einem nicht vorhersagbaren Verhalten der Bodeninteraktion.

Es ist möglich, den Boden sowohl mit normalen Simulationspartikeln als auch mit Boundary-Partikeln (Abschnitt 4.2) zu realisieren. Boundary-Partikel vereinfachen die Simulation eines festen Bodens, da sie starr im Raum liegen und keine Kräfte auf sie wirken. Bei den Simulationen des MCA-Verfahrens wird der Boden aus normalen Partikeln dargestellt. Mit den entsprechenden Materialparametern wird die Geschwindigkeit allerdings fest auf 0 gesetzt, damit die Partikel ihre Position beibehalten ([IAGT10]).

4.4. Konstruktion eines Werkzeugs

Das Werkzeug stellt eine fundamentale Komponente dar, ohne die eine schlussendliche Abschätzung des elastischen und plastischen Verhaltens von Werkstücken bei einer Spannbildung nicht möglich wäre. Primär bezieht sich dieser Abschnitt auf die Konstruktion eines Werkzeugs und seine durch die Projektgruppenanforderungen spezifizierten Ausprägungen.

4.4.1. Modellierung

Bei der Frage, wie das Werkzeug modelliert werden kann, so dass es mit dem Werkstück interagiert, wurden im wesentlichen die gleichen Überlegungen angestellt wie bei der Modellierung des Bodens (siehe Abschnitt 4.3). Desweiteren wurde ein Modellierungsansatz mit Partikeln in Betracht gezogen, der ähnlich dem aus Abschnitt 4.3.2 ist. Allerdings interagieren die Partikel des Werkzeugs mit denen des Werkstücks über eine echte Kollisionsbehandlung.

4.4.2. Werkzeugkonfiguration

Zur Auswertung der Spanbildungssimulation wurde als Projektgruppenanforderung die Variation der Konfiguration des Werkzeugs angegeben. Es sollte dadurch das Verhalten von Spanbildungen untersucht werden, die auf gebräuchlichen Parametern der Fertigung beruhen, zum Beispiel einem Spanwinkel von $5 - 20^\circ$ sowie ungebräuchlichen Parametern, von zum Beispiel Spanwinkeln von -20° . Die Umsetzung der Variationen des Spanwinkels konnte durch die Benutzung von konstruktiver Festkörpergeometrie (Abschnitt 4.1) realisiert werden.

Zusätzlich vorgegebene Einstellungsmöglichkeiten beinhalteten die Geschwindigkeit des Werkzeugs während eines Spanvorgangs und die Zustellung, also die Eindringtiefe des Werkzeugs in das Werkstück. Die Zustellung lässt sich mit einer Translation des Werkzeugs um die vom Benutzer bestimmte Tiefe realisieren. Bezüglich der Zustellung wurde ebenfalls die Festlegung der Anzahl der Partikelschichten in Betracht gezogen.

Die Geschwindigkeit des Werkzeugs kann durch eine Translation pro Zeitschritt bewerkstelligt werden. Da die Bewegung nicht wie anfangs geplant durch die Spezifikation einer Trajektorie, sondern durch einen Richtungsvektor und die Geschwindigkeit realisiert wurde, stellt die Variation der Geschwindigkeit kein Problem dar.

4.5. Fixieren des Werkzeugs

Dieser Abschnitt widmet sich dem Problem der Fixierung des Werkzeugs. Im Allgemeinen betrifft die Fixierung alle virtuellen Objekte, die nicht durch externe Kräfte, wie beispielsweise Gravitationskraft, von ihrer Position verschoben werden sollen. Ein Beispiel dazu ist der partikelbasierte Boden, der in Abschnitt 4.3.2 beschrieben wurde. Das Werkstück soll zusätzlich gegen Verschiebung geschützt sein. Dies wird wie beim partikelbasierten Boden durch fixierte Partikel erreicht, die ihre Position durch interne oder externe Kräfte nicht ändern können. Die Generatorimplementierung erlaubt es, eine oder mehrere unten liegende Partikelschichten des Werkstücks als fest zu setzen. Während der Simulation können die nicht fixierten Partikel die fixierten nicht durchdringen, wenn der Zeitschritt klein genug gewählt wurde. Dies stellen die SPH- und MCA-Methoden sicher. Deshalb bleibt das Werkstück als Ganzes trotz aktivierter Gravitationskraft auf seiner Position. Bei einem zu groß gewählten Zeitschritt ist es aufgrund der Diskretisierung jedoch möglich, dass die beweglichen in die unbeweglichen Partikel eintauchen, siehe Abbildung 7.4. Die letzten Spalten des Werkstücks können ebenfalls fixiert werden, um zu verhindern, dass die freien Partikel im Werkstück vom Werkzeug weggedrückt werden. Sie stellen damit eine Art Wand dar. Die Fixierung der Spalten kann auch höhenbegrenzt vorgenommen werden. Das erlaubt dem Werkstück auf das Werkzeug mit einer Verformung zu reagieren. Ein zu weich gewähltes Werkstück führt allerdings dazu, dass Partikel vor den Wänden ineinandergeschoben werden und die Dichte in der Nähe der Wand damit sehr stark

zunimmt. Abbildung 4.1 stellt in grün die fixierten und fixierenden Partikel im Werkstück dar.

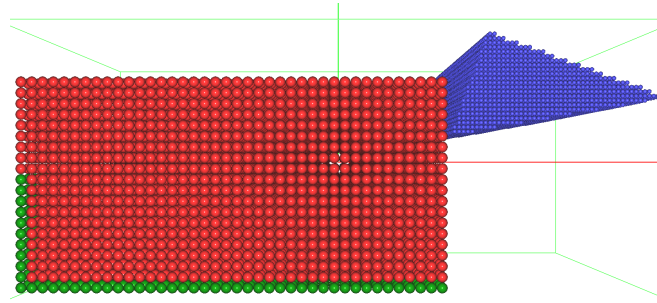


Abbildung 4.1.: Werkstückfixierung

Wie für den partikelbasierten Boden ist auch hier eine zusätzliche Kollisionsbehandlung möglich, um ein Durchdringen der fixen Partikel zu vermeiden, siehe Abschnitt 4.2. Jedoch zeigt es sich, dass bei Objekten, die eine höhere Festigkeit als Flüssigkeit aufzeigen, eine Kollisionsbehandlung nicht nötig ist.

4.6. Oberflächenrauheit des Werkstücks

Die in den Projektgruppenanforderungen spezifizierte Rauheit des Werkstücks konnte aufgrund des partikelbasierten Ansatzes unproblematisch realisiert werden. Zur Konstruktion einer rauen Oberfläche wird die Werkstückoberfläche mit einer Sinuswelle abgetastet und entsprechend mit Partikeln gefüllt, um so eine raue, wellenartige Oberfläche zu ermöglichen. Die Amplitude und die Frequenz der Welle können dabei beliebig gewählt werden. Eine exemplarische Darstellung dieser Rauheit ist in Abbildung 4.2 dargestellt.

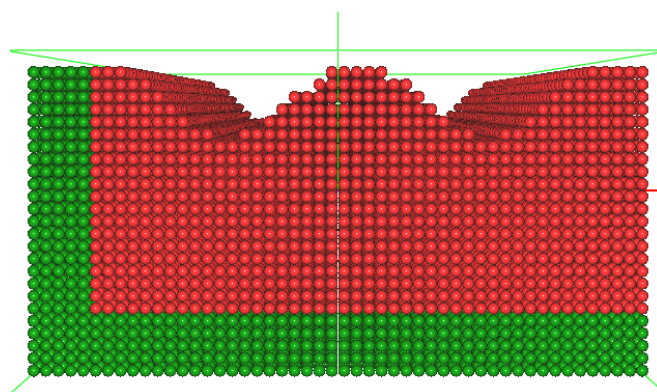


Abbildung 4.2.: Werkstück mit rauer Oberfläche

5. Projektbeschreibung

Dieses Kapitel beschreibt, welche Funktionalitäten das Projekt während seiner Entwicklung angeboten hat. Eine Auflistung der zu Anfang festgelegten Ausprägungen sind in einem Auszug des Pflichtenhefts in Anhang A zu finden. In diesem Kapitel wird unter anderem auch Bezug zu dem ersten Prototypen genommen und die speziellen Erweiterungen im finalen Programm aufgezeigt. Die Beschreibung der Programmstruktur dient dazu, einen verständlichen Rahmen zu vermitteln, um so die Inhalte des Kapitels 6 in einen Kontext zu setzen. Zuletzt werden explizit einige Features aus dem Pflichtenheft genannt, die nicht realisiert worden sind.

5.1. Prototyp

Zum Ende des ersten Semesters der Projektzeit wurde ein Prototyp entwickelt, der einige aus den Projektgruppenzielen in Kapitel 2.1 genannte Funktionalitäten angeboten hat. Der Prototyp bestand zum Zeitpunkt des Zwischenberichts bereits aus einer Simulationssteuerung und einer Visualisierung (siehe Abbildung 5.1), die mittels zwei verschiedener Renderer je nach Hardwareanforderung die Simulation visualisiert hat (siehe dazu auch Kapitel 6.2).

Durch Benutzung eines Feder-Masse-SPH-Hybrids (siehe Kapitel 3.5.3 und Zwischenbericht) konnten folgende Testszenarien simuliert werden:

- Plastischer Stoß eines Partikelwürfels mit einem Boden (Partikelboden und impliziter Boden)
- Kippen eines Würfels an einer Kante
- Zerteilen eines Partikelwürfels an einem Partikelkeil.

Die zur Simulation benötigten Parameter wurden über Projektdateien eingelesen. Diese Projektdateien enthielten den benötigten Simulationsrahmen, also Ausrichtung der Objekte sowie physikalische Parameter wie spezifischer Druck, Dichte und auch die Federkräfte. Nähere Details des Zusammenhangs der Komponenten und der internen Verarbeitung sind dem Zwischenbericht zu entnehmen und werden hier nicht erneut aufgeführt.

Der Prototyp ergab bei der Benutzung von SPH, vor allem in Verbindung mit Feder-Masse-Systemen, brauchbare Ergebnisse. Aufgrund der starken Vereinfachung von phy-

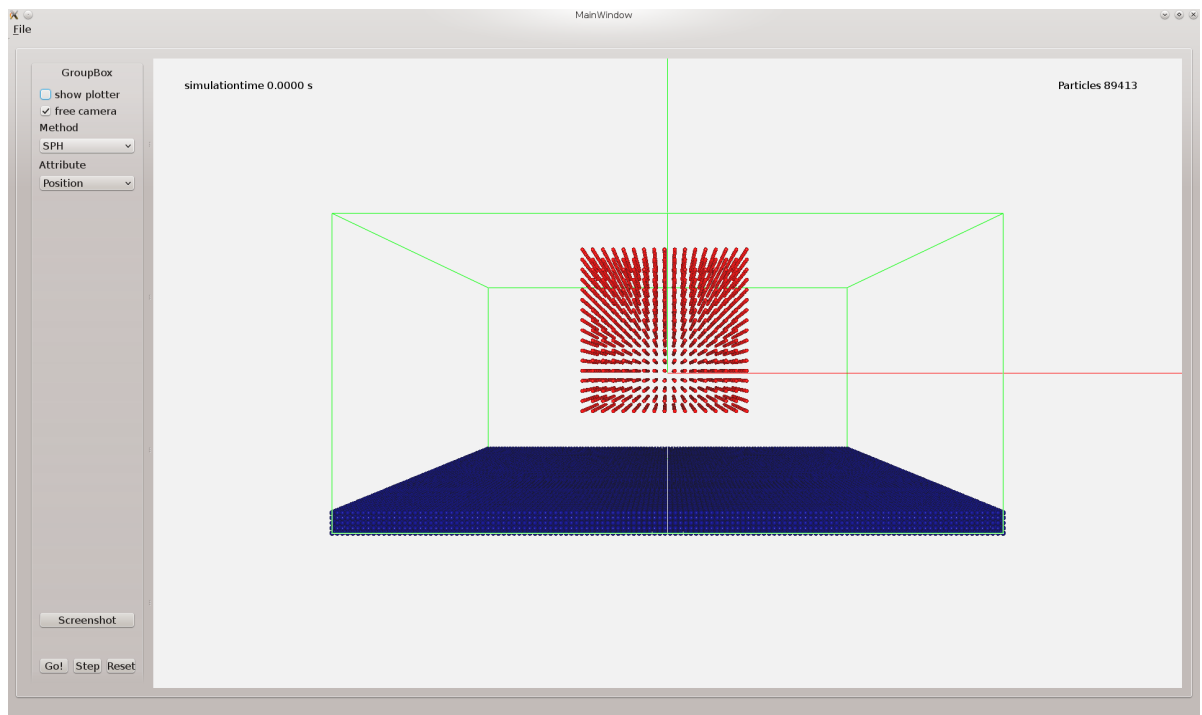


Abbildung 5.1.: Benutzeroberfläche des Prototyps

sikalischen Zusammenhängen durch Feder-Masse-Systeme (siehe Kapitel 3.5) wurde diese Hybrid-Version nach dem ersten Semester nicht weiter verfolgt. Die durch den Prototyp gewonnenen Erkenntnisse wurden im zweiten Semester der Projektgruppe zur Entwicklung eines Endprodukts verwendet. Zusätzlich wurde bereits der Rahmen einer Simulation mittels MCA (siehe Kapitel 3.4) in den Prototypen eingebaut. Jedoch konnte zum Zeitpunkt des Prototyps noch keine voll funktionsfähige Version getestet werden.

5.2. Endprodukt

In diesem Abschnitt wird das fertige Endprodukt der Projektgruppe ChipSim vorgestellt. Insbesondere liegen die neu hinzugefügten Funktionalitäten und Ausprägungen der Komponenten im Fokus der Beschreibung. In Abbildung 5.2 ist die Benutzeroberfläche des Programms zu sehen. Eine detaillierte Erläuterung der Bedienung ist im beiliegenden Benutzerhandbuch (siehe Anhang C) zu finden.

Das Endprodukt unterstützt zwei Simulationsvarianten: MCA und SPH. Es wird im Gegensatz zum Prototypen keine explizite Auswahl der Simulationemethode mehr angeboten, sondern es wird automatisch bei Auswahl einer Projektdatei die entsprechende Simulationemethode gewählt. Feder-Masse-Systeme werden nicht mehr unterstützt, da die

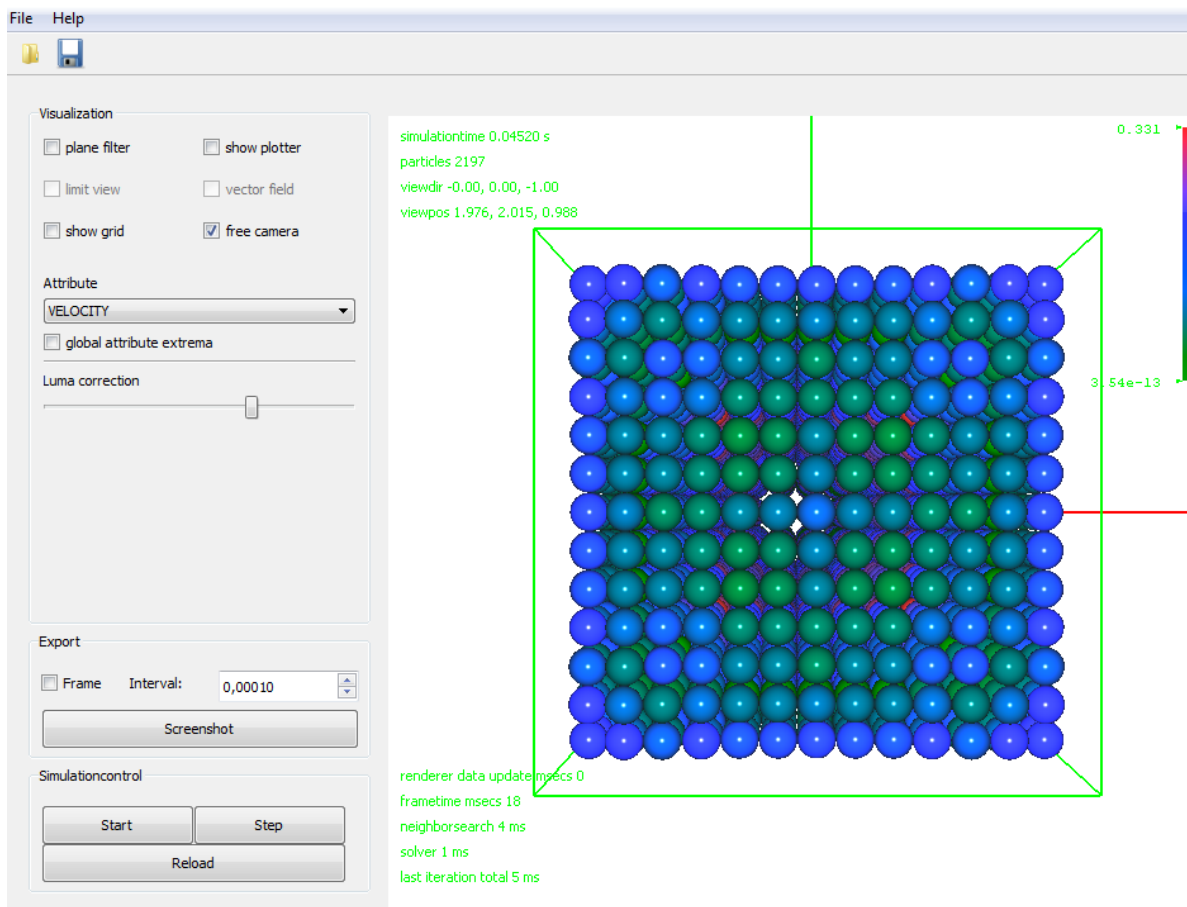


Abbildung 5.2.: Benutzeroberfläche des Endprodukts

dort getroffenen physikalischen Vereinfachungen stark einschränkend sind (siehe Abschnitt 3.5.3). Die Projektdateien enthalten Informationen über die Szene, wie zum Beispiel Position und geometrische Ausprägungen der Simulationskomponenten sowie deren physikalische Eigenschaften. Nähere Informationen zur Konstruktion einer Projektdatei sind ebenfalls dem Benutzerhandbuch zu entnehmen.

Die neue Menüführung ermöglicht eine leichtere Bedienung der gesamten Simulation. So kann eine Simulation beliebig gestartet, pausiert oder neu geladen werden. Zusätzlich zu der Darstellung von Positionen ist das Programm in der Lage interne Kräfte, wie Beschleunigung und Kraftpotential für beide Methoden farblich darzustellen. Die neuen Exportfunktionalitäten erlauben es, Screenshots aktueller Zeitpunkte oder Video-Frames der gesamten Simulation zu speichern. Ferner ist es möglich, den gesamten vorliegenden Zustand einer Simulation zu exportieren. Dadurch kann eine Simulation zu einem späteren Zeitpunkt problemlos geladen und weitersimuliert werden. Über Optionsschalter und Schieberegler ist es beispielsweise möglich die Zellen für die Nachbarschaftssuche anzuzei-

gen, nur eine einzelne Ebene an Partikeln einzublenden oder die Luminanz anzupassen. Im Simulationsbereich werden neben der simulierten Zeit auch Partikelanzahl und die genauen Rechenzeiten für ein Frame angezeigt.

Mit dem beiliegenden Batch-Script können mehrere Konfigurationen mit variierenden Parametern automatisiert sequentiell vom Programm simuliert werden. Je nach Konfiguration wird nach einem vom Benutzer bestimmten Abbruchkriterium der vorliegende Zustand der Simulation gespeichert. So können mit wenig Aufwand große Mengen von Konfigurationen simuliert und später gezielt evaluiert werden. Diese Funktionalität wurde auch zur Auswertung der Parametertests und Spanbildungsverhaltens verwendet (siehe Kapitel 7).

In Kapitel 6 werden die Implementierungen der einzelnen Komponenten des Programms erläutert. Dabei wird ebenfalls auf gewählte Implementierungsentscheidungen eingegangen.

6. Implementierung

Dieses Kapitel befasst sich mit der Umsetzung der theoretischen Überlegungen und beschreibt insbesondere Abweichungen zur Theorie. Im ersten Teil wird aufgezeigt wie die Simulationsumgebung, welche die Teilmodule des Projekts verwaltet und für das Speichern und Laden von Projektdateien zuständig ist, arbeitet. Anschließend wird auf das Visualisierungsmodul eingegangen, wobei die Funktionsweise des Renderers näher erläutert wird. Daraufhin wird die Implementierung des Generators der Szene und der Nachbarschaftssuche beschrieben, die zwar nicht im Vordergrund der Simulation stehen, jedoch wichtige Kernkomponenten darstellen. Es schließt sich eine Erläuterung der für die Physiksimulation zuständigen Kernkomponenten des Projekts an. Diese setzen sich aus den Teilmodulen für SPH und MCA zusammen. Die Kernkomponenten werden in genannter Reihenfolge in eigenen Unterkapiteln, in denen ihre Implementierung und die hierbei getroffenen Entwurfsentscheidungen beschrieben werden, behandelt.

Abbildung 6.1 veranschaulicht die Abhängigkeiten der Hauptkomponenten. Aus Gründen der Übersichtlichkeit wird die `commonlib` nicht dargestellt. Diese Komponente wird von allen anderen verwendet, da hier unter anderem zentrale Datentypen definiert werden.

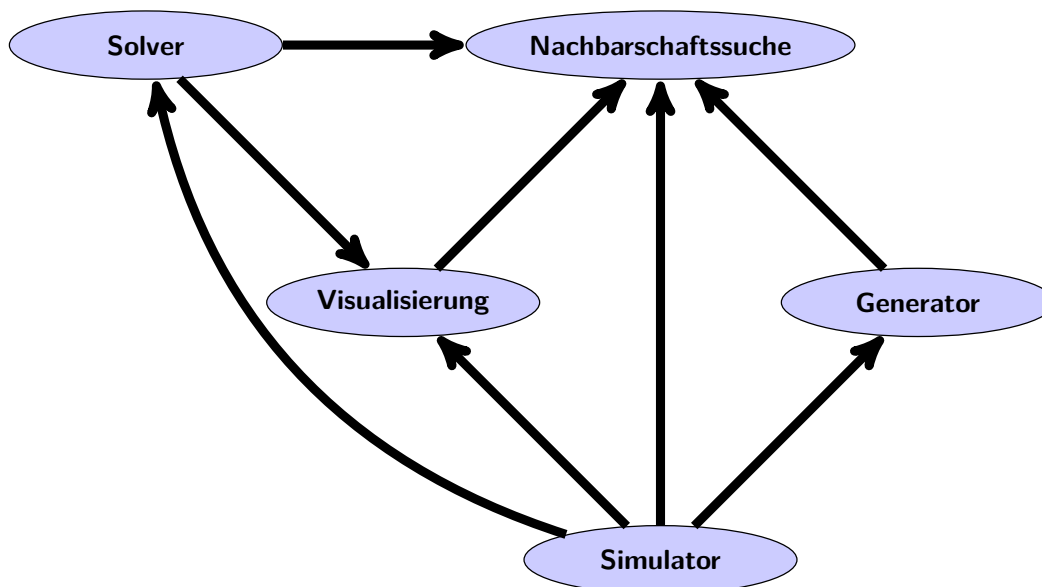


Abbildung 6.1.: Abhängigkeitsgraph der Hauptkomponenten.

6.1. Simulator

Im Folgenden wird die Implementierung des Simulators beschrieben. Wie auch die restlichen Module ist dieser in C++ geschrieben und nutzt die Qt Library für GUI, Signalsteuerung und Multithreading.

Der Simulator selbst ist eine Kapselung von einer Simulationssteuerung, einem Simulationsworkerthread, der jeweils abhängig von der gewählten Methode die Simulation durchführt, und einem Projektdateiparser, welcher Daten im JSON (JavaScript Object Notation)[Cro12] Format einliest. Der Simulator ist hierbei getrennt von der Visualisierung, welche ausschließlich Signale des Simulators entgegennimmt. Die Oberfläche wurde bereits in Abbildung 5.2 in Kapitel 5.2 gezeigt. Die Implementierungsdetails der Funktionalitäten werden im Folgenden beschrieben.

6.1.1. Startphase

Da die Entwicklung auf Rechnern mit unterschiedlicher OpenGL-Unterstützung erfolgte, wurden zwei Renderer geschrieben, wobei einer mit OpenGL 1.4 auskommt und ein weiterer OpenGL 2.1 mit Shadermodel-3-Support erwartet. Zu Beginn wird zunächst versucht, den zweiten Renderer zu initialisieren, falls dies jedoch fehlschlägt, wird die Initialisierung abgebrochen und der erste Renderer als Fallback gewählt. Näheres findet sich in Kapitel 6.2.

6.1.2. Initialisierungsphase

Der Projektfileparser bietet die Möglichkeit, Parameter für die Initialisierung einer Simulation aus einer Textdatei zu laden. Dazu wird das JSON-Format verwendet, da es relativ wenig Overhead mitbringt, von Menschen lesbar ist und ein freier Parser verfügbar ist. Der Simulator initialisiert alle für eine Simulationsmethode notwendigen und in einem Projektfile spezifizierten Daten und konstruiert die Szene mit Hilfe des Generators. Nachdem diese Initphase abgeschlossen ist, erwartet der Simulator von der GUI das Signal für den Start der Simulation.

6.1.3. Simulationsphase

Sobald die Simulation angestoßen wurde, wird ein Iterationstimer gestartet, der den Simulationsworkerthread zur Fortführung seiner blockierenden Endlosschleife veranlasst. Der Simulationsworkerthread berechnet also grundsätzlich nur eine Iteration und blockiert daraufhin auf einem Mutex bis die Blockade durch ein Signal, in diesem Fall das Auslaufen des Iterationstimers, wieder aufgehoben wird. Diese Vorgehensweise wurde gewählt, um zu ermöglichen, dass die Visualisierung parallel auf ihre Kopie der Daten in der Grafikkarte

zugreifen kann, ohne dabei zu streng mit dem Simulator synchronisiert zu sein. Damit wird auch eine flüssige Darstellung ermöglicht, wenn die Geschwindigkeit des Simulators für eine Echtzeitdarstellung nicht mehr ausreicht.

Sobald eine Iteration des Simulatorworkerthreads zu Ende berechnet wurde, sendet dieser ein Signal an den Simulator, was diesen dazu veranlasst, die Ergebnisse der aktuellen Iteration an den jeweiligen Renderer zu schicken, welcher damit ein Update seiner internen Repräsentation der Daten für die Visualisierung durchführt. Nachdem diese Iteration abgeschlossen ist und der Renderer signalisiert hat, dass er mit der Bearbeitung des neuen Datensnapshots fertig ist, kann durch den Iterationstimer die Berechnung der nächsten Iteration angestoßen werden.

Solver

Der Simulationsworkerthread benutzt den Solver, um die einzelnen Iterationsschritte der Simulation durchzuführen. Dieser ist eine Verallgemeinerung der speziellen Solver für die beiden Simulationstypen SPH und MCA. Abbildung 6.2 ist ein Sequenzdiagramm, das das Verhalten des Simulators bzw. die Interaktionen der von ihm aufgerufenen Solverklassen skizziert. Zu Beginn wird der entsprechend gewählte und vom allgemeinen Solver abgeleitete spezielle Solver vom Simulationsworkerthread initialisiert, wobei die Initialisierung des allgemeinen Solvers angestoßen wird. Diese sorgt dafür, dass ein anfänglicher adaptiver Zeitschritt berechnet wird, falls kein fester Zeitschritt angegeben wurde. Zusätzlich werden die Vektorgrößen für die Beschleunigungen und Indizes an die Partikelanzahl angepasst und berechnet, sowie die Nachbarschaftssuche initialisiert. Anschließend wird in der Funktion `preStep` die Nachbarschaftsberechnung aufgerufen, so dass in der darauffolgend aufgerufenen Funktion `step` ein Iterationsschritt durchgeführt werden kann. Je nach dem welcher Solvertyp gewählt wurde, wird an dieser Stelle die Funktion `step` des entsprechenden Solvers ausgeführt. Näheres hierzu wird in den Kapiteln 6.5 und 6.6 erläutert. Unabhängig von der gewählten Methode wird jedoch immer in der Funktion `calculateGravityForce` die Gravitationskraft berechnet, die auf alle Partikel wirkt. Nachdem die gewählte Simulationsmethode alle Berechnungen für einen Iterationsschritt vollendet hat, werden in der Funktion `verwirkliche` die neuen Positionen und - falls nötig - Beschleunigungen mit Hilfe des Leapfrog-Integrationschemas [Kie11] für alle Partikel berechnet und gesetzt. Hierbei ist zu beachten, dass der MCA-Solver eine zusätzliche Funktion `verwirkliche` implementiert, auf die in Kapitel 6.6 näher eingegangen wird. Abschließend wird in der Funktion `postStep` ggf. nochmal der adaptive Zeitschritt für die Simulation angepasst. Abbildung 6.2 zeigt den groben Ablauf eines Simulationsschritts.

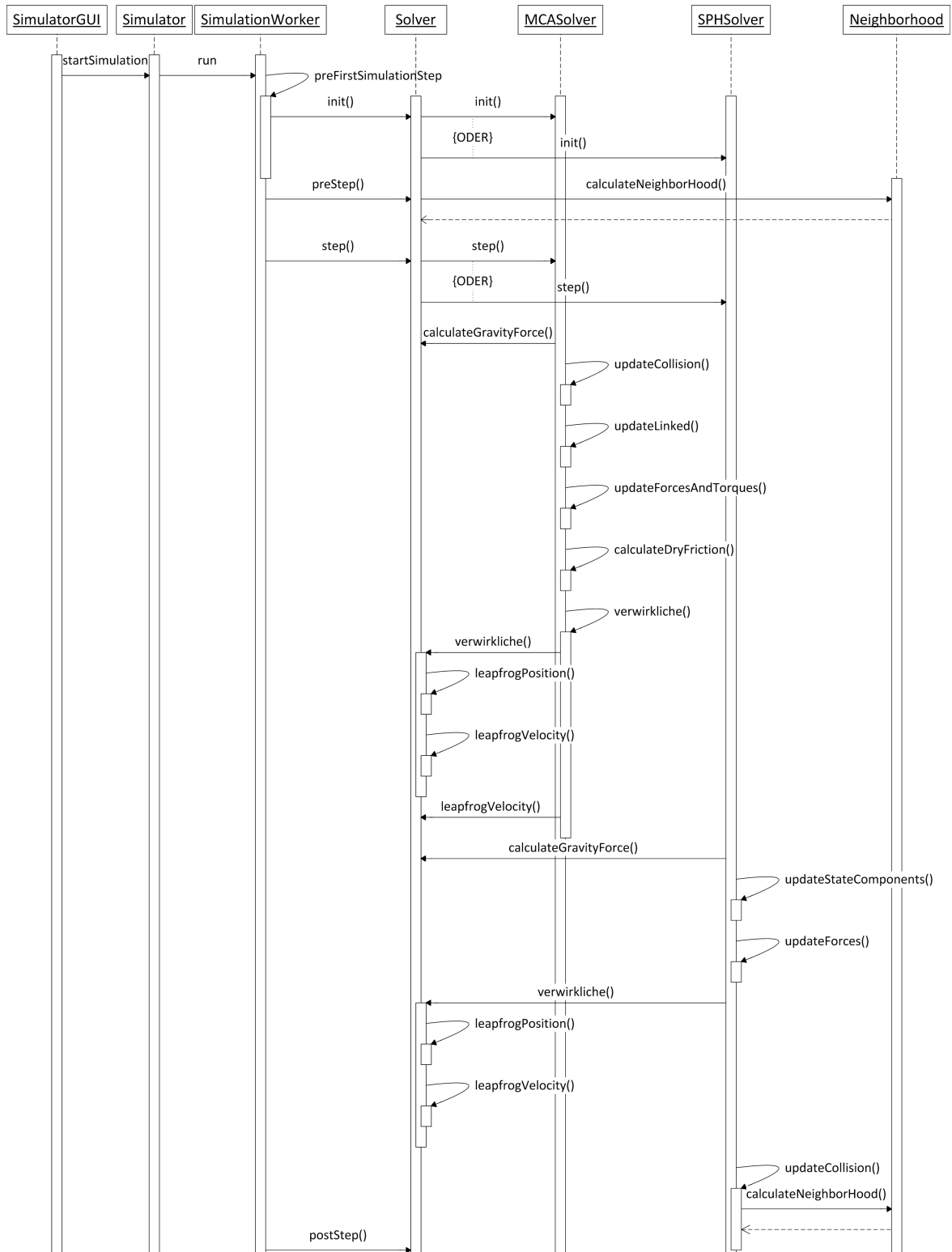


Abbildung 6.2.: Sequenzdiagramm zum Simulator (grob skizziert)

6.2. Visualisierung

Die Visualisierung der Simulation stellt für die Projektgruppe ein wichtiges Werkzeug zur Evaluierung der erzielten Ergebnisse dar. Im Folgenden werden die gewählten Mechanismen zur Visualisierung beschrieben und das Zusammenspiel der einzelnen Bestandteile erläutert. Für die eigentliche Visualisierung wurde OpenGL[The12] und die Shadersprache GLSL verwendet.

Die Visualisierung besteht im Wesentlichen aus drei verschiedenen Teilen: dem Ressourcenmanager, einem Renderwidget und einer graphischen Verarbeitung durch Shader. Im Folgenden werden alle drei Kernkomponenten beschrieben.

6.2.1. Ressourcenmanager

Der Ressourcenmanager ist zu Beginn des Programmstarts dafür zuständig, den Renderer mit einem Format für den OpenGL Kontext zu versorgen, um sicherzustellen, dass die gewählten Visualisierungsdirektiven (für OpenGL und Shader) auf der Maschine benutzt werden. Hat die Szene mittels des Formats einen Kontext erstellt, wird dieser Kontext vom Ressourcenmanager auf Versionsnummer (gewählt wurde OpenGL Version 2.1 oder alternativ 1.4 für leistungsschwächere Grafikkarten) und verschiedene Features geprüft, z.B. `GL_ARB_fragment_shader`, `GL_ARB_vertex_shader` oder `GL_ARB_geometry_shader4`, die benötigt werden, um die Daten mittels Shadern darzustellen (Kapitel 6.2.3).

Die in den Berechnungsschritten veränderten Daten werden durch den Ressourcenmanager mit Hilfe von *Vertex Buffer Objects* [SA06] in ein für den Renderer handhabbares Format gebracht. Dafür wird für jedes Partikel jeweils das durch die GUI vorgegebene Attribut ebenfalls in einem Vertex Buffer Object abgespeichert. Wurden für jeden neuen Visualisierungsschritt die benötigten Daten entsprechend gesammelt, wird ein durch den Ressourcenmanager verwaltetes `RenderObject` aktualisiert.

6.2.2. Renderwidget

Das Renderwidget ist das Visualisierungselement der graphischen Oberfläche. Da verschiedene Systeme auch über verschiedene OpenGL Versionen und Fähigkeiten verfügen, wird er mit einem `QGLFormat` Objekt initialisiert, damit ein Renderwidget, das mit dem vorhandenen System kompatibel ist, erzeugt werden kann. Zur Gewährleistung einer einheitlichen Schnittstelle zwischen dem naiven Renderwidget, der ältere OpenGL Versionen mit eingeschränkter Funktionalität unterstützt, und dem normalen Renderwidget, wird von dem Renderwidget-Interface geerbt.

Benutzereingabe

Eingaben zur Steuerung der Visualisierung während einer Simulation werden von dem Renderwidget abgefangen und als Steuerungsbefehle an die Kamera und den Simulator weitergegeben. Durch Tastatur- und Mauseingabe kann z.B. die Position und der Blickwinkel der Kamera eingestellt oder die Simulation angehalten bzw. neu gestartet werden.

6.2.3. Shader

Je nach Attributtyp, bei dem es sich in diesem Fall um Vektoren oder Skalare handeln kann, gibt es jeweils einen Vertex- und einen Fragmentshader. Dabei gibt es jeweils pro Shader noch zwei Alternativversionen, mit oder ohne **Geometry Shader**-Unterstützung. Aufgabe der Shader ist es, beleuchtete Pseudokugeln nur unter Verwendung von `GL_POINTS` zu rendern, wobei die Grundfarbe der Kugel abhängig von dem gewählten Attribut ist. So gilt zum Beispiel für das Attribut "Position", dass der Boden grün und die Box/Sphäre der Testfälle rot gezeichnet werden. Für andere Attribute hängt die Farbwahl von dem jeweiligen Wert des Attributs für das jeweilige Partikel ab. Vektorielle Werte werden in den Shader übernommen, wo sie auf das Intervall $r, g, b \in [0.0, 1.0]$ abgebildet werden. Alternativ können vektorielle Werte statt durch eine Farbskala mit einem Vektorfeld dargestellt werden. Die genannte Farbskala bewegt sich zwischen den Werten grün, blau und rot, wobei jeder Farbwert ungefähr 1/3 des Farbsegments einnimmt. Für skalare Werte wird gegenwärtig eine Farbskala aufgebaut, welche das Intervall $[\text{scalarMin}, \text{scalarMax}]$ auf einen Gradienten von grün über blau zu rot abbildet.

Zusätzlich zu den oben genannten Shadern kann im Falle der Simulationsmethode SPH noch der Parameter Glättungslänge (`SmoothingLength`) optisch dargestellt werden. Dafür werden transparente Kugeln über die dargestellten Kugeln gelegt, um die Reichweite der Glättungslänge darzustellen (siehe Abbildung 6.3).

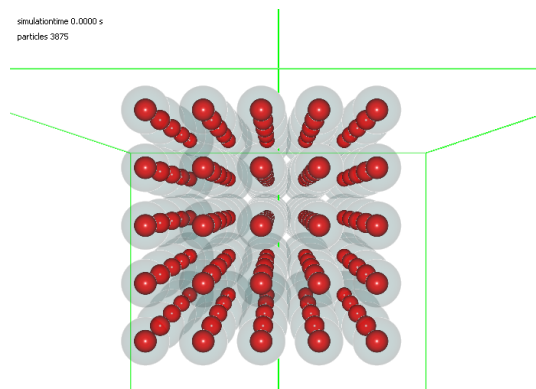


Abbildung 6.3.: Visualisierung der Glättungslänge

Insbesondere für die Vektordarstellung von gerichteten Kräften ist es erforderlich, die Szene mittels Clippingplanes zu filtern. OpenGL liefert mit Clippingplanes und `gl_ClipVertex` im Shader die Möglichkeit, mit Hilfe von definierten Punkten Bereiche zwischen Halbräumen auszuschneiden. Mit dieser Funktionalität wurde die in Abbildung 6.4 sichtbare Darstellung von gerichteten Pfeilen erreicht. Als weiteres Hilfsmittel zur Evaluierung von Simulationen wurde eine Funktionalität eingebaut, mit der es möglich ist einen bestimmten Bereich der Simulation isoliert anzuzeigen. Dafür werden zwei `ClippingPlanes` benutzt und derart zueinander ausgerichtet, dass zwischen den beiden `ClippingPlanes` ein bestimmter Bereich unberührt bleibt, also noch angezeigt werden. Diese Einstellung wurde benutzt, um eine Vektorfeldansicht auf das simulierte Objekt zu erhalten, so dass die Richtungen, in die die Kräfte wirkten, während einer laufenden Simulation besser erkennbar waren. Abbildung 6.5 zeigt die Ansicht für einen auf einen Keil fallenden Würfel.

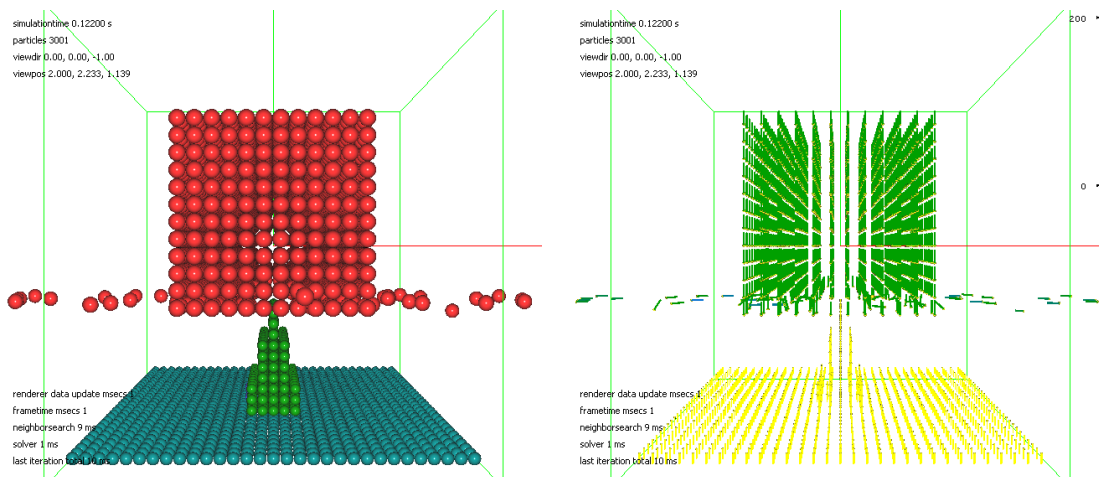


Abbildung 6.4.: Normale Ansicht und Vektorfeldansicht

6.3. Generator

Der Generator ist ein Teil des Projekts, der benötigte Geometrien erzeugt und korrekt initialisiert. Um den Anforderungen der Testfälle zum Ende der Projektgruppe gerecht zu werden, wird für das gesamte Projekt eine erweiterte Version des Generators aus dem Zwischenbericht für die MCA-Methode verwendet. Das bedeutet, dass alle Grundfunktionalitäten übernommen wurden und nur entsprechende Änderungen vorgenommen wurden, sodass auch SPH-Szenen generiert werden können. Konkret wurde dazu von den MCA-Parameter abstrahiert und ein `GeneratorInterface` eingeführt, von dem dann die MCA- und SPH-Zustandsklasse erbt. Zusätzlich wurde die Funktionalität grundlegend erweitert. Der Generator unterstützt verschiedene Szenentypen, die sich im Aufbau

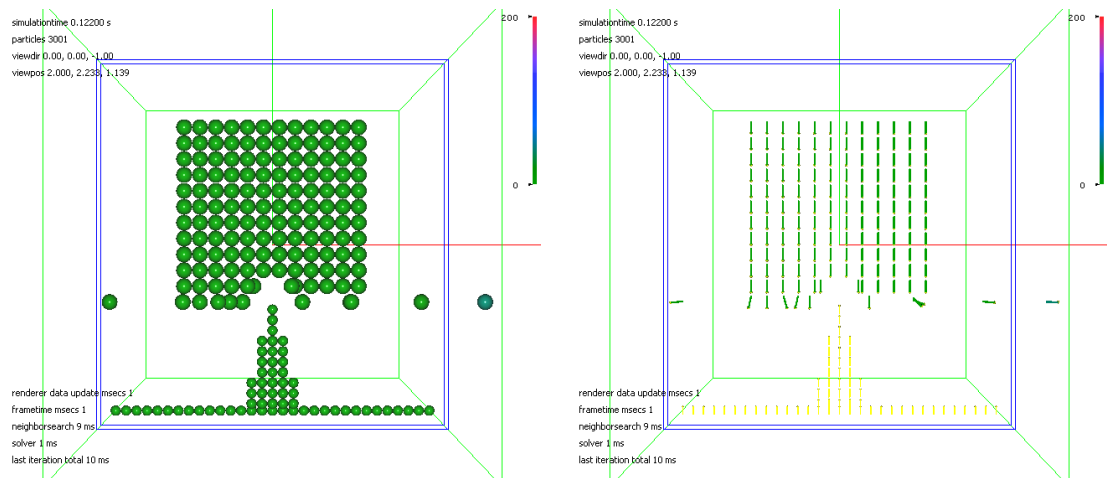


Abbildung 6.5.: Clippingplane bei normaler Ansicht und bei Vektorfeldansicht

grundlegend unterscheiden. Die Szenetypen sind unterteilt in die SPH-Szenen `SPH_BLOCK`, `SPH_CUTTING` und `SPH_BOUNDARY` und die MCA-Szenen `MCA_BLOCK`, `MCA_CUTTING` und `MCA_BOUNDARY`. Für jeden dieser Szenetypen stellt der Generator entsprechende Funktionalität bereit, um die Szenen zu erstellen und die Datenstrukturen mit den Werten aus den Projektdateien zu initialisieren. Im Folgenden sollen die dabei implementierten Module im Detail erläutert werden.

6.3.1. Geometrien

Ein wichtiger Bestandteil des Generators ist die Klasse `Geometry`, die sich in der `commonlib` des Projekts befindet. Es handelt sich dabei um eine rein virtuelle Klasse, die ein gemeinsames Interface für verschiedenartige Geometrien bildet. Die geometrische Beschreibung ist durch ein System umgesetzt, das auf abtastbaren komponierbaren Mengenbeschreibungen basiert. Die Beschreibungen sind in der Software durch das Dekorierermuster realisiert.

In Abbildung 6.6 wird das Klassendiagramm der Klasse `Geometry` und ihrer Derivate dargestellt. An ihm lässt sich der schematische Aufbau des Geometriemoduls ablesen. Die Klasse `Geometry` liefert einige virtuelle Methoden, die von Geometrieklassen, die dieses Interface verwenden, zu implementieren sind. Dazu zählen vor allem die Methoden `lowerLeft`, `upperRight` und `material`. Die Methoden `upperBound` und `lowerBound` geben einen Wert zurück, der in x, y, z Koordinate jeweils kleiner bzw. größer als das Supremum bzw. Infimum der Menge ist. Die Methode `material` dient zur Bestimmung, ob sich ein Punkt im Raum innerhalb eines Objekts befindet. Dabei gibt sie entweder ein entsprechendes Material aus der Enumerationsinstanz `Material` zurück, oder `None`, falls sich der Punkt nicht innerhalb der Geometrie befindet. Mengenoperationen wer-

den durchgeführt, indem mit der Klasse `SetOperation` dekoriert wird. Diese ist über den durchzuführenden binären Operationen `LogicalOperation`, `LowerOperation` und `UpperOperation` (template-)parametrisiert, um beliebige Verknüpfungen auf den übergebenen Mengen und ihren oberen und unteren Schranken durchführen zu können. Ferner können Projektionen einer Geometrie durch Dekorieren mit der Klasse `Transformation` erzeugt werden. Auch `Transformation` ist (template-)parametrisiert und erhält eine Projektionsfunktion sowie ihre Inverse. Es sind auch die gängigen Mengenoperationen Schnitt, Vereinigung und Differenz implementiert und über die Prozeduren `unify`, `intersect` und `difference` aus der Header-Datei `commonlib/geometry.h` erreichbar. Weiterhin gibt es Operationen für die Translation (`translate`), Rotation (`rotate`), Skalierung (`scale`) und die Erzeugung einer Sinusoberfläche (`ripple`). Es ist außerdem möglich, mit `XYSine` einen Sinus in XY-Richtung zu erzeugen sowie mit `CarveOut` einen Körper auszuhöhlen, wobei geprüft wird, ob genügend Rand vorhanden ist.

Es ist anzumerken, dass das Dekorieren von Geometrien zur Erleichterung des Speicher-Managements jeweils tiefe Kopien der dekorierten Hierarchie auf dem Heap anlegt, da der Speicherbedarf von Referenzen auf abstrakte Klassen nicht zur Compilezeit feststeht und verschachtelte Kompositionen sonst für den Aufrufer die direkte Erreichbarkeit aller Teilobjekte fordern würden.

6.3.2. Das Generator-Interface

Den Kern des Generators bildet die Klasse `GeneratorInterface`. Dabei handelt es sich um eine Klasse, die ein gemeinsames Interface für den SPH- und den MCA-Generator bereitstellt.

In Abbildung 6.7 wird ein Klassendiagramm der Klasse `GeneratorInterface` sowie ihrer ererbenden Klassen `MCAStates` und `SPHStates` gezeigt. `GeneratorInterface` beinhaltet dabei von SPH und MCA gemeinsam genutzte Objektparameter und Methoden, sowie Deklarationen virtueller Methoden, deren Spezialisierungen in den ererbenden Klassen implementiert worden sind. Ein Beispiel hierfür ist die Methode `init()`, die für MCA und SPH verschiedenartig implementiert ist, da sich die benötigten Simulationsparameter unterscheiden. Details zu den einstellbaren Simulationsparametern und allen anderen einstellbaren Größen können dem Benutzerhandbuch in Anhang C entnommen werden.

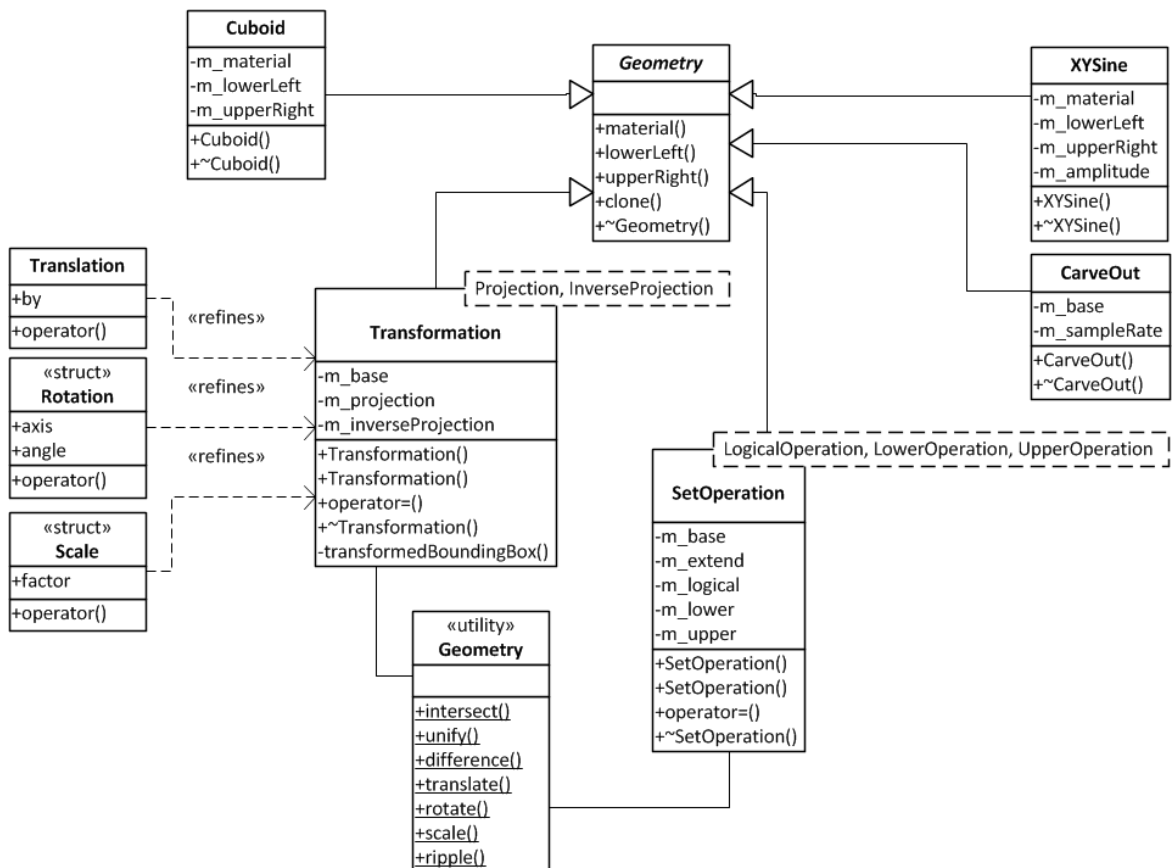


Abbildung 6.6.: Klassendiagramm der Szenengeometriemodellierung

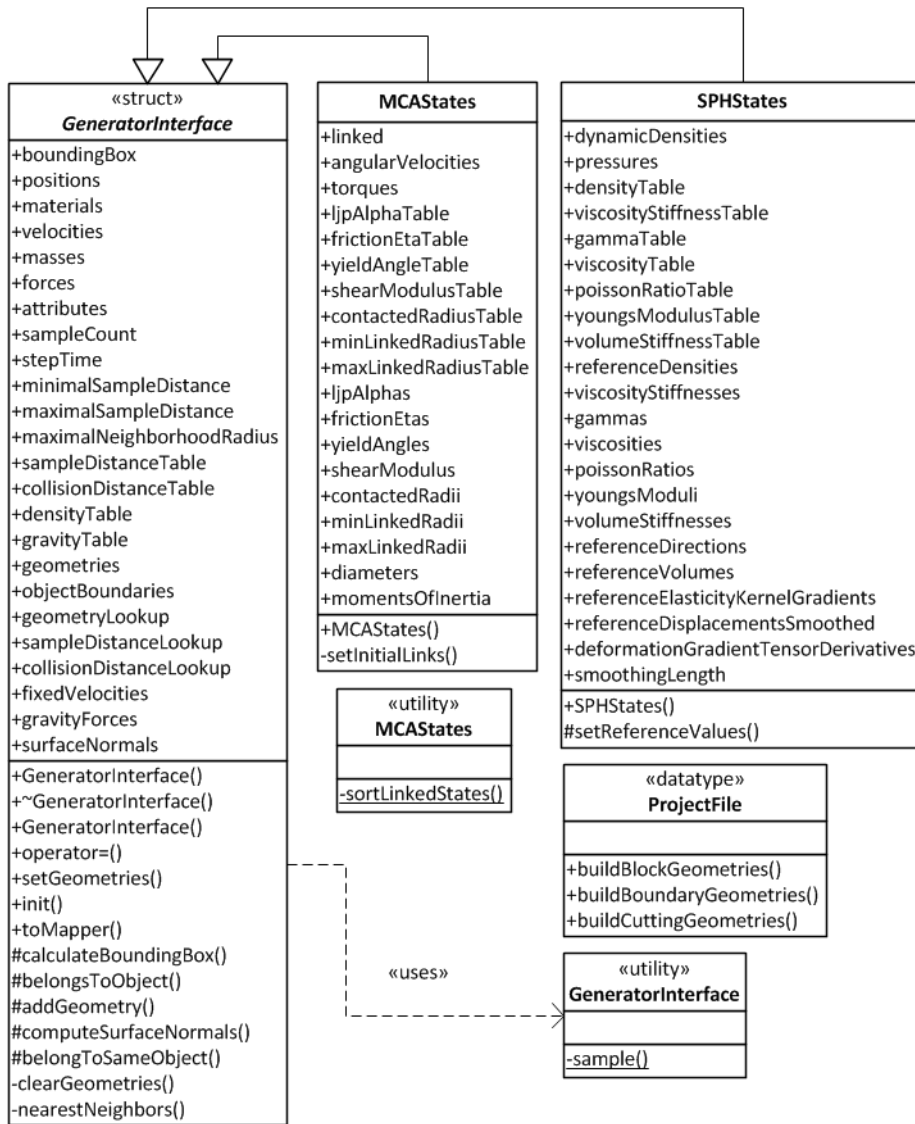


Abbildung 6.7.: Klassendiagramm zum Generator-Interface

6.3.3. Projektdateiparser

Für die Serialisierung der Daten ins JSON Format und das Parsen von JSON Daten wird die externe Bibliothek QJSON [Cas08] verwendet, die unter GNU-GPL Lizenz steht. Diese Bibliothek bietet die Möglichkeit eine Instanz einer eigenen Klasse aus einem JSON Objekt zu erzeugen. Die QJSON Bibliothek basiert auf der Verwendung der Klassen `QVariant`, `QVariantMap` und `QVariantList`.

Die Parserfunktionalität wird durch Methoden in mehreren `generatorlib`-Dateien, auf die nun im einzelnen Eingegangen wird, realisiert:

- `fooscene.h/.cpp`
In dieser Datei sind jeweils alle Getter und Setter der für die im Benutzerhandbuch beschriebenen Szenentypen nötigen Parameter zusammengefasst.
- `scenetypes.h`
Hilfsfunktionen zur Umwandlung der vom Parser gelieferten `QVariantMaps` in die im Programm genutzten Typen befinden sich in dieser Datei.
Durch diese Hilfsfunktionen wird vermieden, in allen zu serialisierenden Klassen `Q_PROPERTYs` für die zu speichernden Attribute zu definieren. Somit ist der für die (De-)Serialisierung zuständige Quellcode mehr vom Rest entkoppelt, als in der für QJSON üblichen Variante über `Q_PROPERTYs`. Exemplarisch ist in Listing 6.1 die Methode zur Umwandlung von `QVariantList` nach `MagnitudeMapType` aufgeführt.

```

1 template <class MagnitudeMapType>
2 MagnitudeMapType writeMagnitudeMap(const QVariantList& table) {
3     MagnitudeMapType result;
4     for (QVariantList::const_iterator i = table.begin(); i != table.end(); ++i) {
5         QVariantMap map = i->toMap();
6         for (size_t m1 = None + 1; m1 != MaterialCount; ++m1) {
7             for (size_t m2 = None + 1; m2 != MaterialCount; ++m2) {
8                 if (map["material1"].toString() == materialNames[m1] &&
9                     map["material2"].toString() == materialNames[m2]) {
10                    result.values[m1][m2] = map["value"].toDouble();
11                }
12            }
13        }
14    }
15    return result;
16 }

```

Listing 6.1: Typenumwandlung: `QVariantList` nach `MagnitudeMapType`

- `materials.h`
Die Definition simulierbare Materialien sowie das Festlegen ihrer für die Simulation relevanten Parameter finden in dieser Datei statt. Die an dieser Stelle definierten Default-Werte werden in der Simulation genutzt, falls sie nicht in der geladenen Projektdatei angegeben sind. Nähere Informationen bezüglich der Default-Werte befinden sich im Benutzerhandbuch in Anhang C.

- `projectfile.h`
In dieser Datei werden über die Methode `readSolverAndScene` und das `struct template ProjectFileHandle` der in der Projektdatei gesetzte Szenentyp ausgewertet und der passende Solver und eine Szene instanziiert.

6.4. Nachbarschaftssuche

Die Grundlage für die angestrebte Implementierung der Nachbarschaftssuche in der Klasse `neighborsearch` bilden die Gegebenheiten der Zielsetzung, siehe Kapitel 1. Weiterhin spielen die gewonnenen Erfahrungen der Projektgruppe während der Konstruktion eines SPH- und MCA-Prototypen in der Implementierung der Nachbarschaftssuche und den Anforderungen von SPH und MCA an eine effiziente Suche eine wichtige Rolle. Diese können dem Zwischenbericht entnommen werden. Die theoretische Betrachtung der Nachbarschaftssuche (vgl. Kapitel 3.6) ist deswegen zu diesem Zeitpunkt der Entwicklung im Vergleich weniger entscheidend für die Wahl eines Verfahrens für die Nachbarschaftssuche. Vor allem die Anforderungen von MCA und die Ergebnisse der während der Prototypphase in MCA eingesetzten alternativen Nachbarschaftssuche sind wichtige Gründe für Teiländerungen der Entscheidung.

Für den SPH-Prototypen wurde das Verfahren auf Basis von regulären Gittern (äquidistante Raumaufteilung) für die SPH-Implementierung als Nachbarschaftssuche im Prototypen favorisiert. Die Wahl erfolgte wegen der guten Unterteilung einer Szene, die durch die Objekte beschränkt ist. Ein Partikel muss nur dann neu einsortiert werden, wenn sich die Zellposition ändert. Ein wichtiger Punkt ist, dass kein Neuaufbau der Datenstruktur nötig ist. Die Zelle eines Partikels kann einfach errechnet werden und die Nachbarzellen sind direkt bestimmbar. Auf Basis der Datenstruktur des regulären Gitters stehen im Fall eines Performance- oder Speicherproblems verschiedene Möglichkeiten der Verbesserung zur Verfügung. Hashing wurde jedoch für die Implementierung von Festkörpern als nicht lauffzeitverbessernd eingestuft und ausgeschlossen.

Die MCA-Simulation verwendet eine eigene sortierungsbasierte Nachbarschaftssuche. Der Ausgangspunkt für diese Implementierung war eine dünnbesetzte, in Vektorform nachgehaltene, symmetrische Adjazenzmatrix, welche die in Kapitel 6.6 beschriebenen einfachen Iterationsschemata unterstützt. Der besondere Vorteil dieser Darstellungsform liegt in ihrem geringen Speicherverbrauch und der Möglichkeit, die Matrix spaltenweise parallel zu befüllen.

Die in MCA umgesetzte Nachbarschaftssuche wurde gegen eine Referenzimplementierung, die alle Positionspaare i, j vergleicht, auf randomisierten Punktwolken und verschiede-

nen Radii getestet. Sie erwies sich als korrekt und für den gegebenen Anwendungsfall in der Praxis als etwa um einen Faktor drei schneller als die professionelle kd-tree-Implementierung *FLANN* [Muj12].

Die Teiländerungen der Entscheidung sind weiterhin abhängig von der Strukturänderung der gesamten Implementierung, die eine Vereinheitlichung von SPH und MCA vorsieht. Dies bedeutet, dass die Nachbarschaftssuche beide Simulationen gleichermaßen bedienen muss. Grundsätzlich wurde wegen einer um Faktor zwei besseren Laufzeit der Regulären-Gitter-Methode bei 70000 Partikeln und gleicher Laufzeit bei 1000 Partikeln der Vorzug der Regulären-Gitter-Methode vor dem Suchverfahren der MCA Implementierung gegeben. Die Änderungen betreffen das Ergebnis, das eine dünnbesetzte, in Vektorform nachgehaltene, symmetrische Adjazenzmatrix ist. Weiterhin wird die in MCA beobachtete Laufzeitverbesserung durch Ausnutzen der Symmetrie der Abstände zweier Elemente in die ehemalige SPH-Suche integriert. Somit werden in den Objekten keine Informationen über deren Zelle mehr gespeichert und das Zellengitter wird nun sequentiell ausgewertet.

Im Folgenden werden die wichtigsten Methoden der Nachbarschaftssuche und die Datenstruktur, die zu Grunde liegt, beschrieben. Die beschriebenen Hilfsmethoden dienen dabei nur der internen Strukturierung der Methoden.

6.4.1. Datenstruktur

Die Datenstruktur ist ein dreidimensionales Feld von Zeigern auf Vektoren von Partikel-Indizes. Das dreidimensionale Feld wird als Zellengitter bezeichnet. Das Zellengitter ist vergrößerbar in einem auf Faktor 10 begrenzten Bereich in Bezug zur ursprünglichen Szene. Jedes Feld wird als Zelle angesehen und verwaltet die sie beinhaltenden Elemente in einer Liste, die aus Performanzgründen über einen Vektor realisiert wird. Die Liste selber wird über einen Zeiger gehalten, was die Methode *resize* gegenüber der Vorgängerversion beschleunigt. Die Größe einer Zelle entspricht dem Radius der SPH-Partikel oder dem *linkRadius* der MCA-Partikel. Zwei zusätzliche Vektoren dienen der Verwaltung der Partikel, die den Bereich der Zellengitter verlassen haben. Eine Liste beinhaltet die Partikel, die temporär den Zellenbereich verlassen haben und nach einem Ausführen der *resize*-Methode wieder in eine Zelle einsortiert werden könnten. Die zweite Liste verwaltet die Elemente, die nicht mehr ins Zellengitter einsortierbar sind, da eine Vergrößerung der Zellenbereiche über einen Faktor von 10 nicht vorhergesehen ist.

6.4.2. Initialisierung

Die Initialisierung erfolgt in der Methode *init*. Es wird aus den Dimensionen der Szene, die an die Initialisierung übergeben werden, eine *BoundingBox* des Zellenbereiches berechnet. Diese *BoundingBox* entspricht der Dimension der Szene, erweitert um die Größe einer Zelle in jede Raumrichtung. Aus der Größe des Zellenbereiches wird die

entsprechende Anzahl der Zellen für die drei Raumdimensionen berechnet und ein entsprechend großes dreidimensionales Feld erzeugt. Dazu wird die Größe des Zellenbereiches für jede Raumdimension durch die gewünschte Zellengröße geteilt. Zuletzt werden die beiden zusätzlichen Vektoren erzeugt und die Partikel in die Zellen einsortiert. Partikel, die bereits bei der Initialisierung außerhalb des Zellenbereiches liegen, werden in die dafür vorgesehene Liste einsortiert. Sollten Partikel außerhalb des Zellenbereiches liegen, wird bereits in der Initialisierung die *resize*-Methode ausgeführt. Zusätzlich wird in der Initialisierung die maximale Größe des Zellenbereiches festgelegt. Diese wird statisch mit Faktor 10 der BoundingBox des Zellenbereiches zur Initialisierungszeit berechnet.

6.4.3. Aktualisierung der Zelleninhalte

In der Methode `updatedCells` werden die Zellen der Partikel aktualisiert. Dies erfolgt durch eine sequentielle Überprüfung jedes Partikels in jeder Zelle. Partikel werden dabei neu einsortiert, wenn ihre Zelle nicht mehr gültig ist. Eine Berechnung der Zelle eines Partikels wird in der Hilfsmethode `calculateElementCell` durchgeführt. Dabei werden zur Bestimmung der Position die Raumkoordinaten des Elements, welche um die Position der Zellen-BoundingBox subtrahiert wurden, durch die Zellengröße geteilt. Sollten Partikel nicht einsortiert werden können, da ihre Position außerhalb des Zellenbereiches liegt, werden diese in einer der zusätzlichen Vektorlisten zwischengespeichert und die *resize* Methode wird im Anschluss an die Aktualisierung der Zellen ausgeführt.

6.4.4. Aktualisierung der Nachbarschaften

Die Methode `calculateNeighborhood` berechnet die Nachbarschaften und legt diese in einer SparseMatrix ab. Dazu wird zuerst die Methode `updatedCells` ausgeführt, um die räumliche Sortierung der Partikel in den Zellen zu aktualisieren. Im Anschluss wird sequentiell über die Zellen die gesamte Nachbarschaft bestimmt. Für jede Zelle wird für jedes Partikel die Nachbarschaft mit Hilfe von drei Hilfsmethoden berechnet. `checkOwnCell` berechnet die Nachbarn in der gleichen Zelle. `checkNeighborCellsNormal` und `checkNeighborCellsSpezial` bestimmen die Nachbarzellen. Beide Methoden verwenden intern `checkNeighborCell` für die Bestimmung der Nachbarn.

`checkNeighborCellsNormal` und `checkNeighborCellsSpezial` bestimmen die benötigten Nachbarzellen, wobei `checkNeighborCellsNormal` für Zellen aufgerufen wird, die sich nicht am Rand des Zellengitters befinden und `checkNeighborCellsSpezial` für Zellen, die sich am Rand befinden. Im Fall von `checkNeighborCellsNormal` entfällt der Vergleich, ob eine Zelle am Rand liegt. Fehlerhafte Zugriffe im dreidimensionalen Gitter, die zu einem Programmabsturz führen würden, können dort nicht auftreten. Diese Unterscheidung der Methoden funktioniert effizient, da die Index-Bereiche des dreidimensionalen Feldes bekannt sind und ein einfacher Vergleich für die Ausgangszelle ausreicht. `checkNeighborCellsSpezial` führt hingegen alle diese Vergleiche durch.

`checkNeighborCell` und `checkOwnCell` stellen als Hilfsmethoden die Nachbarschaften fest und übergeben diese an eine Hilfsfunktion, die die Distanz bestimmt und das Ergebnis in die `SparseMatrix` schreibt.

6.4.5. Aktualisierung der Datenstruktur

Die Methode `resizeCells` vergrößert das Zellengitter für den Fall, dass Partikel das Zellengitter verlassen. Dazu berechnet die Methode `calculateSceneResizeBoundingBox` zuerst die Vergrößerung des Zellengitters unter Beachtung der maximal möglichen Vergrößerung. Das Zellengitter wird um ganze Zellen vergrößert, daher können wegen der Zeigerstruktur, die die Vektorlisten hält, diese in ein neues, größeres Zellengitter übernommen werden. Somit entfällt die Neueinsortierung aller Partikel und die Neuerstellung der Datenstruktur wird auf das Hinzufügen von neuen Vektorlisten minimiert. Die Partikel, die das Zellengitter verlassen haben, können nun in diese ein- bzw. aussortiert werden, wenn eine Vergrößerung, die ihre Einsortierung erlauben würde, nicht möglich ist.

6.4.6. Ausblick

Eine parallelisierte Version der Suche wäre eine Möglichkeit, die hier verwendete Suche zu beschleunigen. Problematische Methoden bei einer Parallelisierung wären die `calculateNeighborHood` und die `updatedCells` Methode. Die anderen Methoden hingegen benötigen keine Synchronisierung von Teiloperationen.

Bei `calculateNeighborHood` ist der kritische Teil der Operation das Ablegen der gefundenen Nachbarschaften in der `SparseMatrix`, da dies einen Schreibzugriff darstellt, bei dem sichergestellt werden muss, dass nicht zeitgleich in die selbe Liste der Matrix geschrieben wird. Bei `updatedCells` besteht ein Synchronisationsproblem bei der Umsortierung eines Partikels. Die Vektorliste, aus der es entfernt wird, und die Vektorliste, in die es geschrieben wird, müssen beide gegen weitere Zugriffe gesichert werden.

Für MCA mit seinen im Vergleich zu SPH großen Zellengrößen würde es sich zusätzlich anbieten, die Zellengröße zu halbieren. Das Ausschlussverhalten der Nachbarschaftssuche kann so verbessert werden und die Anzahl der Abstandsberechnungen würde sich fast halbieren. Für SPH ist dies nicht übertragbar, da die Zellen bereits wenige bis keine Elemente enthalten und so nur leere Zellen generiert werden würden. Nachteilig wäre jedoch, dass die Suche nicht mehr allgemein wäre und eine dauerhafte Fallunterscheidung stattfinden würde. Bei der Aktualisierung der Nachbarschaften müssten mehr Zellen betrachtet werden. `checkNeighborCellsNormal` und `checkNeighborCellsSpezial` müssten für SPH und MCA eine Fallunterscheidung durchführen.

Eine weitere Möglichkeit zur Verbesserung der Laufzeit, die Erwähnung finden sollte, ist die Berechnung der Nachbarschaftssuche auf der GPU. Um Geschwindigkeitseinbußen

durch das Verschieben von Daten zwischen CPU- und GPU-Speicher zu verhindern, müssen der Solver und die Nachbarschaftssuche mit allen Berechnungen auf der GPU ausgeführt werden. Für eine GPU-Implementierung einer Nachbarschaftssuche eröffnen sich durch die statischen Größen von Feldern im Grafikkartenspeicher einige Probleme. Es kann nicht einfach für jede Zelle ein Vektor gefüllt werden, denn die Anzahl der Elemente einer Zelle muss bekannt sein. Für das Ergebnis der Nachbarschaftssuche muss ebenfalls bekannt sein, wie viele Nachbarn ein Partikel hat. Insgesamt wird eine angepasste Datenstruktur sowie spezielle Berechnungen zur Bestimmung ihrer genauen Größe benötigt. Das Problem der Zellen lässt sich durch eine angepasste Struktur lösen die nach [ZLZ10] im folgenden beschrieben wird. Es werden zwei Felder erstellt, eines für alle Zellen und eines für alle Partikel. Das Feld für die Partikel hat zwei Einträge, den Index des Partikels und die Zelle des Partikels. Dieses Feld wird mit einer parallelisierbaren Sortierung wie RadixSort aufsteigend nach den Zellen sortiert. Für jede Zelle wird im Feld für die Zellen abgespeichert, an welchem Index im Partikel-Feld sein erstes Partikel steht. Dieser Index ergibt zusammen mit dem Index der nächsten Zelle die Anzahl der Einträge einer Zelle. Abbildung 6.8 zeigt eine Darstellung der Datenstruktur.

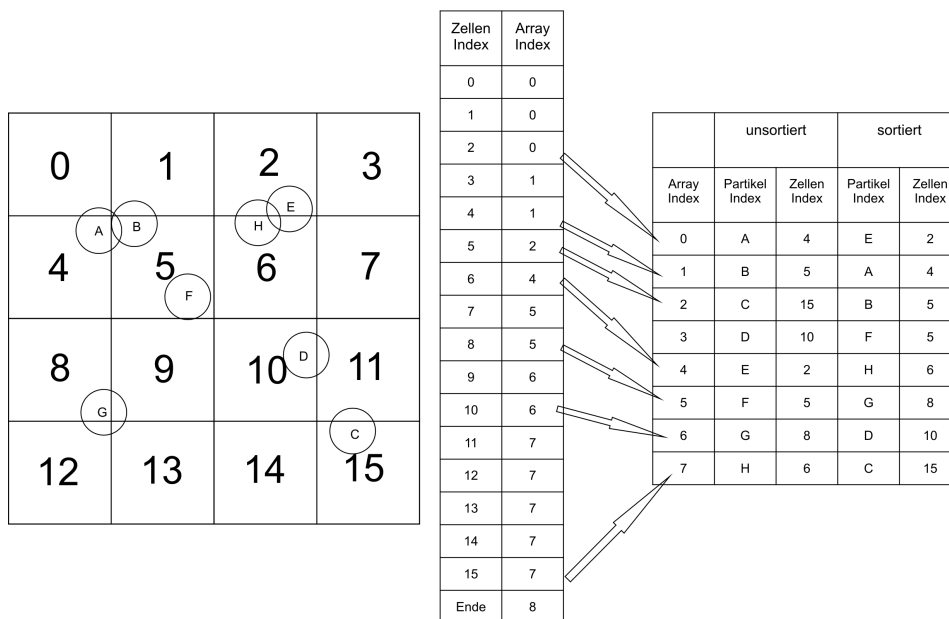


Abbildung 6.8.: Datenstruktur für GPU-Nachbarschaftssuche

Somit ist diese Datenstruktur statisch. Für eine Aktualisierung der Positionen von Partikeln in den Zellen wird für jedes Partikel parallel seine Zelle neu berechnet und danach das Feld neu sortiert. Für die Zellen kann anschließend linear der Index des ersten Elements der Zelle berechnet werden. Auf dieser Struktur kann die Symmetrie der Abstände der Partikel ausgenutzt werden. Eine Berechnung der Nachbarschaften geschieht in zwei Schritten. Im ersten Schritt muss bestimmt werden, wie viele Nachbarn

ein Partikel hat, um eine Datenstruktur für die Lösung aufzubauen. Die Datenstruktur für die Lösung kann ebenfalls aus zwei Feldern erzeugt werden. Ein Feld speichert für alle Partikel einen Verweis auf das zweite Feld, der den Index des ersten Nachbarn des Partikels angibt. Zusammen mit dem Index des nächsten Partikels im Feld ergibt sich die Anzahl der Nachbarn. Das zweite Feld speichert die nach den Zellen geordneten Nachbarn. Es reicht aus, einen Index des Nachbarpartikels zu speichern. Dies wird in Abbildung 6.9 dargestellt.

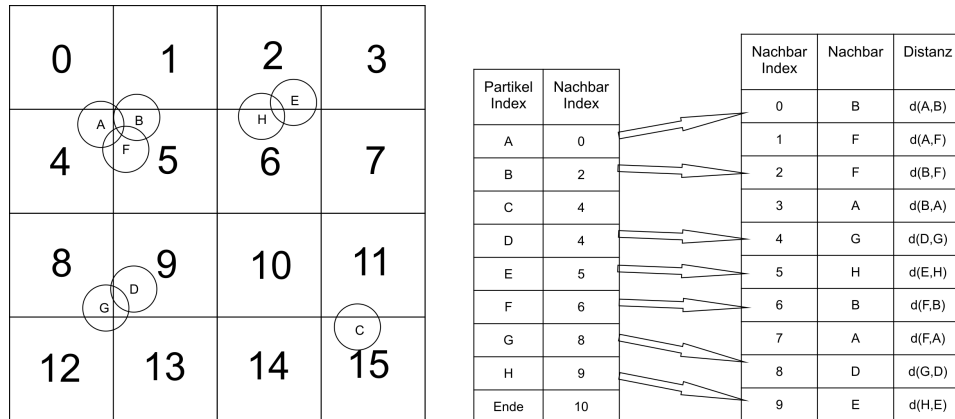


Abbildung 6.9.: GPU-Ergebnisstruktur einer Nachbarschaftssuche

In einer ersten parallelen Berechnung der Nachbarschaft wird für jedes Partikel im ersten Feld für jeden gefundenen Nachbar der Wert um eins inkrementiert. Diese Berechnung kann bereits unter Ausnutzung der Symmetrie stattfinden. In einem zweiten Schritt werden die Summen aufsummiert, der Wert von Partikel i ist die Summe aller inkrementell berechneten Werte seiner Vorgänger im Feld und kann als lineares aufsteigendes Summieren bezeichnet werden. Der letzte Wert ergibt dann die Größe des Lösungsfeldes, das erzeugt werden muss. Der Indexbereich der Nachbarn eines Partikels ist über das erste Feld bereits bekannt. In der zweiten ebenfalls parallelen (symmetrischen) Berechnung der Nachbarschaft kann nun anhand der Indexwerte aus dem ersten Feld für jede gefundene Nachbarschaft diese direkt abgelegt werden. Zusätzlich lässt sich der Speicherverbrauch wegen der Symmetrie halbieren, indem für eine Nachbarschaft nicht zwei Einträge abgelegt werden, sondern nur einer. Natürlich sollten die Distanzen bei der zweiten Berechnung mit abgespeichert werden.

6.5. Smoothed Particle Hydrodynamics

Das im Theoriekapitel 3.3 vorgestellte SPH-Konzept wird durch den SPH-Solver implementiert. Der SPH-Solver umfasst den Berechnungsteil der Simulationsschleife, in der alle für einen SPH-Schritt notwendigen Berechnungen durchführt und alle Partikeleigenschaften aktualisiert werden.

Der SPH-Solver beinhaltet im Detail:

1. Initialisierung
2. Berechnung der Zustandsvariablen
3. Berechnung der Kräfte
4. Berechnung der Kollision
5. Aktualisierung der Partikelposition und der Partikelgeschwindigkeit.

Der SPH-Solver ist eine Spezialisierung des generischen Solvers (siehe Abbildung 6.11). Da die Zeitintegration nicht spezifisch für SPH ist, wird diese nicht im SPH-Solver durchgeführt, sondern im generischen Solver.

6.5.1. Initialisierung

Bei der Initialisierung des SPH-Solvers werden lediglich zwei Schritte durchgeführt. Zum einen werden alle für die späteren Berechnungen benötigten Kernelfunktionen initialisiert und zum anderen wird Speicher für diejenigen Komponenten allokiert, welche bei der Berechnung von elastischer sowie plastischer Verformung eine Rolle spielen. Konkret wurde hier die Funktion W_{poly6} als Kernelfunktion zur Berechnung der Dichte gewählt, wie sie in Abschnitt 3.3.1 zu finden ist. Die Glättungsfunktion ist der Gradient der Funktion W_{spiky} (siehe Abschnitt 3.3.1), welcher bei der Berechnung der Druckkraft verwendet wird.

6.5.2. Berechnung der Zustandsvariablen

Die Zustandsgrößen der Simulation werden in der Methode `updateStateComponents()` durchgeführt. Als erstes wird dabei gemäß des im Abschnitt 3.3.4 vorgestellten Festkörpermechanismus überprüft, ob ein Riss im Material entstehen soll. Für ein Partikel i werden alle Referenznachbarn j daraufhin überprüft, ob sich dieser Nachbar zu weit entfernt hat. Ist dies der Fall, so wird diese virtuelle Verbindung zwischen beiden Partikeln gelöscht.

In den nächsten Schritten werden Druck und Dichte berechnet. Die Berechnung dieser beiden Größen erfolgt dabei nach den Formeln, die in Kapitel 3.3.2 erläutert wurden. Da bei der Berechnung der Dichte eines Partikels die Dichten der umliegenden Partikel

eine Rolle spielen und die Berechnung sequentiell auf den Partikeln erfolgt, muss auf die zeitliche Konsistenz der verwendeten Werte geachtet werden. Wenn ein Partikel i zu einem Zeitpunkt t die Dichte $\rho_i(t)$ besitzt und sich in der Nachbarschaft eines Partikels j mit Dichte $\rho_j(t)$ befindet, so darf zur Berechnung von $\rho_i(t+1)$ nicht die Dichte $\rho_j(t+1)$ herangezogen werden, auch wenn diese schon berechnet wurde. Stattdessen muss $\rho_j(t)$ verwendet werden. Die Einhaltung dieser Konsistenz wurde in der Implementierung mittels `dynamicDensity` gewährleistet. Bei diesem Verfahren wird bei einem Lesezugriff die Dichte des vorherigen Zeitschrittes zurückgegeben. Ein Schreibzugriff überschreibt die im aktuellen Schritt gültige Dichte. Nach Beendigung eines Schrittes der Zeitintegration werden die neugeschriebenen Dichten als aktuell gültige Dichten übernommen.

Nach der Berechnung von Druck und Dichte werden die Tensoren der elastischen sowie plastischen Verformung dem gegenwärtigen Zustand angepasst. Die Verformungen werden als Abweichungen der geometrischen Ausgangslage zwischen Paaren von Partikeln aufgefasst.

6.5.3. Berechnung der Kräfte

Die in der Funktion `updateForces()` berechneten Kräfte sind die Druckkraft, die Viskositätskraft sowie die Elastizitätskraft. Da die Kraftberechnungen unabhängig voneinander sind, reicht eine einzelne Iteration über die Partikel aus um alle Kräfte zu berechnen. Die Viskositätskraftberechnung kann beispielsweise für einzelne Partikel deaktiviert werden, indem die viskose Steifheit auf den Wert `VISCOSITY_ZERO` gesetzt wird. Ähnliche Mechanismen existieren auch für die anderen optionalen Kräfte.

Eine Ausnahme unter den Kräften stellen die Gravitationskraft und die Randpartikelkraft dar. Die Gravitationskraft ist keine interne Kraft und bedarf somit keiner Berechnung durch den SPH-Formalismus. Sie wird verfahrensunabhängig im generischen Solver berechnet und ist nicht Bestandteil des SPH-Solvers.

Die Randpartikelkraft ersetzt eine Kollisionsbehandlung und wird in der Funktion `updateCollision()` berechnet. Sie wird erst nach der Zeitintegration und der Aktualisierung der Positionen der Partikel aufgerufen. Sie korrigiert die Partikelpositionen gemäß des in Abschnitt 4.2 vorgestellten Ansatzes.

6.5.4. Optimierung durch Ausnutzen der Symmetrie

Sowohl bei der Berechnung der Zustandsgrößen als auch bei der Berechnung der Kräfte wurde versucht, eine möglichst hohe Performanz zu erreichen. Der SPH-Formalismus besitzt eine Symmetrieeigenschaft, die sich für diese Zwecke ausnutzen lässt.

Da für Abstände zwischen zwei Partikel i und j entsprechend $|\vec{x}_{ij}| = |\vec{x}_{ji}|$ und für den Richtungsvektor $\vec{x}_{ij} = -\vec{x}_{ji}$ gelten, gilt ebenfalls $W(x_{ij}, h) = W(x_{ji}, h)$ und $\nabla W(x_{ij}, h) =$

$-\nabla W(x_{ji}, h)$, da die Glättungsfunktionen W immer symmetrisch gewählt wurden. Bei der Berechnung einer Zustandsgröße für ein Partikel i ist der Einfluss des Partikels j auf Partikel i genau so groß wie der Einfluss von Partikel i auf Partikel j und kann direkt mitberechnet werden. Bei den Kraftberechnungen werden Gradienten von Glättungsfunktionen verwendet. Somit muss die Richtungsabhängigkeit beachten werden. Die Kraft, die Partikel i auf Partikel j ausübt, muss demnach das umgekehrte Vorzeichen besitzen wie die Kraft, die Partikel j auf Partikel i ausübt.

6.5.5. PCISPH

Das in Abschnitt 3.3.3 theoretisch erläuterte Konzept von PCISPH findet keine praktische Verwendung in der Implementierung. Es wurde vermutet, dass eine Implementierung von PCISPH durch Vermeidung einiger Probleme der regulären Druckberechnung eine Verbesserung der Simulationsergebnisse liefert. Eine Implementierung des PCISPH-Algorithmus führte allerdings zu keiner phänomenologisch erkennbaren Verbesserung der Simulation und wurde deshalb nicht in die Software übernommen.

6.6. Movable Cellular Automata

Bei der Initialisierung der MCA-Simulation wird die Funktion `init` des Solvers (Abschnitt 6.1.3) verwendet, wobei auch der MCA-Solver eine Spezialisierung des generischen Solvers ist (siehe Abbildung 6.11). Zusätzlich wird die Vektorgröße für die Winkelbeschleunigung an die Partikelanzahl angepasst.

Zustandsberechnung

Die Berechnung der Zustandsgrößen findet in der Funktion `Step` des Solvers statt, welche von der MCA-Variante überschrieben wird. Im Folgenden wird detaillierter auf diese Funktion eingegangen. Die Funktion `Step` führt im wesentlichen zwei Berechnungen durch. Zum einen werden die Kräfte neu berechnet, zum anderen werden die neu berechneten Werte für die Umsetzung genutzt. Eine genaue Betrachtung der Kraftberechnung führt zu den wichtigsten Funktionen `collisions`, `updateLinked` und `updateForcesAndTorques`. Die Funktion `collisions` führt die Kollisionsbehandlung, wie in Abschnitt 3.4.3 beschrieben, durch. Hierbei wird die einwirkende Geschwindigkeit im Falle einer Kollision, also wenn der Abstand von i und j kleiner als der `contactedRadii` für i und j ist, durch eine entsprechende Gegengeschwindigkeit ersetzt.

Ohne diese Kollisionsbehandlung haben sich die Werkstückpartikel bei einem Werkzeug, das sich durch das Werkstück bewegt, unrealistisch komprimieren lassen (siehe Abbildung 6.10).

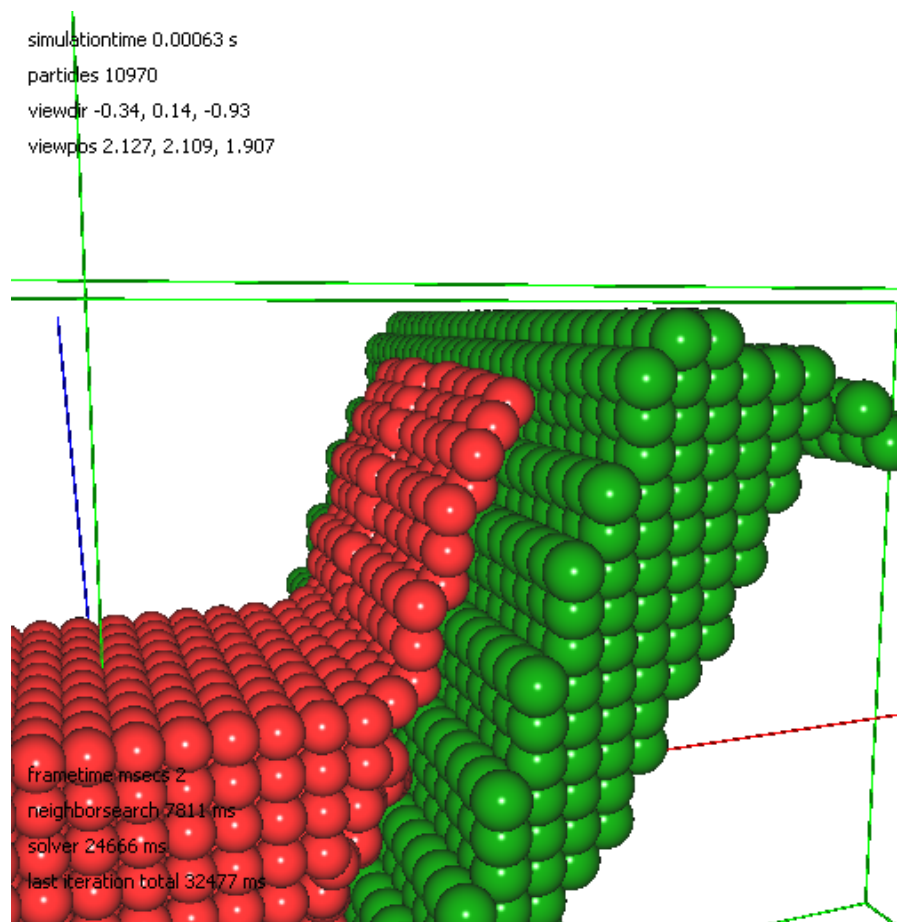


Abbildung 6.10.: MCA-Cutting ohne Kollisionsbehandlung

Im nächsten Schritt wird die einwirkende Gravitationskraft mit Hilfe der Funktion `calculateGravityForce` des Solvers berechnet. Die Funktion `updateLinked` prüft für die Automaten i und j , ob die Entfernung zueinander eine Änderung des Koppelungsstatus notwendig macht. Waren beide vorher verbunden, wird diese Verbindung entfernt, wenn `rMaxlinked` überschritten wurde. Zusätzlich wird der Deformationswinkel zwischen den beiden Automaten i und j auf 0 gesetzt. Wenn i und j vorher nicht verbunden waren und nun der Abstand kleiner als `rMinlinked` ist, werden sie miteinander verbunden. Treffen beide Fälle nicht zu, ändert sich der Linkedstatus nicht. Bei dieser Berechnung wird die Symmetrie der Verbindung der beiden Automaten i und j ausgenutzt, um die Speicherung in einer Sparse Matrix durchzuführen, was die Performanz verbessert. Auch bei der Speicherung des Deformationswinkels erweist sich dies als Vorteil.

Die eigentliche Kräfteberechnung findet in der Funktion `updateForcesAndTorques` statt. Dabei wird unterschieden, ob die beiden Automaten i und j verbunden oder kontaktiert sind. Sind sie kontaktiert, wird neben den Kräften für `linked` auch noch die Reibungskraft (Formel 3.60) berechnet. Im Fall einer Verlinkung werden die Kräfte `ljpForce`

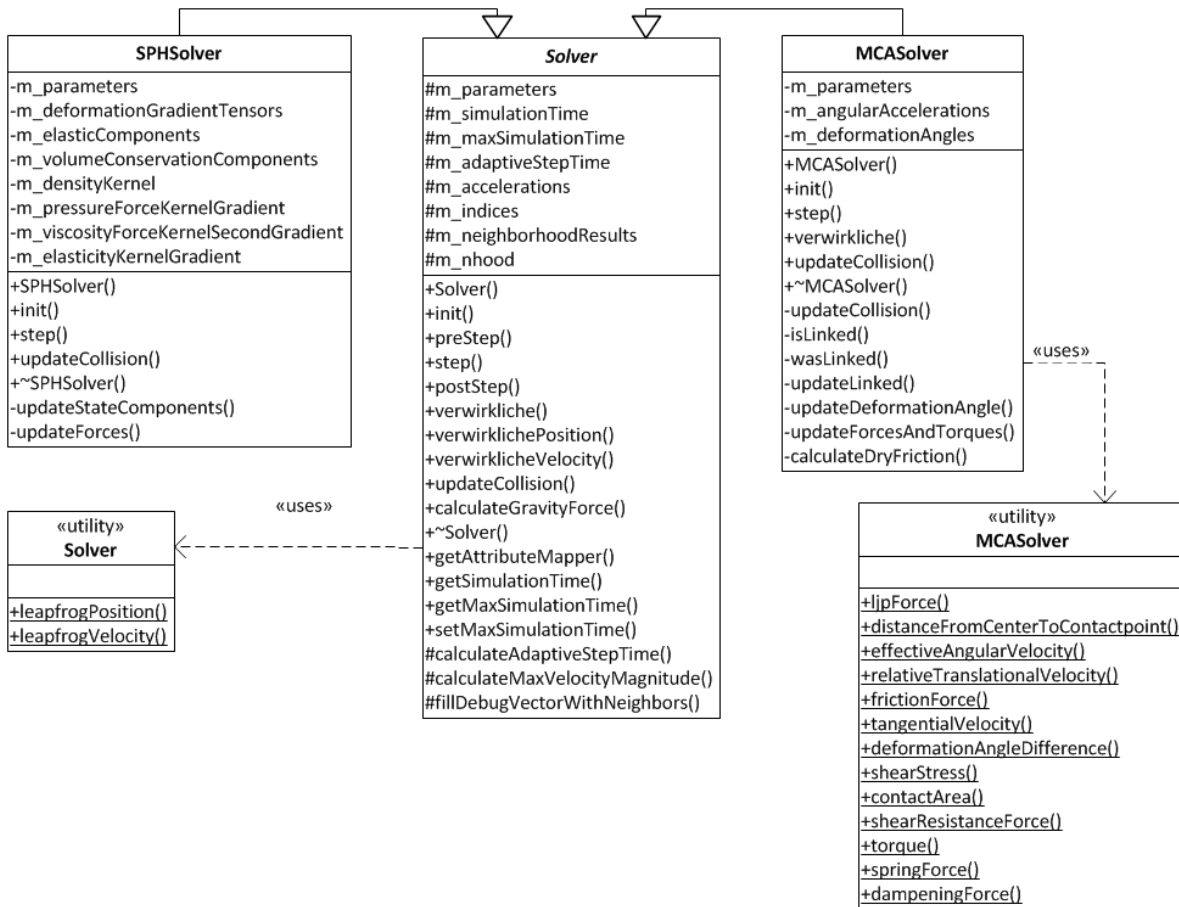


Abbildung 6.11.: Klassendiagramm zum SPH- und MCA-Solver

(Formel 3.57) und `shearResistanceForce` (Formel 3.62) sowie der Deformationswinkel `updateDeformationAngle` zwischen den Automaten i und j (Formel 3.61) und das Drehmoment für den Automaten i `torque` (Formel 3.67) berechnet.

Im Anschluss wird die Trockenreibungskraft nach der im Theoriekapitel vorgestellten Formel 3.64 berechnet und der Kräftesumme von i hinzugefügt. Zum Abschluss wird noch die Torsion berechnet, die durch die Trockenreibungskraft entsteht und zu der Torsion von i hinzuaddiert. Analog wird mit j verfahren.

Ein wesentlicher Unterschied zwischen dem MCA-Solver gegenüber dem SPH-Solver befindet sich bei der Positions- und Geschwindigkeitsberechnung. Bei MCA wird nach der Berechnung aller Kräfte die neue Position berechnet. Im Anschluss wird die Funktion `verwirkliche` ausgeführt, welche für MCA, im Vergleich zu der Funktion des allgemeinen Solvers (Abschnitt 6.1.3), um die Berechnung der neuen Winkelgeschwindigkeiten erweitert wurde. Diese erfolgt ebenfalls mit dem Leapfrog-Integrationschema [Kie11].

6.7. GPU-Solver

Die Evaluation des Prototyps zeigte bereits, dass die verwendeten Simulationsverfahren sehr viel Rechenzeit selbst auf aktuellsten Rechnern benötigen, weshalb eine Portierung auf ein GPGPU-Framework, mit der GPU als Beispiel für eine Many-Core-Architektur, in Betracht gezogen wurde. Bei einem Vergleich, unabhängig von einer zu lösenden Aufgabe, der sogenannten FLOPS (Floating Point Operations Per Second) zwischen einer aktuellen CPU mit einer Grafikkarte der etwa gleichen Generation, so ergibt sich theoretisch ein deutliches Verbesserungspotential durch die Portierung einer Berechnung auf die Grafikkarte. Zum Beispiel hat ein Intel i7-2600k nach Herstellerangaben eine Leistung von 108 GFLOPS [Cor12]. Eine Geforce 560 Ti leistet zum Vergleich 1264 GFLOPS [San11]. Unter Vernachlässigung architektureller Unterschiede wäre damit eine Leistungsverbesserung um den Faktor 10 möglich. Abhängig von dem zu lösenden Problem kann dieser Faktor jedoch stark in beide Richtungen abweichen. So sind Grafikkarten besonders gut für stark datenparallele Anwendungen geeignet, bei denen die Transferkosten weit unter den Berechnungskosten liegen. Die gewählten Simulationsverfahren MCA und SPH erfüllen genau diese Bedingung, wobei SPH bereits von Anderen auf die Grafikkarte portiert wurde [HKK07].

Die Möglichkeit, die Lösungsverfahren für MCA und SPH auf ein GPGPU-Framework wie zum Beispiel OpenCL zu portieren, wurde eingehend untersucht. Aufgrund von Zeitmangel konnte jedoch nicht rechtzeitig eine funktionierende Implementierung beendet werden. Aus diesem Grund erläutert dieser Abschnitt die Erfahrungen, welche bei der Teilportierung gesammelt wurden, welche Limitierungen OpenCL hat und wie diese sich auswirken sowie die architekturellen Änderungen, welche am Solver- und Workerinterface und am Gesamtprogramm nötig wurden.

Zu Beginn der Portierung wurde die Anforderung aufgestellt, dass jeder unnötige erneute Datentransfer zwischen Grafikkarte und normalem RAM zu vermeiden ist. Aus diesem Grund wurde die Lösung verfolgt, sogenanntes OpenGL-OpenCL-Interoperability zu nutzen, damit die Visualisierung ohne Kopieren der Berechnungsergebnisse von der Grafikkarte auf die CPU und wieder zurück direkt diese nutzt und visualisiert. Dies hat zur Folge, dass die sonst klare Schichtentrennung zwischen Berechnungsthread und Renderthread aufgeweicht wird. Das Teilen der Daten zwischen OpenGL und OpenCL erfolgt über die Erstellung und Initialisierung von Vertex Buffer Objects (VBOs) im Renderthread und Weitergabe dieser an den OpenCL-Berechnungsthread, welcher die VBO-IDs dazu nutzen kann, Wrapperbuffer (CL-GL-Buffer) zu diesen VBOs zu erstellen, welche in OpenCL direkt angesprochen werden können. Das Problem dabei ist, dass ein sich gegenseitig ausschließender Zugriff auf die VBOs gewährleistet werden muss, also entweder nutzt in einem Moment der Renderer die VBOs zum Rendern oder der OpenCL-Berechnungsthread diese für eine Berechnung. Aus diesem Grund wurde ein Doublebuffersystem entworfen, so dass OpenCL immer auf einer Kopie der Daten der VBOs arbeitet und nach Abschluss einer Berechnung blockierend unter Nutzung einer

geteilten Mutex, welche den Renderer blockieren kann, das Bufferupdate durchführt. Dadurch ist ein Locking auf der Renderermutex nur zum Zeitpunkt der Aktualisierung der Daten für den Renderer erforderlich. OpenCL hat eine eigene API zur Gewährleistung exklusiven Zugriffs auf CL-GL-Buffer, welche auch zwingend benutzt werden muss. Dies erzwingt, dass OpenCL auf sein eigenes Set von VBOs und einem eigenen OpenGL-State arbeitet, während der Renderer aktuelle Daten unabhängig vom OpenCL-Thread darstellen kann, da OpenCL für das Acquire & Release von Interoperabilityobjekten Zugriff auf einen OpenGL-Renderkontext benötigt. Dieser zweite Renderkontext im OpenCL-Thread ist per OpenGL-Context-Sharing mit dem ersten Rendererkontext im Rendererthread verbunden.

Bei dem Versuch, die implementierten Lösungsverfahren (zunächst nur SPH) auf der Grafikkarte möglichst effizient zu implementieren, zeigten sich gravierende Limitierungen im Standard OpenCL. Die Sprache OpenCL ist defakto C99, jedoch mit diversen Einschränkungen, welche weitreichende Folgen für Codequalität und Portabilität — eins der von der Khronos Group erklärten Ziele von OpenCL — nach sich ziehen. Zu diesen Limitierungen gehören:

1. Arraygrößen müssen compile-time constant sein
2. Rekursion ist nicht erlaubt
3. Funktionspointer sind eine selten implementierte Herstellererweiterung
4. Kein "#include" → entweder massive Codeduplizierung oder Stringtemplate für Code
5. Schwache Standardisierung von Fehlermeldungen
6. CL-GL-Interop ist eine Herstellererweiterung ohne Fallback
7. Keine Funktionsüberladung
8. Keine automatische Überladung für Funktionen mit lokalen vs. globalen Variablen
9. Keine Standardisierung von Kernelcompilerparametern
10. Simples printf ist eine Herstellererweiterung
11. Nur sehr rudimentäres Debugging

Während die Punkte 2-4 und 5-8 nur störend sind, die Entwicklung verlangsamen und fehleranfälliger machen, führt Punkt 1 zu einem schwierigen Abwägungsproblem. Dynamische Strukturen sind damit in OpenCL im Kernelkontext unmöglich. Um also dynamische Strukturen zu erstellen, muss ein Mehrfachdurchlaufansatz entworfen werden, bei dem in einem Durchlauf die Größen des erzeugten Outputs bestimmt werden und beim zweiten Durchlauf das Befüllen der Datenstruktur erfolgt. Dies kann zum Beispiel eine Verdoppelung der Laufzeit bedeuten. Alternativ zum Mehrfachdurchlaufansatz

besteht die Möglichkeit der Nutzung von Worst-Case-Abschätzungen, welche jedoch häufig unpraktikabel sind. Das Nachbarschaftsproblem führt damit zum Beispiel zu einer Worst-Case-Abschätzung von $n^2 - n$ und ist damit sehr schnell viel zu groß für den begrenzten RAM einer Grafikkarte. Weitere Ansätze der Beschränkung des Outputs, zum Beispiel der Begrenzung der maximalen Anzahl möglicher Nachbarn, sind ebenso denkbar, reduzieren aber die Genauigkeit des Ergebnisses und die damit verbundene Priorisierung von Outputgrößen ist für sich eine eigene Problemstellung. SPH und MCA erfordern jeweils eine Sparsematrix, jeweils für Referenznachbarn bzw. Linkstatus. Dieses Problem konnte bis zum Implementierungsstopp der Projektgruppe nicht zufriedenstellend gelöst werden.

Zu den weiteren Kritikpunkten an OpenCL gehören insbesondere Punkt 5 und 9. So kam es vor, dass mehrere völlig unterschiedliche Anwenderfehler zu der gleichen Fehlermeldung "CL_OUT_OF_RESOURCES" der NVIDIA-OpenCL-Implementierung führten. Desweiteren gibt es keine Standardisierung der Kernelcompilerparameter (zum Beispiel "-g" zum Erzeugen von GDB-kompatiblen Debuginformationen), wodurch eine Anwendung per Design zu sehr an Spezifikationen bestimmter Hardware gekoppelt wird. Hier besteht bei OpenCL in vielen Bereichen noch massiver Verbesserungsbedarf.

7. Auswertung

In diesem Kapitel folgt eine Auswertung der während der zwei Semester erarbeiteten Ergebnisse. Neben allgemeiner gehaltenen Parametertests zur Suche von nachvollziehbarem Verhalten bezüglich physikalischer Kräfte wie Dichte, Druck u.a. wurde auch generelles physikalisch simulierbares Verhalten, wie zum Beispiel Elastizität und Plastizität, getestet. Als wichtiger Bestandteil der Analyse wird im letzten Teil die Auswertung des Spanbildungsverhaltens diskutiert. Anschließend werden die erzielten Ergebnisse auf Richtigkeit und Anwendbarkeit bewertet.

Aufgrund der unterschiedlichen Basis der beiden gewählten Simulationstechniken erfolgt eine Auswertung und Bewertung in diesem Kapitel weitestgehend voneinander getrennt. Die erzielten Ergebnisse sind also explizit auf ein Verfahren bezogen und werden erst am Ende des Kapitels in Bezug zur Spanbildung gegenübergestellt.

7.1. MCA-Testfälle

Dieses Kapitel beinhaltet die prinzipiell simulierbaren Eigenschaften für die Simulationstechnik MCA. Dabei ist für einige Eigenschaften eine Reihe von Versuchen aufgeführt, um geringe Variationen des Verhaltens zu simulieren und ggf. auf verschiedene Parameter zurückzuführen. In den folgenden Abschnitten wird auf die Eigenschaften Festkörperstabilität in Ruhelage und unter Krafteinwirkung und die Bruch- und Drehungseigenschaften am Beispiel eines Würfels eingegangen. Im Abschluss wird ein Stab in Ruhelage bei verschiedenen Parametereinstellungen betrachtet.

7.1.1. Ruhelage eines Würfels unter Gravitation auf festem Boden

Ein grundlegender Test der Festkörpereigenschaften des simulierten Materials ist ein Würfel, der auf einem festen Boden liegt und hierbei stabil bleiben soll. Hierbei ist interessant, ob der Würfel die ganze Zeit über stabil auf dem Boden liegt oder sich verformt. Für die Simulation wird eine `MCA_BLOCK`-Szene verwendet. Der Würfel besteht aus dem zu untersuchenden Material und der Boden ist fest.

Als feste Parameter gelten die Maße des Würfels, dieser besteht aus 1000 Partikeln und hat eine Kantenlänge von 0,015 m. Der Boden, der aus 3380 Partikeln besteht, hat eine Höhe von 0,005 m. Eine Simulation mit insgesamt 4380 Partikeln wurde gewählt, damit

verwertbare Simulationsergebnisse bei einem Zeitschritt von $1 \cdot 10^{-7}$ s schon nach 12 Stunden Laufzeit vorlagen. Diese Simulation wurde auf einem Intel Core2 Duo T6500 mit 2,1 GHz und 4 GB Arbeitsspeicher durchgeführt.

Der Würfel soll ein stahlähnliches Verhalten zeigen, hierfür wurden die wesentlichen Parameter wie folgt gewählt:

Parameter	Formelzeichen	Wert
shearModulus(Stahl)	G_i	$79,3 \cdot 10^9$ Pa
yieldAngle(Stahl/Stahl)	γ_{ijY}	$1,7647 \cdot 10^{-4}$
ljpAlpha(Stahl/Stahl)	α_{ij}	$2 \cdot 10^{-3}$ J
frictionEta	η_{ij}	$1,5 \cdot 10^{-5}$
frictionMus	μ_{ij}	0,0

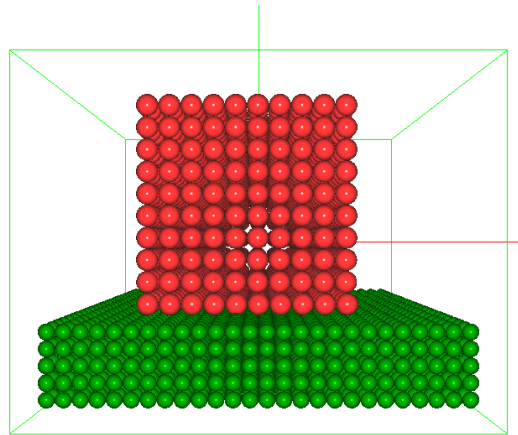


Abbildung 7.1.: Ruhelage: Nach einer Simulationszeit von 0,1462 s

Der Wert für shearModulus(Stahl) ist der Wert für das Material Stahl. Die Suche nach geeigneten Simulationsparametern, um einen stabilen Festkörper simulieren zu können, hat sich als schwierige Aufgabe herausgestellt. Wird der Wert für yieldAngle erhöht, wird der Würfel instabil, wird er niedriger gewählt, hat dieser kaum noch Einfluss auf das Simulationsverhalten. Die Werte der Reibungsparameter frictionEta und frictionMu wurden so gewählt, weil sich in Versuchsreihen herausgestellt hat, dass im Zusammenspiel mit den anderen Parametern ein stabiles Festkörperverhalten erreicht wird.

In Abbildung 7.1 wird gezeigt, dass es mit diesem Parametersatz keine Veränderung gibt, wie es bei einem stabilen Festkörper aus Stahl zu erwarten ist. Diese langanhaltende Stabilität wurde auch bei längerer Simulationsdauer beobachtet.

7.1.2. Aufprall eines Würfels auf festem Boden unter Gravitation

Ein wichtiger Test der Festkörpereigenschaften des simulierten Materials ist ein Würfel, der aus einer geringen Höhe fällt. Hierbei ist das Materialverhalten beim Aufprall bzw. danach bis zu einem Ruhezustand interessant. Für die Simulation wird wieder eine MCA_BLOCK-Szene verwendet.

Versuch 1 In diesem Versuch soll ein Würfel nach dem Aufprall stabil bleiben und ein stahlähnliches Verhalten zeigen. Hierfür wurden die wesentlichen Parameter wie folgt gewählt:

Parameter	Formelzeichen	Wert
shearModulus(Stahl)	G_i	$79,3 \cdot 10^9$ Pa
yieldAngle(Stahl/Stahl)	γ_{ijY}	$1,7647 \cdot 10^{-4}$
ljpAlpha(Stahl/Stahl)	α_{ij}	$2 \cdot 10^{-3}$ J
frictionEta	η_{ij}	$1,5 \cdot 10^{-5}$
frictionMus	μ_{ij}	1,0

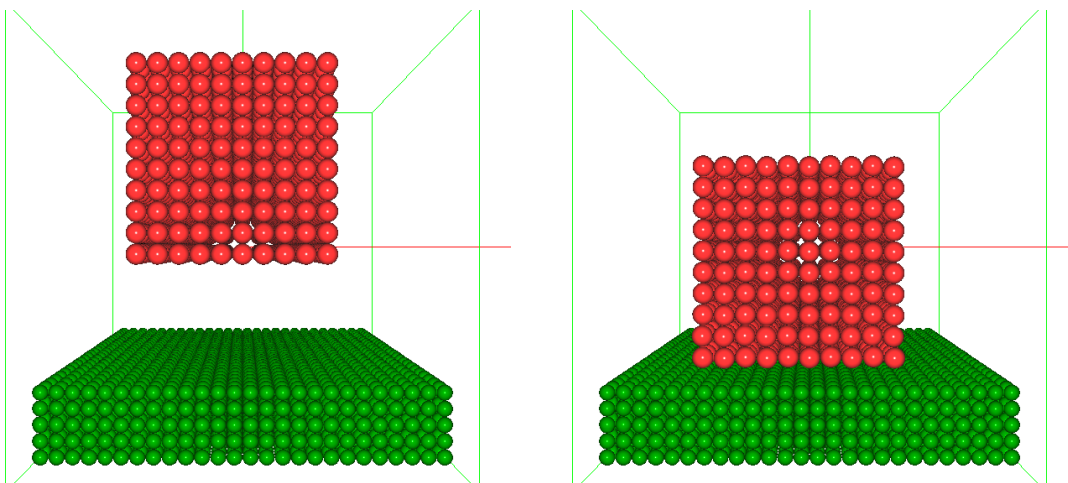


Abbildung 7.2.: Fallversuch 1: Ausgangssituation und nach einer Simulationszeit von 0,1s

Da bei diesem Versuch der Würfel ein stabiles, stahlähnliches Verhalten aufweisen soll, wurden die Parameter wie zuvor bei Versuch 7.1.1 gewählt, zusätzlich wird die Trockenreibungskraft benutzt.

Hierbei (Abb. 7.2) hat sich gezeigt, dass sich mit diesem Parametersatz ein stabiler Festkörper simulieren lässt, der auch bei einem Aufprall stabil bleibt und keine sichtbaren Veränderung aufweist, wie es bei einem Würfel aus Stahl zu erwarten ist. Die Laufzeit dieser Simulation betrug 9 Stunden.

Versuch 2 In diesem Versuch soll sich der Würfel nach dem Aufprall verformen und ein realistisches Verhalten aufzeigen. Zu diesem Zweck wurden die wesentlichen Parameter wie folgt gewählt:

Parameter	Formelzeichen	Wert
shearModulus(Stahl)	G_i	$79,3 \cdot 10^2 \text{ Pa}$
yieldAngle(Stahl/Stahl)	γ_{ijY}	$6,5 \cdot 10^{-10}$
ljpAlpha(Stahl/Stahl)	α_{ij}	$9,5 \cdot 10^{-20} \text{ J}$
frictionEta(Stahl/Stahl)	η_{ij}	$5,75 \cdot 10^{-8}$
frictionEta(Stahl/Boden)	η_{ij}	$-2,75 \cdot 10^{-15}$
frictionEta(Boden/Stahl)	η_{ij}	$-2,75 \cdot 10^{-15}$
frictionMus(Stahl/Stahl)	μ_{ij}	0,8

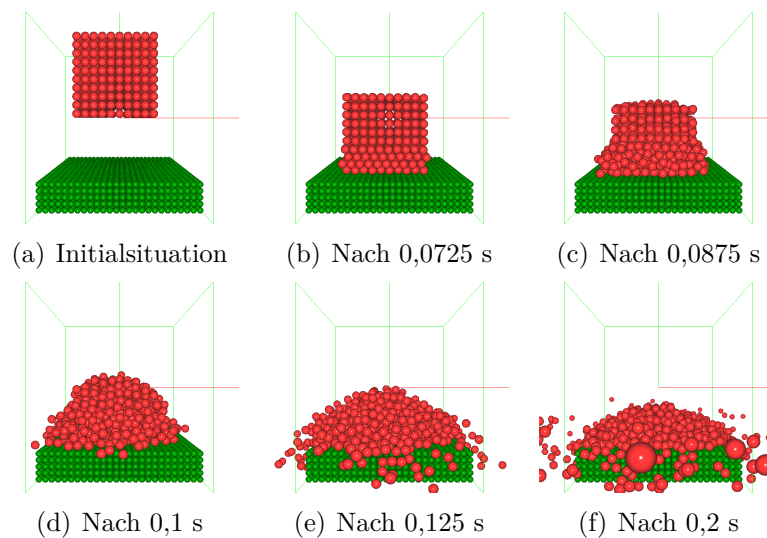


Abbildung 7.3.: Fallversuch 2

Die einzelnen Parameter wurden so eingestellt, dass deren Einfluss auf die Berechnung der Kräfte vergleichsweise gering ist. Einzig frictionMu wurde mit 0,8 so gewählt, dass die Trockenreibungskraft den größten Einfluss hat.

In Abbildung 7.3 wird das Verhalten des Würfels zu verschiedenen Simulationszeitpunkten abgebildet. Der Würfel verschiebt sich kurz nach dem Aufprall ebenenweise, danach verhält er sich wie eine gallertartige Masse und am Ende bleibt ein Teil als Partikelhaufen auf dem Boden liegen, während der Rest zur Seite fließt. Die Laufzeit dieser Simulation betrug 19 Stunden.

Zusammenfassend kann gesagt werden, dass der Würfel sich wie eine zähflüssige Masse verhält. Durch diesen Versuch wird eine mögliche Eignung für eine Simulation von Flüssigkeiten gezeigt, dies ist jedoch nicht das Ziel der PG, womit eine weitere Untersuchung nicht durchgeführt wurde.

Versuch 3 Bei dem Versuch einen gummiartigen Würfel zu erzeugen, entstand ein sehr merkwürdiges Materialverhalten, welches mit diesen Parametern erzeugt wurde:

Parameter	Formelzeichen	Wert
shearModulus(Stahl)	G_i	$79,3 \cdot 10^9$ Pa
yieldAngle(Stahl/Stahl)	γ_{ijY}	$1,7647 \cdot 10^{-4}$
ljpAlpha(Stahl/Stahl)	α_{ij}	$2 \cdot 10^{-3}$ J
frictionEta(Stahl/Stahl)	η_{ij}	$1,5 \cdot 10^{-5}$
frictionEta(Stahl/Boden)	η_{ij}	$1,5 \cdot 10^{-6}$
frictionEta(Boden/Stahl)	η_{ij}	$1,5 \cdot 10^{-6}$
frictionMus(Stahl/Stahl)	μ_{ij}	1,8

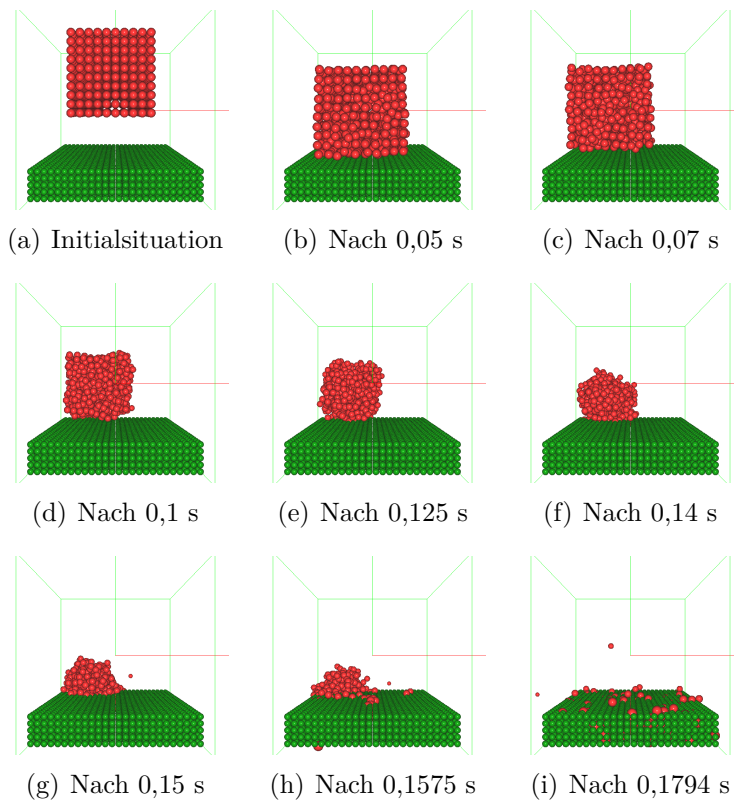


Abbildung 7.4.: Fallversuch 3

Die Parameter sind ähnlich zu Versuch 7.1.2 gewählt worden, es wurde nur die viskose Reibung zwischen Boden und Würfel um den Faktor 10 reduziert und die Trockenreibung auf 1,8 geändert, um die Stabilität des Materials etwas zu verbessern. Bei diesem Versuch (Abb. 7.4) verformt der Würfel sich bereits im Fall vor dem Aufprall. Nach der Kollision mit dem Boden wandert der Würfel in die hintere linke Ecke und zieht sich hierbei

immer mehr zusammen. Dies wird solange fortgesetzt, bis die einzelnen Partikel durch die Kollisionsbehandlung so schnell beschleunigt werden, dass der Würfel auseinander reißt und die einzelnen Partikel sogar in den Boden eindringen. In mehreren Versuchen konnten keine guten Parameter für ein gummiähnliches Material gefunden werden. Falls passende Parameter existieren, muss mehr Zeit in eine Suche investiert werden, um diese zu finden.

Der Effekt, dass Partikel durch hohe Geschwindigkeit in den Boden eindringen, kann verringert werden, wenn ein kleinerer Zeitschritt für diesen Versuch gewählt wird. Hierbei wäre besonders die Nutzung eines adaptiven Zeitschrittes hilfreich. Bei einer Zeitschrittgröße von $1 \cdot 10^{-7}$ s betrug die Simulationslaufzeit 3 Tage. Aus diesem Grund und weil sich nicht das gewünschte gummiartige Verhalten gezeigt hat, wurde hierzu keine weitere Simulation durchgeführt, die bei einem Zeitschritt von $1 \cdot 10^{-8}$ s etwa 25 Tage dauern würde.

Allgemeiner kann gesagt werden, dass aufgrund der hohen Komplexität der Parametersuche ein Finden von guten Parametern für ein bestimmtes Materialverhalten sehr schwierig und dies einer der zentralen Ansatzpunkte für Verbesserungen ist.

7.1.3. Fall eines Würfels unter Gravitation auf einen Keil

Ein weiterer Test der Festkörpereigenschaften des simulierten Materials ist ein Würfel, der aus einer geringen Höhe auf einen Keil fallen gelassen wird, so dass ggf. eine plastische Deformation entsteht. Hierbei ist das Materialverhalten beim Aufprall bzw. danach bis zu einem Ruhezustand interessant. Für die Simulation wird eine MCA_BLOCK-Szene verwendet. Der Würfel und der Keil bestehen aus dem zu untersuchenden Material, die Parameter werden bei den entsprechenden Versuchen genannt und der Boden ist fest. Zusätzlich wurden zwei kleine Würfel neben den Keil in die Szene eingefügt, um weitere Partikelinteraktionen beobachten zu können. Im Folgenden werden die einzelnen Versuche vorgestellt und verglichen.

Feste Parameter sind die Maße der Objekte. Der Würfel besteht aus 1000 Partikeln mit einer Kantenlänge von 1,5 mm und der feste Boden aus 3380 Partikeln mit einer Höhe von 0,5 mm. Grundsätzlich basieren die folgenden Versuche und gewählten Parameter auf den Versuchen in Kapitel 7.1.1 und Kapitel 7.1.2. Die Laufzeit der Simulationen betrug jeweils ca. 10 Stunden.

Versuch 1 In diesem Versuch soll ein Würfel nach dem Aufprall auf einen Keil stabil bleiben, ggf. auf eine Seite des Keils fallen, sich wie ein Würfel aus Stahl verhalten und nicht auseinander brechen. Hierfür wurden die wesentlichen Parameter wie folgt gewählt:

Parameter	Formelzeichen	Wert
shearModulus(Stahl)	G_i	$79,3 \cdot 10^9 \text{ Pa}$
yieldAngle(Stahl/Stahl)	γ_{ijY}	$1,7647 \cdot 10^{-4}$
ljpAlpha(Stahl/Stahl)	α_{ij}	$2 \cdot 10^{-3} \text{ J}$
frictionEta(Stahl/Stahl)	η_{ij}	$1,5 \cdot 10^{-5}$
frictionEta(Stahl/Boden)	η_{ij}	$1,5 \cdot 10^{-6}$
frictionEta(Boden/Stahl)	η_{ij}	$1,5 \cdot 10^{-6}$
frictionMus(Stahl/Stahl)	μ_{ij}	0,6

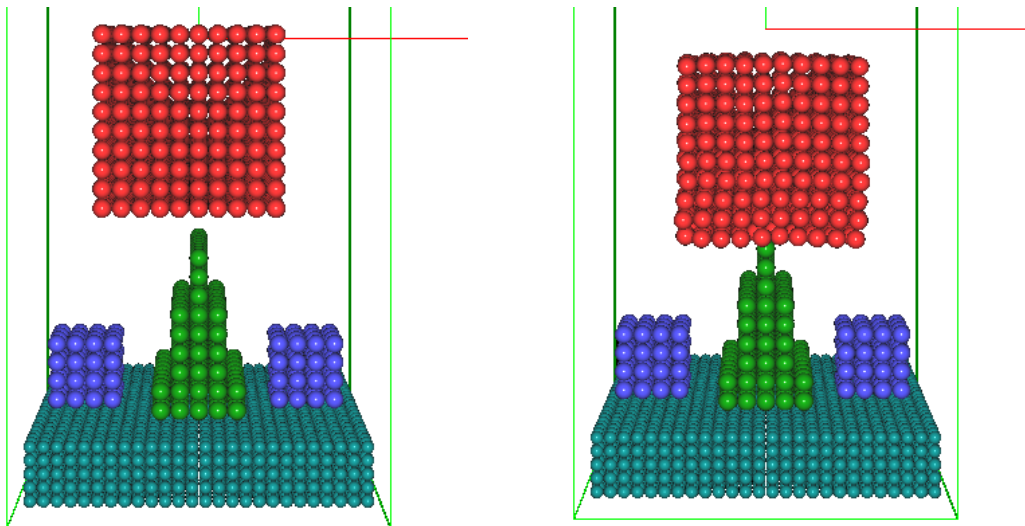


Abbildung 7.5.: Versuch 1 nach einer Simulationszeit von 0,0813 s und 0,1198 s

Bei diesem Versuch (Abb. 7.5) wird ersichtlich, dass sich die Würfelform mit diesem Parametersatz zu Beginn nicht verändert, so wie es bei einem stabilen Festkörper aus Stahl zu erwarten ist. Jedoch wird der Würfel bei einer längeren Simulationszeit instabil, da sich die unterste Schicht leicht hin und her verschiebt, während der Würfel auf dem Keil liegen bleibt und nicht kippt.

Versuch 2 Der folgende Versuch entspricht in fast allen Punkten dem ersten Versuch, jedoch soll auf jeden Fall ein Kippen des Würfels stattfinden. Hierfür wurden die wesentlichen Parameter wie im Versuch zuvor gewählt, wobei der Würfel etwas zur Seite versetzt wurde.

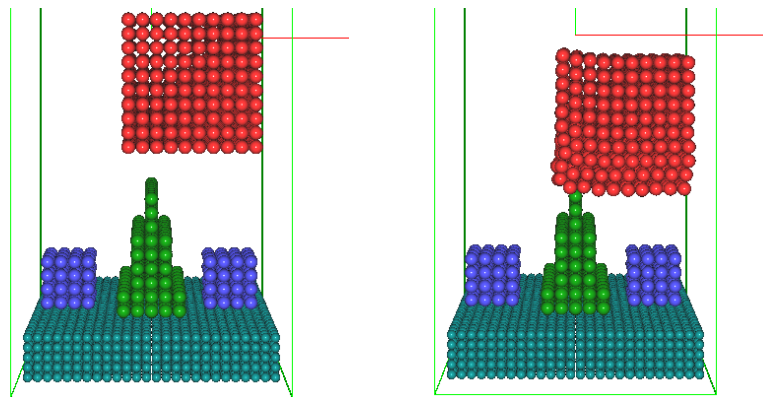


Abbildung 7.6.: Versuch 2 nach einer Simulationszeit von 0,0778 s und 0,1292 s

Der Würfel verhält sich wie im vorherigen Versuch. Er bleibt auf dem Keil liegen und kippt wider Erwarten nicht (Abb. 7.6). Dies könnte sowohl daran liegen, dass die Gravitationskraft nicht stark genug auf den Würfel wirkt, als auch daran, dass die Reibungskräfte zwischen Keil- und Würfelpartikel einen zu großen Einfluss ausüben. Die Betrachtung der Kräfte in der folgenden Abbildung 7.7 bekräftigt diese Vermutungen. Zum Simulationstart sind die Kräfte in der linken Abbildung relativ ausgeglichen, die Farbkodierung zeigt nur minimale Differenzen auf. Zum Zeitpunkt des Kontaktes mit dem Keil, dargestellt in dem rechten Teil der Abbildung, zeigt die Farbkodierung einen stärkeren Unterschied der Kräfte im Kontaktbereich im Vergleich zum Rest der Würfels.

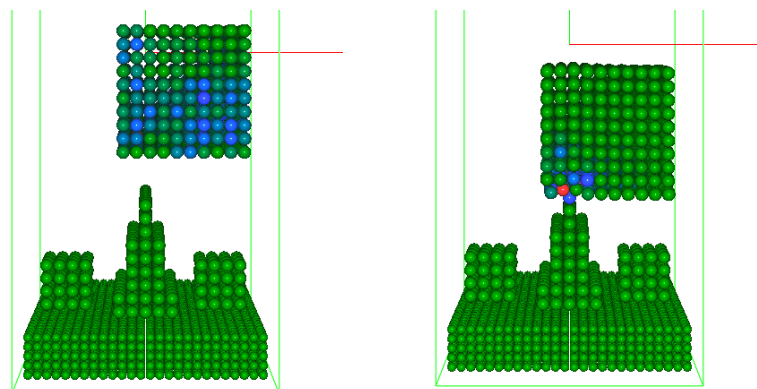


Abbildung 7.7.: Versuch 2 mit Anzeige der internen Kräfte

Versuch 3 Dieser Versuch (Abb. 7.8) ähnelt dem ersten Versuch, jedoch werden durch die gewählten Parameter die angesprochenen Instabilitäten im Laufe der Simulation deutlich. Parameter, die die Reibungskräfte beeinflussen (frictionEta und frictionMus), wurden auf 0 gesetzt:

Parameter	Formelzeichen	Wert
shearModulus(Stahl)	G_i	$79,3 \cdot 10^9$ Pa
yieldAngle(Stahl/Stahl)	γ_{ijY}	$1,7647 \cdot 10^{-4}$
ljpAlpha(Stahl/Stahl)	α_{ij}	$2 \cdot 10^{-3}$ J
frictionEta	η_{ij}	0,0
frictionMus(Stahl/Stahl)	μ_{ij}	0,0

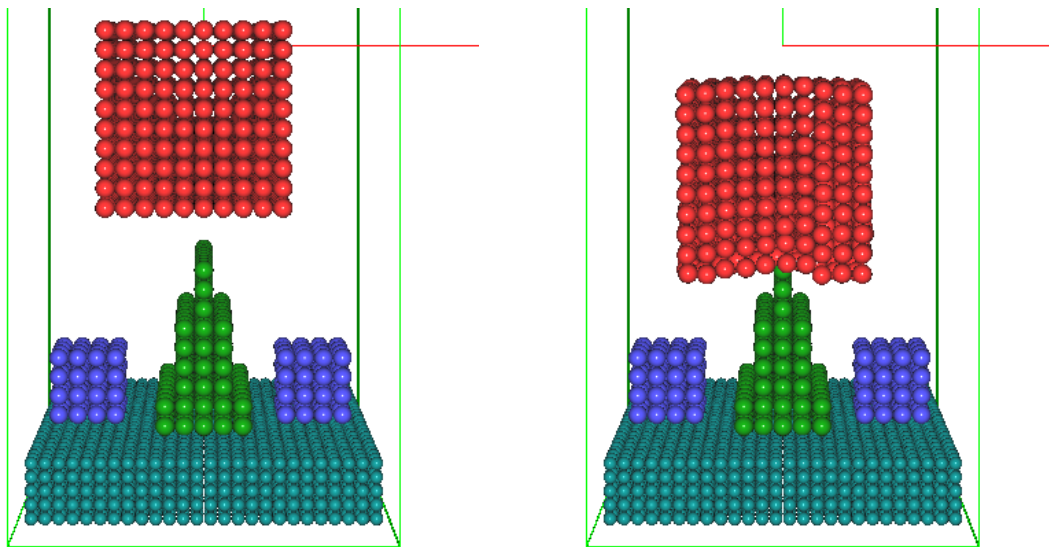


Abbildung 7.8.: Versuch 3 nach einer Simulationszeit von 0,0821 s und von 0,1599 s

Das Verhalten des Würfels ähnelt dem Verhalten in Versuch 1. Es ist diesmal jedoch deutlich zu sehen, dass sich die Partikelebenen sowohl horizontal als auch vertikal gegeneinander verschieben. Die zwischen den Partikeln wirkenden Kräfte scheinen selbst ohne Reibung einen deutlich größeren Einfluss zu haben als die Gravitationskraft.

Versuch 4 Auch dieser Versuch (Abb. 7.9) ähnelt dem ersten, jedoch wird der Würfel aufgrund der anders gewählten Parameter durch den Keil durchstoßen und verhält sich wie ein weicher Körper:

Parameter	Formelzeichen	Wert
shearModulus(Stahl)	G_i	$79,3 \cdot 10^9 \text{ Pa}$
yieldAngle(Stahl/Stahl)	γ_{ijY}	$6,5 \cdot 10^{-10}$
yieldAngle(Boden/Boden)	γ_{ijY}	$6,5 \cdot 10^{-10}$
ljpAlpha(Stahl/Stahl)	α_{ij}	$9,5 \cdot 10^{-200} \text{ J}$
ljpAlpha(Stahl/Boden)	α_{ij}	$9,5 \cdot 10^{-150} \text{ J}$
ljpAlpha(Boden/Stahl)	α_{ij}	$-9,5 \cdot 10^{-15} \text{ J}$
ljpAlpha(Boden/Boden)	α_{ij}	$-9,5 \cdot 10^{-20} \text{ J}$
frictionEta(Stahl/Stahl)	η_{ij}	$5,75 \cdot 10^{-8}$
frictionEta(Boden/Stahl)	η_{ij}	$-2,75 \cdot 10^{-15}$
frictionEta(Stahl/Boden)	η_{ij}	$-2,75 \cdot 10^{-15}$
frictionEta(Boden/Boden)	η_{ij}	$5,78 \cdot 10^{-8}$
frictionMus(Stahl/Stahl)	μ_{ij}	0,8

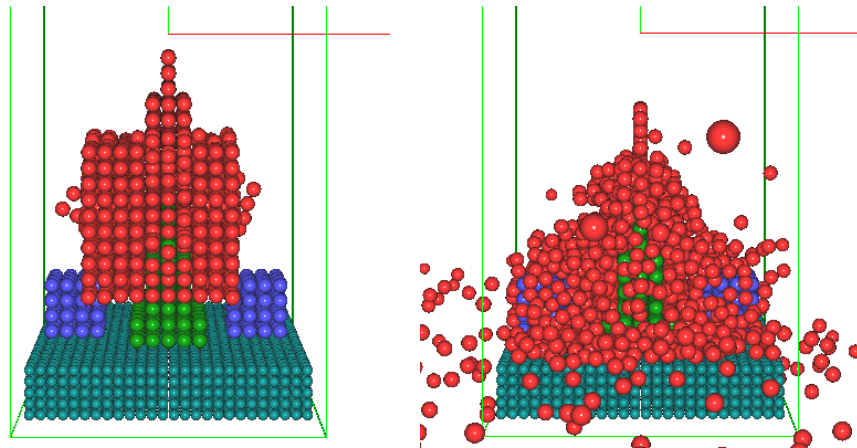


Abbildung 7.9.: Versuch 4 nach einer Simulationszeit von 0,0962 s und von 0,1396 s

Die unmittelbar über dem Keil liegenden Würfelpartikelebenen bleiben auf dem Keil liegen. Sämtliche andere Partikel werden in alle Richtungen zerstreut. Die Parameter für das Lennard-Jones-Potential scheinen den größten Einfluss auf den Zusammenhalt der Würfelpartikel zu haben.

Zusammenfassend lässt sich sagen, dass die Ergebnisse mit denen in Kapitel 7.1.1 und Kapitel 7.1.2 vergleichbar sind. Der simulierte Keil hat die gleiche Funktion wie ein kompletter fester Boden, obwohl die Kontaktfläche zum Würfel deutlich geringer ist. Ein Kippen eines stabilen Würfels konnte nicht simuliert werden.

7.1.4. Ruhelage eines Festkörperstabes

Ein wichtiger Bestandteil zur Bestimmung geeigneter Parameter ist die Festigkeit des Materials unter Scherung und Biegung. Ein geeigneter Testfall zur Überprüfung gewählter Parameter ist eine Simulation der Ruhelage eines Festkörperstabes. Bei diesem Versuch wird eine MCA_BLOCK-Szene verwendet und der Stab als Block modelliert. Die Auflageflächen des Stabes, die sich jeweils an seinen Enden befinden, werden durch die Verwendung von `obstacleBlocks` realisiert. Anschließend wird der Stab aus geringer Höhe auf die Blöcke unter Gravitationseinfluss fallen gelassen. Zunächst wird in diesem Unterabschnitt ein Standardfall definiert. Anschließend wird auf die verwendeten Parameter der einzelnen Versuche eingegangen und die erzielten Simulationsergebnisse werden analysiert.

Standardversuch Die Auswertung des Stabversuches erfolgt ausgehend von einem Standardfall. Dabei wird ein Partikelstab aus geringer Höhe auf zwei Hindernisse fallen gelassen und durch diese abgebremst. Hierbei lassen sich verschiedene Festkörpereigenschaften erkennen und der Effekt der variierten Parameter wird sichtbar. Der Standardfall enthält einen Satz von Parametern, die in anschließenden Versuchen separat voneinander variiert werden. Die wichtigsten verwendeten Parameter (Bedeutung nachzulesen in Anhang C) für den Standardfall sind:

Parameter	Formelzeichen	Wert
stepTime		$1 \cdot 10^{-7}$ s
shearModulus(Stahl)	G_i	$79,3 \cdot 10^9$ Pa
yieldAngle(Stahl/Stahl)	γ_{ijY}	$1 \cdot 10^{-4}$
ljpAlpha(Stahl/Stahl)	α_{ij}	$1 \cdot 10^{-3}$ J
frictionEta(Stahl/Stahl)	η_{ij}	$1,5 \cdot 10^{-5}$
frictionEta(Stahl/Boden)	η_{ij}	$1,5 \cdot 10^{-6}$
frictionMus(Stahl/Stahl)	μ_{ij}	0,8

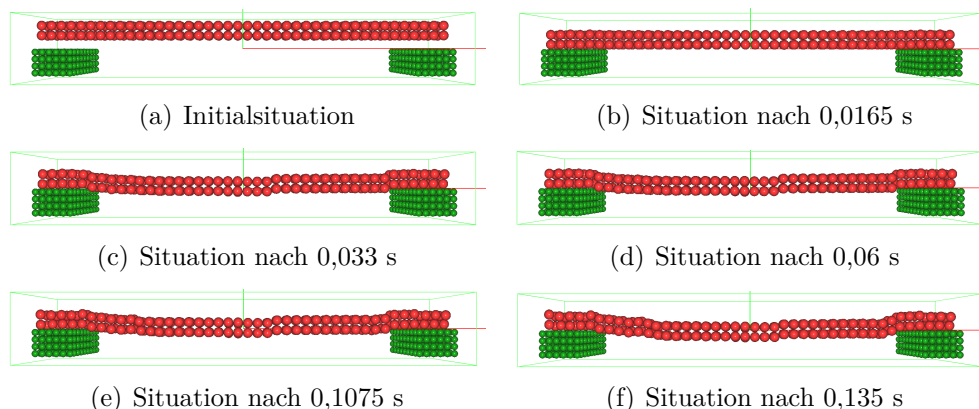


Abbildung 7.10.: Simulationsverlauf des Standardversuchs

Abbildung 7.10 zeigt die Simulation des Standardfalls im Anfangszustand (Abbildung 7.10(a)) und zum Zeitpunkt des Aufpralls (Abbildung 7.10(b)). Nur unter Einfluss der intern wirkenden MCA-Kräfte und der externen Gravitation bleibt der Stab zunächst stabil. Im weiteren Verlauf der Simulation (Abbildung 7.10(c)) beginnt sich der Stab nach der Kollision mit den festen Hindernissen zu verformen. Die MCA-Kräfte verhindern, dass sich die Partikel, die nicht auf den Hindernissen aufliegen, frei nach unten bewegen können. Abbildung 7.10(e) zeigt den weiteren Verlauf der Simulation auf: Teilweise verkanten die Partikel und der Stab biegt sich weiter durch, bis er eine stabile Konfiguration, wie in Abbildung 7.10(f) zu sehen, erreicht.

Im folgenden Abschnitt werden nun zwei ausgewählte Parameter des Standardfalls variiert. Dabei wird aufgezeigt, in welcher Weise die Änderung von α_{ij} und γ_{ijY} das Simulationsverhalten beeinflusst. Alle anderen Parameter werden dabei nicht geändert, die Ausgangssituation bleibt in jedem Fall identisch zum Standardversuch.

Variation von α_{ij} Zunächst werden drei Versuche mit verschiedenen α_{ij} durchgeführt, die sich von denen des Standardversuchs unterscheiden. Das Lennard-Jones-Potential approximiert intermolekulare Wechselwirkungen. Es wird untersucht, inwiefern eine Änderung der Parameter des Lennard-Jones-Potentials eine Änderung der Steifigkeit des Materials bewirkt.

Versuch 1

Parameter	Formelzeichen	Wert
ljpAlpha(Stahl/Stahl)	α_{ij}	$1 \cdot 10^{-4} \text{ J}$

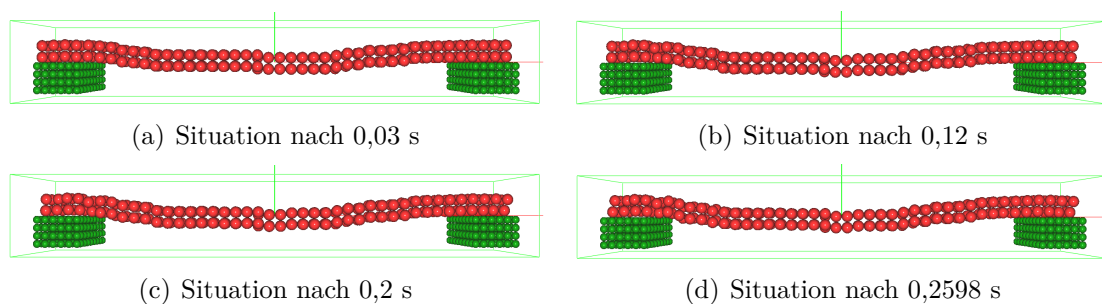


Abbildung 7.11.: Simulationsverlauf von Versuch 1

Abbildung 7.11 zeigt das Simulationsverhalten für $\alpha_{ij} = 1 \cdot 10^{-4} \text{ J}$. Durch den kleineren Wert für α_{ij} ist eine schwächere Bindung zwischen den Partikeln zu erwarten. Im Laufe der Simulation (Abbildung 7.11(c) und 7.11(d)) biegt sich der Stab analog zum Standardversuch durch.

Versuch 2

Parameter	Formelzeichen	Wert
ljpAlpha(Stahl/Stahl)	α_{ij}	$5 \cdot 10^{-2} \text{ J}$

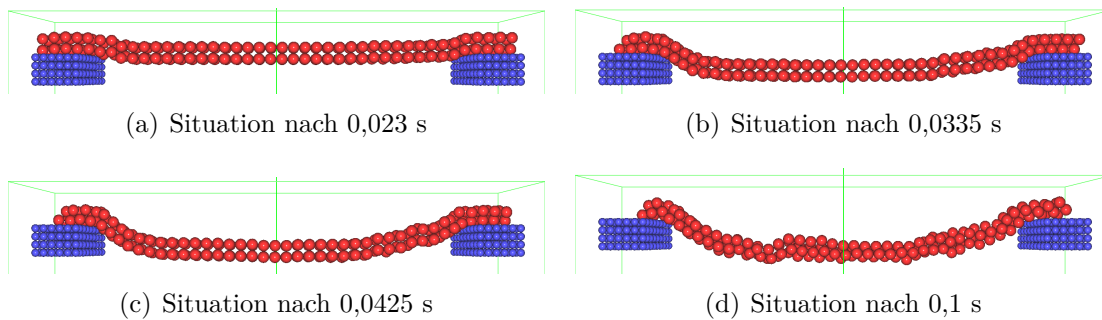


Abbildung 7.12.: Simulationsverlauf von Versuch 2

Abbildung 7.12 zeigt das Simulationsverhalten für $\alpha_{ij} = 5 \cdot 10^{-2} \text{ J}$. Zu Beginn der Simulation verhält sich der Stab ähnlich wie im Standardversuch, jedoch ist die Durchbiegung deutlich größer. Dieses Verhalten ist insofern unerwartet, da nun durch das stärkere Lennard-Jones-Potential eine größere Kraft zwischen den Partikeln wirkt. Im weiteren Verlauf der Simulation (Abbildung 7.12(c) und 7.12(d)) biegt sich der Stab zunächst weiter durch, wird aber zum Ende hin instabil.

Versuch 3

Parameter	Formelzeichen	Wert
ljpAlpha(Stahl/Stahl)	α_{ij}	$1 \cdot 10^{-2} \text{ J}$

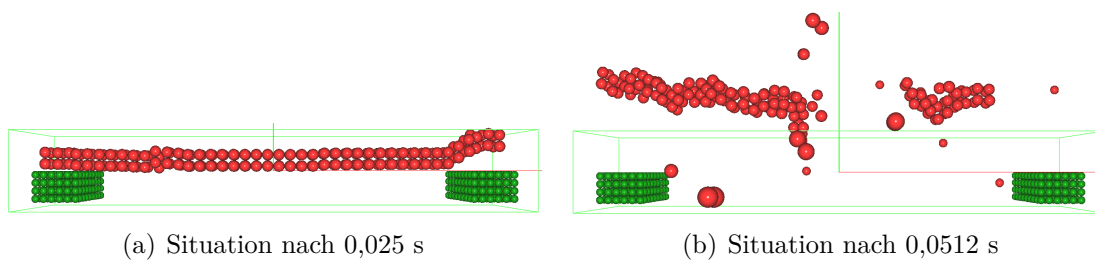


Abbildung 7.13.: Simulationsverlauf von Versuch 3

Abbildung 7.13 zeigt das Simulationsverhalten für $\alpha_{ij} = 1 \cdot 10^{-2} \text{ J}$. Die Simulation mit diesem Parametersatz ist nicht stabil, der Stab explodiert.

Die Simulationsergebnisse für die getesteten Parametersätze mit unterschiedlichen α_{ij} entsprechen nicht den Erwartungen. Es war zwar zu erwarten, dass die Variation des Lennard-Jones-Potentials Auswirkungen auf die Stabilität eines Festkörperstabes unter Scherbelastungen haben würde. Insbesondere die stärkere Durchbiegung im zweiten Versuch ist unerwartet, weil ein größerer Wert für α_{ij} auch einen stärkeren Partikelzusammenhalt erwarten ließe. Da Scherungen für kleine Auslenkungen orthogonal zur Krafrichtung des Lennard-Jones-Potentials sind, dominiert in der Kraft gegen scherende Verformung jedoch der Schubmodul, welches durch Variation von γ_{ijY} im Folgenden untersucht werden soll. Eine Möglichkeit für die stark unerwarteten Ergebnisse ist numerischer Natur, da die auftretenden Kräfte gegen den gewählten Zeitschritt groß werden können.

Variation von γ_{ijY} Im folgenden Abschnitt werden drei Versuche mit verschiedenen γ_{ijY} durchgeführt, die sich von denen des Standardversuchs unterscheiden. Dieser Parameter hat Einfluss auf die Wirksamkeit des Schubmoduls und damit auf den Scherwiderstand des Materials.

Versuch 4

Parameter	Formelzeichen	Wert
yieldAngle(Stahl/Stahl)	γ_{ijY}	$1 \cdot 10^{-5}$

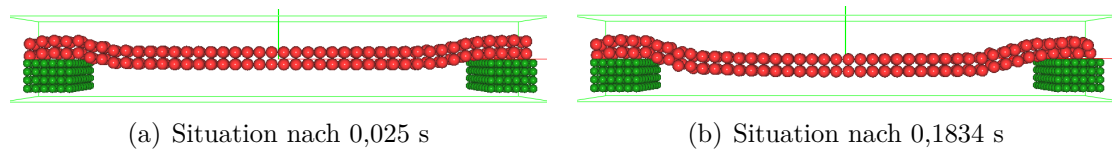


Abbildung 7.14.: Simulationsverlauf von Versuch 4

Abbildung 7.14 zeigt das Simulationsverhalten für $\gamma_{ijY} = 1 \cdot 10^{-5}$. Der Stab verformt sich analog zum Standardfall und bleibt stabil durchgebogen.

Versuch 5 und 6

Parameter	Formelzeichen	Wert
Versuch5: yieldAngle(Stahl/Stahl)	γ_{ijY}	$5 \cdot 10^{-2}$
Versuch6: yieldAngle(Stahl/Stahl)	γ_{ijY}	$1 \cdot 10^{-2}$

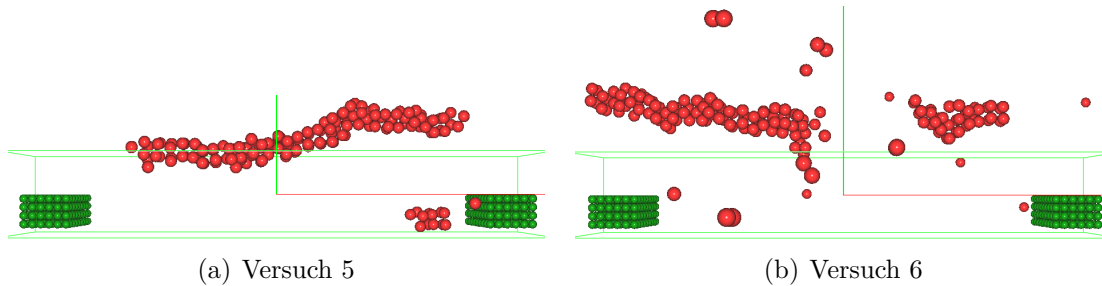


Abbildung 7.15.: Situationen für Versuche 5 und 6

Abbildung 7.15 zeigt das Simulationsverhalten für $\gamma_{ijY} = 5 \cdot 10^{-2}$ (Versuch 5) bzw. $\gamma_{ijY} = 1 \cdot 10^{-2}$ (Versuch 6). Beide Simulationen sind instabil und explodieren.

Der Schubmodul G_i ist durch seinen hohen Wert für Stahl ($G_i = 79,3 \cdot 10^9$ Pa) eine kritische Komponente bei diskreten Simulationen. γ_{ijY} stellt dabei eine Art *cut-off* (vgl. Formel 3.63) dar, ab dem die Scherkraft einen konstanten Wert annimmt. Durch Vergrößerung von γ_{ijY} resultiert demnach eine größere Scherkraft. Auch hier können diese Kräfte groß gegen den gewählten Zeitschritt werden und zu Instabilitäten in den durchgeführten Simulationen führen.

7.2. SPH-Testfälle

In diesem Abschnitt wird das SPH-Verfahren mit Hilfe einiger Testfälle genauer untersucht. Dabei wird zunächst ein grundlegender Versuch zum Verhalten von SPH-Partikeln als Flüssigkeitssimulation durchgeführt. Anschließend folgen einige Festkörperversuche, deren Ergebnisse näher diskutiert werden.

7.2.1. Auffangen einer Flüssigkeit

Um die Funktionalität der in Abschnitt 4.2 beschriebenen Boundary-Partikel zu testen und zu demonstrieren, wurde ein Testszenario erstellt, welches einen Partikelblock simuliert, der aus einer Flüssigkeit besteht und in einen auffangenden Behälter fällt (siehe Abbildung 7.16). Dieser Behälter wurde in der Simulation mit entsprechenden Boundary-Partikeln modelliert und zeigte das erwartete Verhalten. Der Block zerfloss durch den Aufprall innerhalb des Behälters wie Wasser in alle Richtungen und wurde durch die Boundary-Partikel und die auftretenden Kräfte an den Rändern wieder nach oben aus dem Behälter hinaus gedrängt.

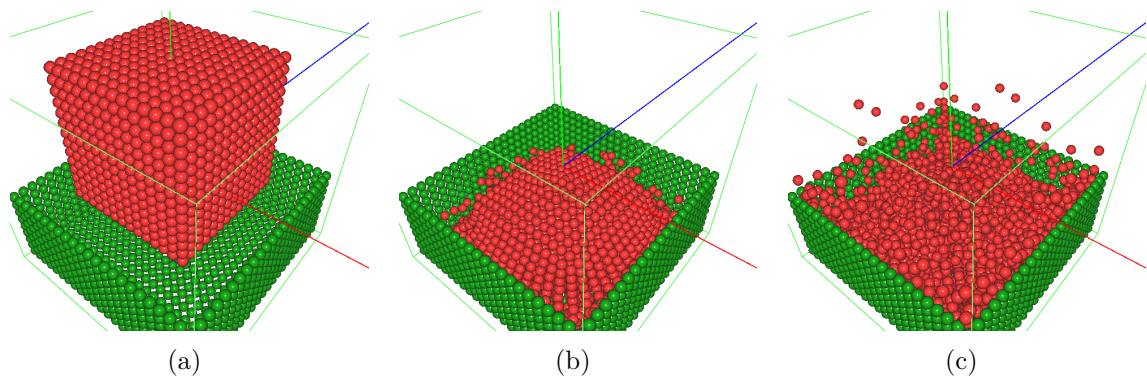


Abbildung 7.16.: Auftreten eines flüssigen Partikelblocks (rot) in ein Gefäß (grün)

Für die Simulation wurden die Kräfte `youngsModulus` und `poissonRatio` deaktiviert, da der Block nur aus Wasserpunkten bestehen soll, die sich in keiner festen Konfiguration zueinander befinden sollen. Dieser Versuch wurde als gelungen bewertet, da das simulierte Verhalten dem Verhalten einer Flüssigkeit wie Wasser ähnelt. Die Implementierung der Boundary-Partikel erscheint ebenfalls ein geeignetes Mittel zu sein, um mit SPH simulierte Flüssigkeiten festzuhalten.

7.2.2. Ruhelage eines Würfels unter Gravitation auf festem Boden

Analog zu den Tests für MCA wurde auch für SPH ein Würfel unter Gravitation auf festem Boden simuliert. Versuchsziel ist ebenfalls die Simulation eines festen Blocks, der über einen Zeitraum keine Verformung aufweist. Zur Simulation wurde eine `SPH_BLOCK`-Szene benutzt. Die geometrischen Größen entsprechen den Angaben der MCA-Block-Versuche.

Es konnte jedoch keine Parameterkombination gefunden werden, mit der ein Block über einen Zeitraum länger als 0,14 s keine Verformung aufweist (Abb. 7.17). Ausschlaggebende Parameter für die Simulation sind:

Parameter	Formelzeichen	Wert
<code>yieldGammaTable(Stahl)</code>	γ_y	12,01
<code>creepGamma(Stahl)</code>	γ_c	0,2
<code>youngsModulusTable(Stahl)</code>	E	$9 \cdot 10^{12}$ Pa

Die genannten Parameter führen zu einem stabilen Block für einen Zeitraum bis 0,14 s. Nach dieser Zeitspanne beginnt der Block sich an den Ecken aufzulösen. Grundlegendes Problem hierbei stellt vermutlich die Benutzung von `creepGamma` dar, da dieser Parameter die Dehnungsfähigkeit der Partikelverbindungen definiert. Wird `creepGamma` (wie in diesem Versuch) auf einen sehr kleinen Wert gesetzt, verhindert dies in Verbindung mit dem `youngsModulus` eine starke Verformung des Materials. Der hohe `youngsModulus` soll

an dieser Stelle für eine Festigkeit in der Größenordnung von Stahl sorgen. Da jedoch bei sehr kleinen creepGamma-Werten das Material anfängt zu schwingen, ergibt sich nach einiger Zeit (in diesem Fall nach 1,4 s) ein Ablösen der Eckpartikel. Auf eine analoge Problematik wird zusätzlich in Kapitel 7.4.2 eingegangen. Trotz geringfügiger Variationen dieser Parameter bei Beibehaltung des Verhaltens von Stahl (o.ä.) konnte kein besseres Verhalten des Blocks hervorgerufen werden.

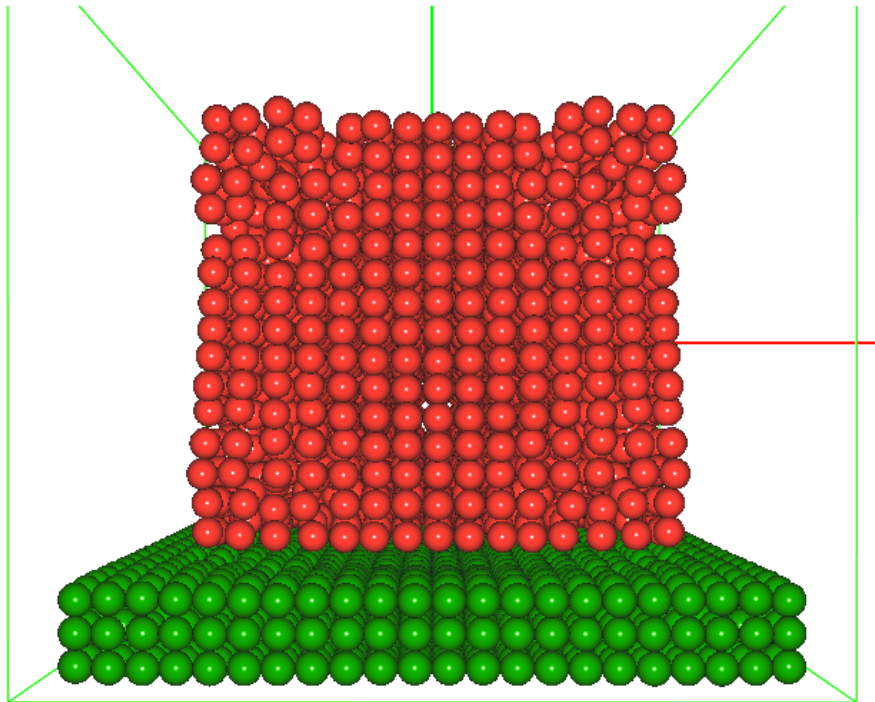


Abbildung 7.17.: SPH-Block in Ruhelage nach 1,4 s.

Da es nicht gelang einen Block zu simulieren, der sich in einer Ruhelage stabil verhält, konnte dementsprechend auch nur ein fallender, instabiler Block simuliert werden, der das selbe Verhalten nach (in diesem Fall) kürzerer Zeit zeigt. Interessant wäre deshalb eher das Auftreffen eines Blocks auf nicht-flache Objekte, um interne strukturelle Schwächen des gewählten Ansatzes zu untersuchen.

7.2.3. Fall eines Würfels unter Gravitation auf einen Keil

Analog zu der MCA-Variante des Keilversuchs ist das Verhalten eines Würfels interessant, der in einer SPH_BLOCK-Szene auf einen Keil fallen gelassen wird. Trotz der Ergebnisse der Versuche in Kapitel 7.2.2 wurde der Entschluss gefasst, einige Keilversuche mit SPH vorzunehmen, um ggf. Materialverhalten beobachten zu können, welches bei der Parametersuche für Spanbildungstests hilfreich sein könnte. Der Würfel besteht aus 1331 Partikeln mit einer Kantenlänge von 330 mm und der feste Boden aus 3125 Partikeln mit einer Höhe von 133 mm. Grundsätzlich basieren die folgenden Versuche und gewählten Parameter auf den Versuchen in Kapitel 7.2.2, wurden jedoch während einiger Tests systematisch verändert, um eventuelle Grenzwerte der Parameter festzustellen. Bei der folgenden Auswahl von Versuchen ist der Würfel nicht von vornherein komplett zusammengefallen und es konnte jeweils ein anderes Verhalten beobachtet werden.

Versuch 1 Wie in der MCA-Variante des Versuchs soll ein Würfel nach dem Aufprall auf einen Keil stabil bleiben, ggf. auf eine Seite des Keils fallen und sich wie ein Würfel aus Stahl verhalten, der nicht auseinanderbricht. Hierfür wurden die wesentlichen Parameter wie folgt gewählt:

Parameter	Formelzeichen	Wert
viscosity	μ_i	$1,1 \cdot 10^4 \text{ Pa} \cdot \text{s}$
viscosityStiffness	k	$9,5 \cdot 10^4$
creepGamma	γ_c	0,1
yieldGamma	γ_y	0,01
density	ρ_i	$7,850 \cdot 10^3 \frac{\text{kg}}{\text{m}^3}$
poissonRatio	ν	0,28
youngsModulus	E	$2,0 \cdot 10^{11} \text{ Pa}$

Bei diesem Versuch wird ersichtlich, dass der Würfel anfangs leicht schwingt, was ggf. auf die Wahl des Parameters creepGamma zurückzuführen ist (vgl. Anmerkungen in Kapitel 7.4.2). Bereits im Fall verändert der Würfel seine Form, so dass die Partikel nicht mehr auf parallel verlaufenden Ebenen liegen (Abb. 7.18). Diese Form behält er jedoch bis zum Zusammenstoß mit dem Keil bei.

Der Keil dringt schließlich in den Würfel ein, trennt ihn jedoch nicht gänzlich (Abb. 7.19), was sich mit der Wahl des Parameters yieldgamma erklären lassen könnte (siehe auch hierzu Kapitel 7.4.2). Es ist außerdem zu beobachten, dass vom Boden und Keil auf Teile des Würfels eine leichte Anziehungskraft zu wirken scheint. Über diese kann ohne weitere zeitaufwendige Parametertests nur spekuliert werden, warum sie auftritt. Dieses Verhalten tritt in gleicher Weise bei einem Keil aus Boundary-Partikeln auf und auch selbst dann, wenn die oben gelisteten Parameterwerte für den Keil und den Boden doppelt so groß gewählt werden.

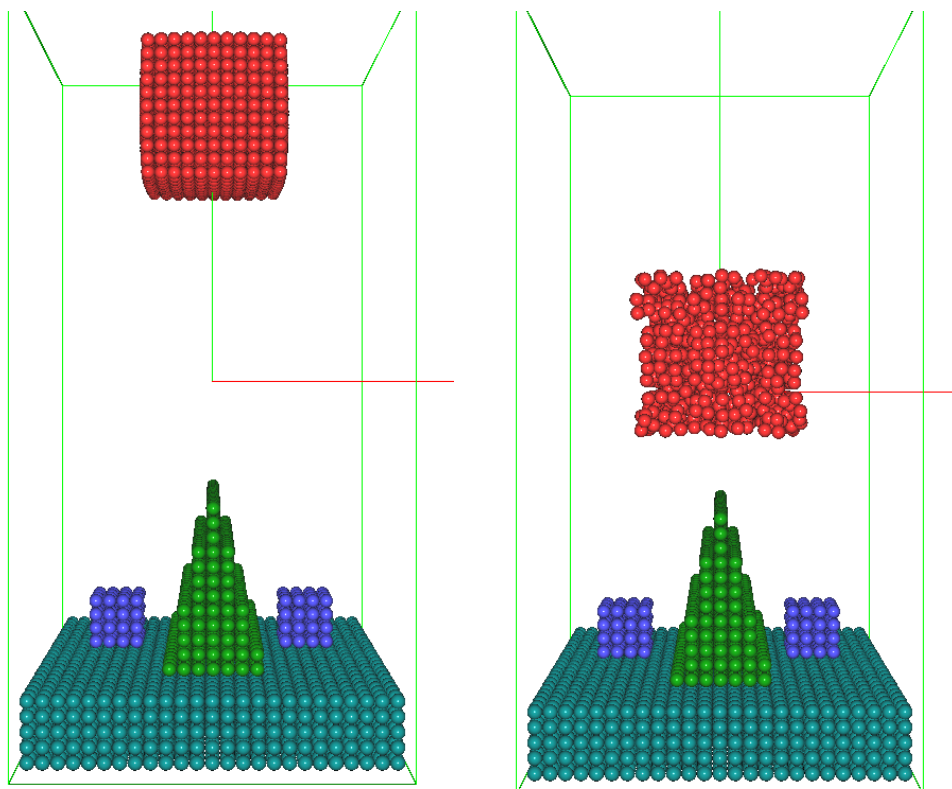


Abbildung 7.18.: Versuch 1 nach einer Simulationszeit von 0,01499 s und von 0,33859 s

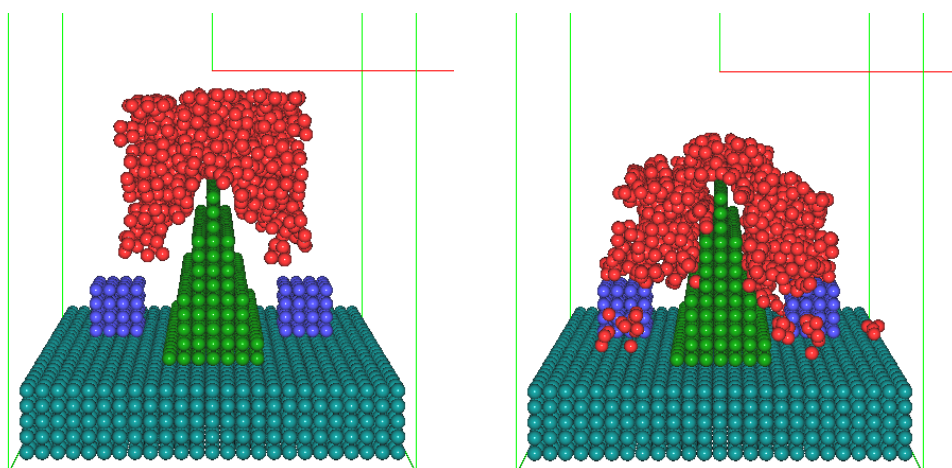


Abbildung 7.19.: Situation in Versuch 1 nach einer Simulationszeit von 0,47179 s und 1,73260 s

Versuch 2 Dieser Versuch entspricht dem ersten Versuch, jedoch wurden einige Parameter für den Würfel sowie die Dichte des Bodens um ein Vielfaches größer gewählt:

Parameter	Formelzeichen	Wert
viscosity(Stahl)	μ_i	$1,1 \cdot 10^5 \text{ Pa} \cdot \text{s}$
viscosityStiffness(Stahl)	k	$9,5 \cdot 10^5$
density(Boden)	ρ_i	$2,0 \cdot 10^4 \frac{\text{kg}}{\text{m}^3}$
poissonRatio(Stahl)	ν	1,28
youngsModulus(Stahl)	E	$2,0 \cdot 10^{12} \text{ Pa}$

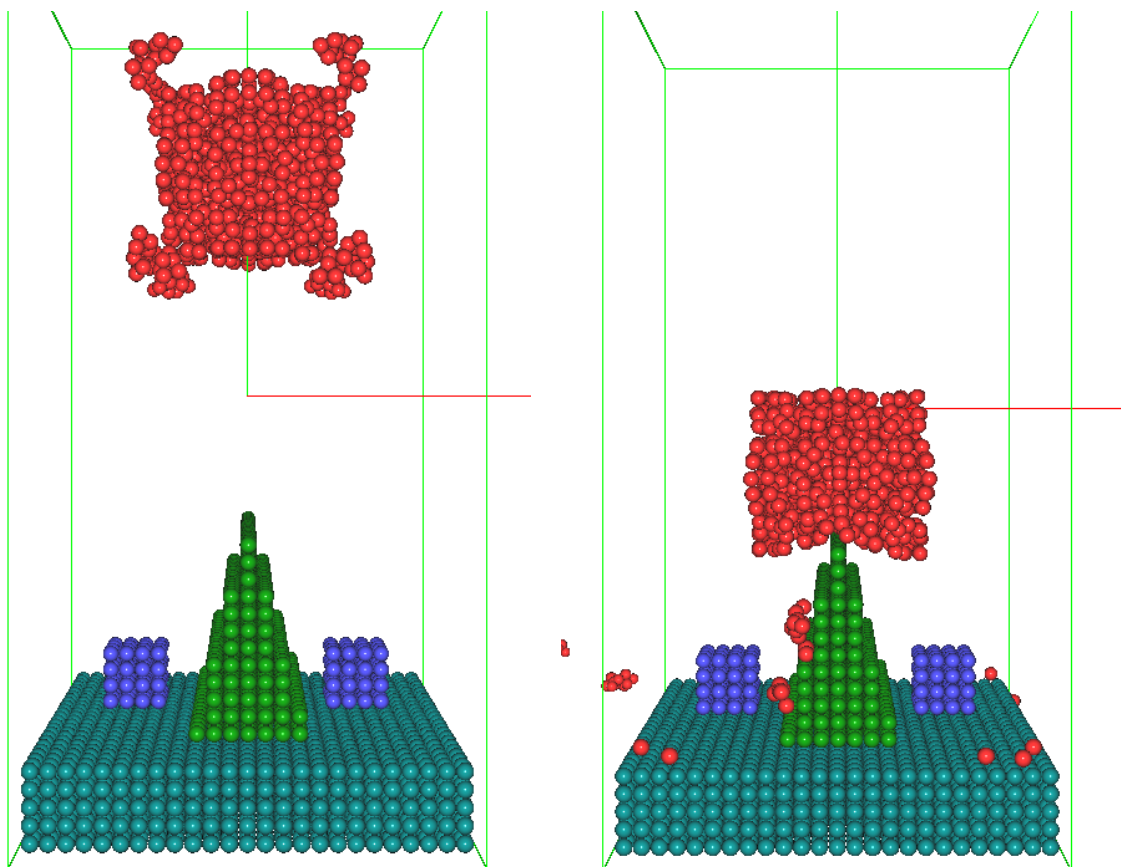


Abbildung 7.20.: Versuch 2 nach einer Simulationszeit von 0,194 s und von 1,4783 s

Bei diesem Versuch (Abb. 7.20) verhält sich der Würfel ebenso instabil. Im Fall verliert er an den Ecken einige Partikel, jedoch bleiben die restlichen Partikel zusammen. Selbst beim Aufprall auf den Keil wird der Würfel nur leicht durchdrungen und bleibt stecken. Insbesondere die Viskositäts-Parameter scheinen den Zusammenhalt der Würfelpartikel stark positiv zu beeinflussen.

Versuch 3 Versuch 3 entspricht in fast allen Punkten dem zweiten Versuch, jedoch soll auf jeden Fall ein Kippen des Würfels stattfinden. Hierfür wurden die wesentlichen Parameter wie im Versuch zuvor gewählt, wobei der Würfel etwas zur Seite versetzt wurde. Der Würfel verhält sich wie im vorherigen Versuch, jedoch scheint er sich nach dem Aufkommen auf den Keil auf die erwartete Seite zu neigen (Abb. 7.21).

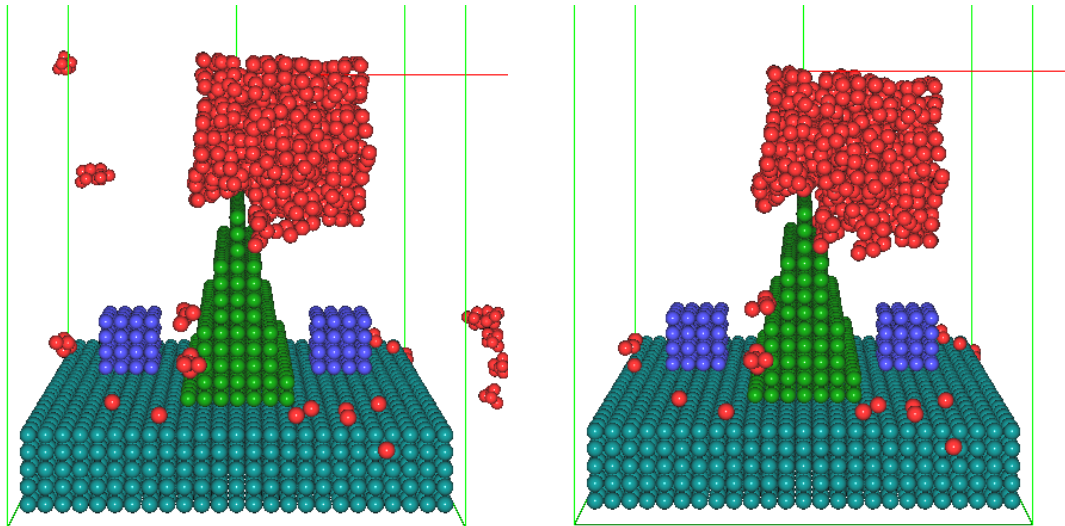


Abbildung 7.21.: Versuch 3 nach einer Simulationszeit von 0,6644 s und von 1,6375 s

Zusammenfassend lässt sich sagen, dass die Ergebnisse mit denen in Kapitel 7.2.2 vergleichbar sind. Es war nicht möglich einen stabilen Würfel zu simulieren. Bei einem instabilen Würfel ließ sich jedoch ein Kippverhalten beobachten, was weitere Parametertests sinnvoll erscheinen lässt.

7.3. Vergleich mit der Mechanik

Bei einer vergleichenden Betrachtung eines Stabes, der an beiden Enden auf einem festen Boden aufliegt, ist ein Vergleichswert der maximalen Durchbiegung aus der Mechanik sinnvoll für die Bewertung der MCA-Ergebnisse (Abschnitt 7.1.4). Es wirkt nur die Gravitationskraft $F_{Gravitation}$ auf den Stab.

Die hierbei maximale Durchbiegung w_{max} berechnet sich mit:

$$w_{max} = \frac{5 \cdot F_{Gravitation} \cdot L^3}{384 \cdot E \cdot I_y}. \quad (7.1)$$

L ist die Länge des Stabes, E der Elastizitätsmodul und I_y ist das Flächenträgheitsmoment. Dieses lässt sich mit:

$$I_y = \frac{B \cdot H^3}{12} \quad (7.2)$$

berechnen, hierbei ist B die Breite und H die Höhe des Stabes.

Die Maße des Stabes sind: $L = 0,062$ m, $B = 2,2 \cdot 10^{-3}$ m, $H = 2,2 \cdot 10^{-3}$ m.

Ist der Stab aus Stahl mit einer Dichte von $7840 \frac{\text{kg}}{\text{m}^3}$ und einem Elastizitätsmodul von $210 \cdot 10^9 \frac{\text{kg}}{\text{m} \cdot \text{s}^2}$, wirkt auf ihn eine Gravitationskraft von $F_{Gravitation} = 2,30714 \cdot 10^{-2}$ N und die Durchbiegung beträgt $w_{max} = 1,74646 \cdot 10^{-7}$ m.

Ist der Stab aus Gummi mit einer Dichte von $3265 \frac{\text{kg}}{\text{m}^3}$ und einem Elastizitätsmodul von $3,05 \cdot 10^6 \frac{\text{kg}}{\text{m} \cdot \text{s}^2}$, wirkt auf ihn eine Gravitationskraft von $F_{Gravitation} = 9,60818 \cdot 10^{-3}$ N und die Durchbiegung beträgt $w_{max} = 5,00778 \cdot 10^{-3}$ m.

Bei Betrachtung der Ergebnisse der MCA-Simulation dieses Testfalles (Abschnitt 7.1.4) ist das simulierte Material mit keinem der theoretisch errechneten Materialien vergleichbar. Dies ist ein Anzeichen dafür, dass die verwendeten Simulationsparameter nicht optimal gewählt sind, um die theoretisch errechneten Materialien im Simulator realistisch abzubilden.

7.4. Tests zur Spanbildung

Ein Ziel der Projektgruppe war die Simulation von Spanbildung. Bedingt durch die oben beschriebenen Probleme bei der Parameterwahl war eine Spanbildung nur ansatzweise beobachtbar. Im Folgenden wird, unterteilt nach der Simulationsmethode, auf die dabei beobachteten Phänomene eingegangen.

7.4.1. Spanbildungsverhalten beim MCA-Verfahren

Es wurde ein Fräsprozess simuliert, bei dem ein Schneidkeil mit einer konstanten Geschwindigkeit von $1 \frac{\text{m}}{\text{s}}$ durch einen eingespannten Block schnitt. Die Maße des Blocks sind $0,1 \times 0,025 \times 0,02 \text{ m}^3$. Es wurden Spanwinkel von -5° , 0° , 5° und 30° simuliert, worauf in diesem Abschnitt eingegangen wird. Die Eingrifftiefe des Werkzeugs beträgt $0,0075 \text{ m}$, was bei der gewählten `sampleDistance` von $0,0015 \text{ m}$ fünf Schichten des Werkstücks entspricht. Als Zeitschrittlänge wurden, falls nicht anders vermerkt, $5 \cdot 10^{-8} \text{ s}$ verwendet. Bei Zeitschritten von $5 \cdot 10^{-7} \text{ s}$ oder mehr ist bei den gewählten Parametern explosionsartiges Verhalten beobachtbar, sobald das Werkzeug das Werkstück berührt. Stabile Simulationen mit größeren Zeitschritten sind beispielsweise möglich, wenn die über den `shearModulus` gesteuerten Scheerkräfte unrealistisch klein gewählt werden. Auch die Wahl einer geringen `sampleDistance` führt zur Notwendigkeit kleinerer Zeitschritte, wobei hier auch der zur Verfügung stehende Arbeitsspeicher zu Einschränkungen führte. Die durchgeführten Simulationen mit noch kleineren Zeitschritten wiesen bei den gewählten Materialparametern kein grundlegend anderes Verhalten auf.

Inkompressibilität Ein Problem, das bei den vorangegangenen Parametertests ohne Werkzeug nicht in dieser Weise aufgefallen war, ist die Inkompressibilität der simulierten Festkörper (Metalle). Das Werkstück wird durch das Werkzeug komprimiert (Abb. 7.22), statt nur auszuweichen. Eine Ursache dafür ist die gewählte kubische Gitterstruktur bei der Diskretisierung des Werkstücks. Bei dieser Verteilung besitzt jedes innere Partikel 6 nächste Nachbarn, statt der maximal möglichen 12 (siehe Abbildung 7.23.: optimale Kugelpackung). Bei Keilwinkeln von bis zu 30° ist deutlich erkennbar, dass die Partikel des Werkstücks zunächst in eine andere, der kubischen ähnelnden, Gitterstruktur gestaucht werden.

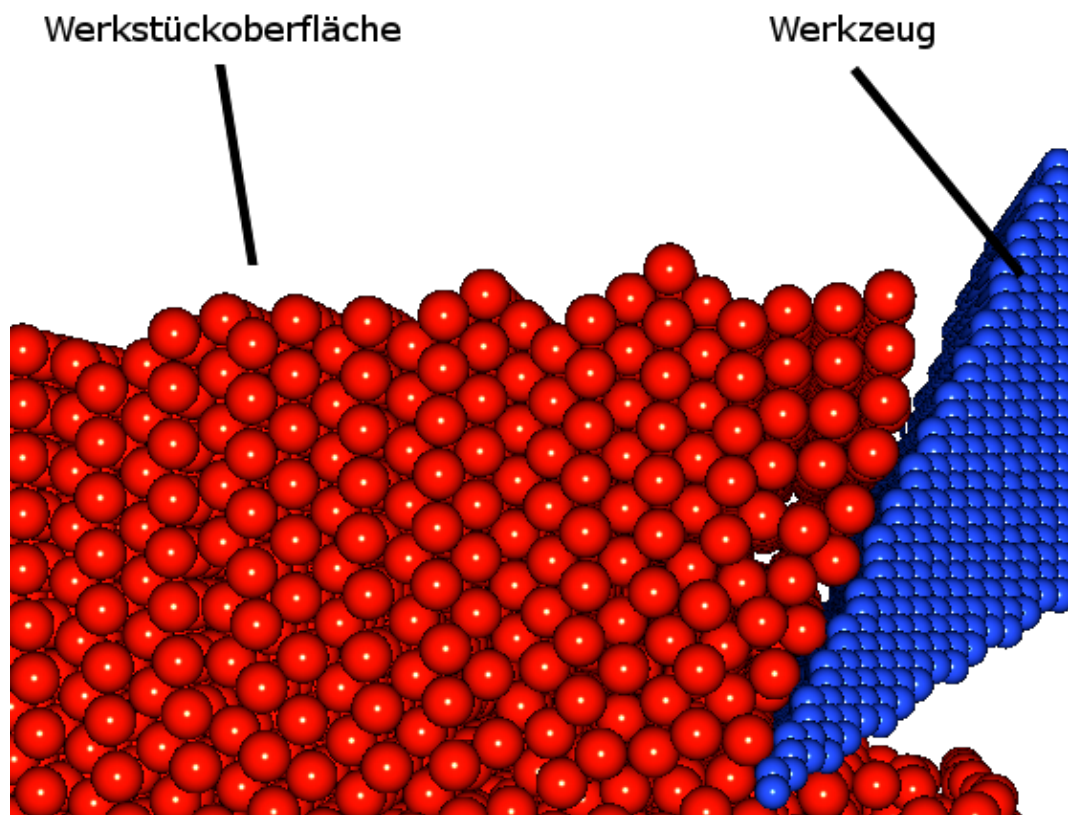


Abbildung 7.22.: Stauchung des Materials in eine andere regelmäßige Struktur.

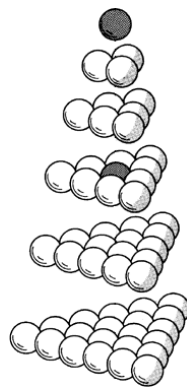


Abbildung 7.23.: Optimale Kugelpackung.

Quelle: Abb. 419.30 Synergetics [Ful79]

Nähere Informationen: <http://www.geodsz.com/deu/d/Kugelpackung>

Werkzeugausprägungen

Um ein Hochfließen eines Spans an der Schneidkante zu erzielen, wurden Szenen mit weniger Reibung zwischen Werkzeug und Werkstück und kleinere Sampledistanzen des Werkzeugs getestet. Dieses Hochfließen ist aber erst bei stärkeren Neigungen des Werkzeugs zu beobachten. Abbildung 7.24 zeigt Screenshots von Simulationen, die sich nur im Spanwinkel unterschieden. Die Spalten zeigen von links nach rechts Spanwinkel von -5° , 0° , 5° und 30° . In der ersten Zeile ist die Ausgangssituation, vor dem Kontakt zwischen Werkzeug und Werkstück, zu sehen. Die weiteren Zeilen zeigen den zeitlichen Ablauf mit einer Schrittlänge von 0,0025 s pro Zeile.

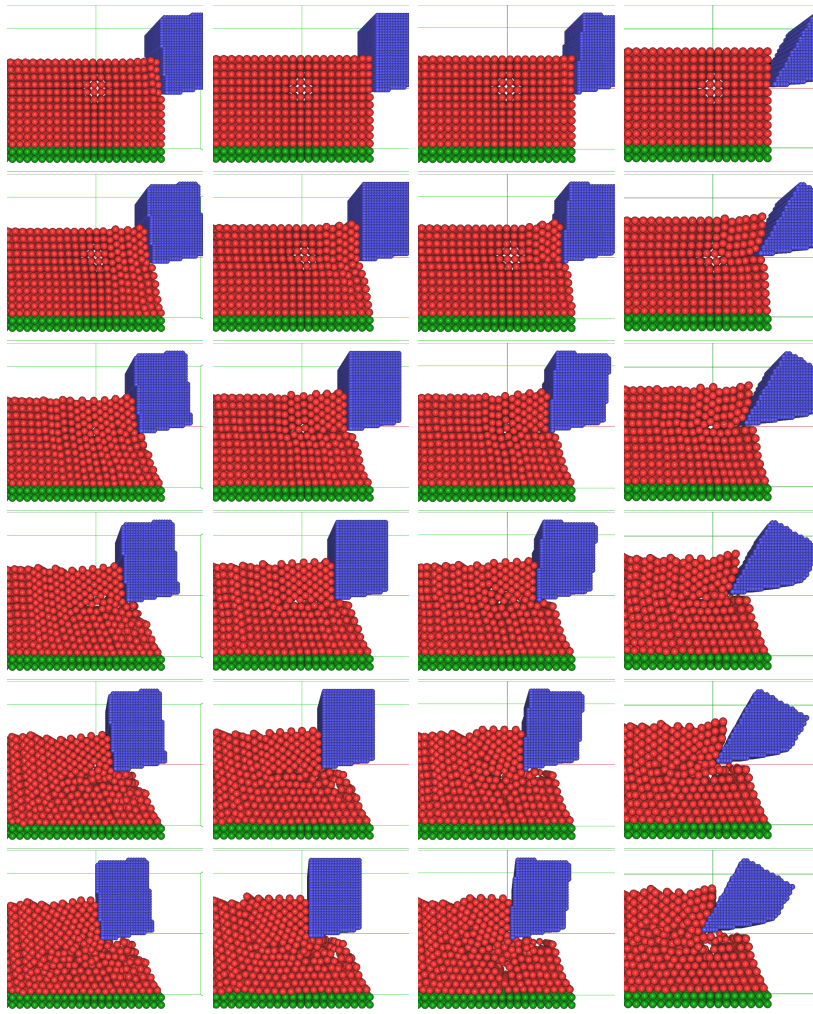


Abbildung 7.24.: Variation des Spanwinkels.

Im Gegensatz zur Variation des Spanwinkels ergibt eine Variation des Freiwinkels keine relevante Änderung des Spanbildungsverhaltens, was dem erwarteten Verhalten entspricht.

Werkstückvariationen beim MCA-Verfahren Es wurden auch Simulationen durchgeführt, bei denen die Werkstückoberfläche anfangs eine gewellte Struktur aufwies. Diese Variationen der Werkstückoberfläche ergeben keine relevanten Änderungen am Spanbildungsverhalten. Interessant ist in diesem Zusammenhang eher, dass sich bei anfänglich glatter Werkstückoberfläche hier manchmal Wellenstrukturen ausbilden, sobald das Material gestaucht wird (siehe Abb. 7.22).

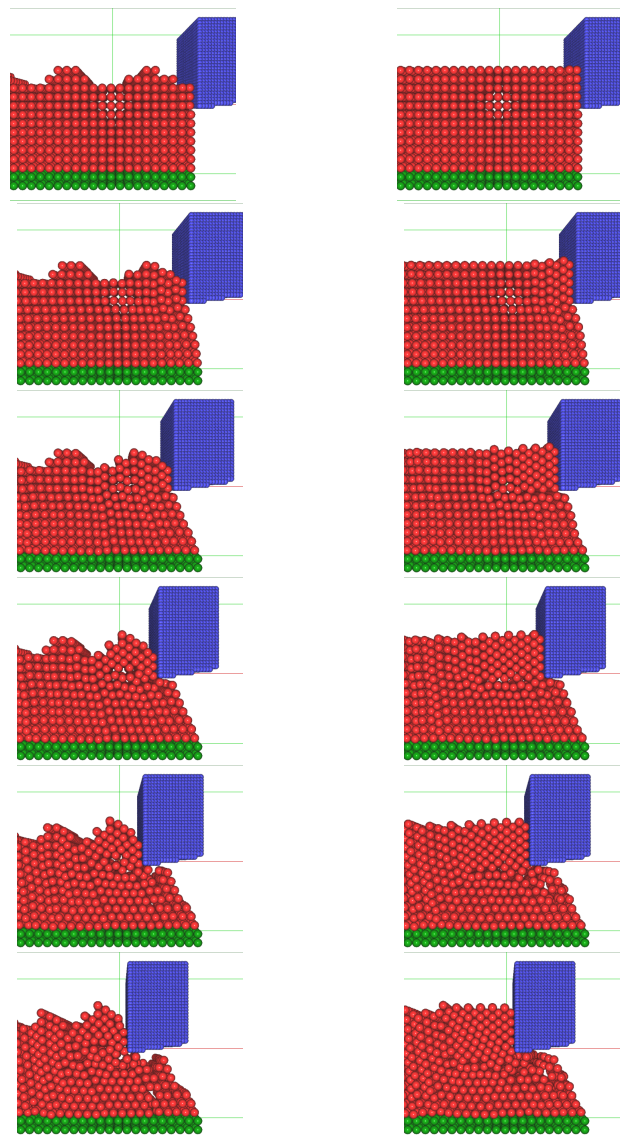


Abbildung 7.25.: Werkstückvariationen beim MCA-Verfahren

7.4.2. Spanbildungsverhalten beim SPH-Verfahren

Die Simulation von mehreren praxisnahen Fräsprozessen sollte der Beurteilung der Eignung dieser Implementierung zur Simulation von Spanbildungsverhalten dienen. Die ursprünglich geplanten Simulationen von drei verschiedenen Materialien bei drei verschiedenen Vorschüben und einer Zustellung von $\frac{3}{10}$ mm bei drei verschiedenen Spanwinkeln wurde zugunsten von drei Simulationen, die bewertbare Ergebnisse ergaben, verworfen. Die ursprünglich geplanten Simulationen sind mit dieser Implementierung aus folgenden Gründen nicht realisierbar und zeigen somit einige der Grenzen von SPH auf einem Arbeitsplatzrechner auf. Ein Grund besteht in der Zustellung von $\frac{3}{10}$ mm. Da zur Ausbildung eines Spans mehrere Schichten von Partikeln benötigt werden, muss das Werkstück sehr fein diskretisiert werden. Eine Diskretisierung von $\frac{1}{10}$ mm, die drei Schichten von Partikeln entspricht, war geplant. Folglich musste der Zeitschritt sehr klein gewählt werden, um die Bewegung von Partikeln in einem Simulationsschritt so einzuschränken, dass bei Bewegung nur einfache Kollisionen stattfinden. Eine nicht einfache Kollision stellt die Bewegung eines Partikels durch mehrere andere Partikel innerhalb eines Simulationsschrittes dar, was fälschlicherweise nicht ausgelösten Kollisionen entspricht. Der Zeitschritt sollte so klein sein, dass keine Kollisionen stattfinden, sondern die SPH-Kräfte diese vermeiden könnten. Für die geplanten Simulationen ist ein Zeitschritt von weniger als $1 \cdot 10^{-9}$ s nötig. Das impliziert eine Laufzeit von ca. 2,5 Monaten für eine Simulationszeit von 0,01 s bei einem Werkstück von ca. 270000 Partikeln und den Abmessungen von $2 \text{ cm} \times 1 \text{ cm} \times 1 \text{ cm}$ auf einem Intel i3-2100 mit 3,1 Ghz. Die Wahl eines kleineren Werkstücks für eine kleinere Laufzeit ist möglich und entspricht der Simulation nur eines Ausschnittes aus einem realen Spanbildungsprozess. In Fällen wo ein großes Werkstück in der Simulation auf Grund von Instabilitäten zwischen Partikeln insgesamt zu instabil oder weich im Verhalten wird, könnten diese Effekte durch eine geringere Anzahl der Partikel reduziert werden. Das Ziel der Simulation eines Ausschnittes des Werkstücks ist, durch die Einsparung an Partikeln, dieses in einer feineren Diskretisierung simulieren zu können. Anzumerken ist, dass bei einer höheren Anzahl von Partikeln die Laufzeit mit der Anzahl mindestens linear steigen würde. Ein weiterer Grund für das Verwerfen der ursprünglich geplanten Simulation besteht in der Anzahl der Partikel, die abhängig von der Diskretisierung des Werkstücks sind. Für diese SPH-Implementierung besteht bei ca. 300000 Partikeln eine Grenze in Bezug auf den zur Verfügung stehenden Arbeitsspeicher von 3,2 GB. Eine Verkleinerung des Werkstücks auf $1 \text{ cm} \times 0,5 \text{ cm} \times 0,5 \text{ cm}$ würde die Laufzeit und den Speicherverbrauch um ca. Faktor 8 senken. Eine genauere Beschreibung der Laufzeit und Speicheranforderungen werden im Kapitel 7.5.3 diskutiert.

Die folgenden Simulationen wurden als Ersatz zur Beurteilung der Eignung der Implementierung herangezogen. Die Spanwinkel, die untersucht wurden, waren -20° , 20° und 0° bei einem Freiwinkel von 10° . Es wurde ersatzweise eine Zustellung von 3 mm gewählt. Die Diskretisierung des Werkstücks wurde mit $\frac{5}{10}$ mm so gewählt, dass die Zustellung sechs Schichten von Partikel entspricht. Die unterste Ebene der Partikel im Werkstück

sowie die letzten beiden Spalten (Partikel ganz links) sind als feste Partikel gewählt, um das Werkstück einzuspannen. Die Gesamtmaße des Werkstücks sind $2\text{ cm} \times 1\text{ cm} \times 1\text{ cm}$. Das Werkzeug ist ausreichend groß gewählt worden, um über die ganze Oberfläche einen Span abtragen zu können. Es wird lediglich feiner aufgelöst, um die Schneide genauer zu modellieren. Die Anzahl der Partikel im Schneidkeil ist vernachlässigbar, da der Schneidkeil hohl implementiert ist. Vielmehr empfiehlt es sich eher den Schneidkeil feiner als das Werkstück zu diskretisieren, um Kraftverluste an der Schneidkante zu minimieren. Die folgende Abbildung 7.26 stellt die vorausgegangenen Grundeinstellungen dar, bei denen sich nur der Spanwinkel unterscheidet.

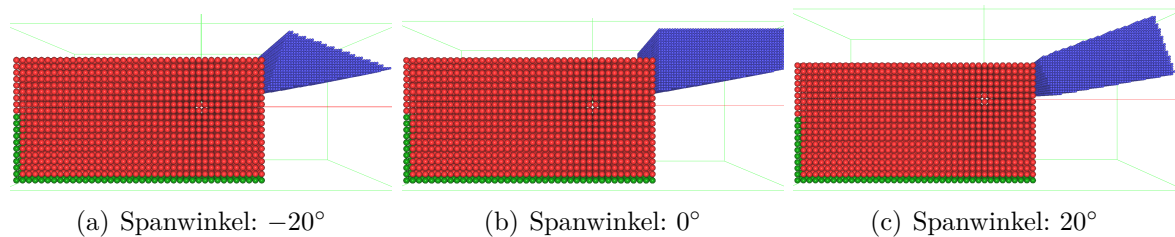


Abbildung 7.26.: Variation des Spanwinkels

Die allgemeinen Parameter der Simulation sind jeweils ein Zeitschritt `stepTime` von $5 \cdot 10^{-8}$ s. Ein geringerer Zeitschritt erhöht lediglich die Rechenzeit. Ein höherer Zeitschritt ist aufgrund des großen `youngsModulus` [MKN⁺04] und der geringen Diskretisierung nicht praktikabel. Für das zu simulierende Material Stahl ist der Materialparameter des `youngsModulus` ca. $210\text{ GPa} = 210 \frac{\text{kN}}{\text{mm}^2}$ und erfordert somit einen sehr kleinen Zeitschritt. Der Zeitschritt wurde über die Ermittlung eines maximal möglichen Zeitschrittes hinaus nicht variiert. Die `smoothingLength` wird mit $0,00091\text{ m}$ entsprechend der Diskretisierung so gewählt, dass initial ein Partikel 26 Nachbarn hat.

Die von der Diskretisierung abhängigen Parameter sind:

Parameter	Formelzeichen	Wert
<code>stepTime</code>		$5 \cdot 10^{-8}\text{ s}$
<code>smoothingLength</code>	h	$0,00091\text{ m}$

Die Gravitation `gravity` ist für eine Spanbildung vernachlässigbar und wurde bei den Simulationen nicht berücksichtigt. Der Parameter `gamma` der Druckberechnung wird üblicherweise mit 7 gewählt, vergleiche [Sol10].

Folgenden Parameter des Werkzeugs und Werkstücks sind berechnungsspezifisch gewählt:

Parameter	Formelzeichen	Wert
<code>gamma</code>	γ	7
<code>gravity</code>		0

Die sonstigen Parameter des Werkzeugs und Werkstücks entsprechen denen von Stahl und sind folgender Materialparameterdatenbank entnommen: <http://www.matweb.com/>. Lediglich die Werte für creepGamma und yieldgamma sind mangels recherchierbarer Werte frei gewählt und müssten eigentlich durch Simulationen bestimmt werden. Beide Parameter steuern die Berechnung des youngsModulus und zählen somit zu den materialspezifischen Parametern. Die beiden Parameter orientieren sich an den Ergebnissen der Block-auf-Boden-Versuche, siehe Abschnitt 7.2.2. Bessere Werte konnten nicht bestimmt werden.

Die materialspezifischen Parameter sind:

Parameter	Formelzeichen	Wert
density	ρ_i	7850
poissonRatio	ν	0,28
viscosity	μ_i	11000
viscosityStiffness	k	95000
youngsModulus	E	$2 \cdot 10^{11}$
yieldGamma	γ_y	0,01
creepGamma	γ_c	0,1

Die Dauer einer Simulation liegt bei über 48 Stunden, bis eine Bewertung durch einen Benutzer möglich ist. Das Werkstück verhält sich wie hartes Gummi, das bei einer gewissen Belastung reißt, und nicht wie ein Werkstück aus Stahl, das einen Span ausbildet. Ungeachtet der schlechten Ergebnisse der Würfelversuche, die in dieser Implementierung schon ein schlechtes SPH-Festkörperverhalten aufzeigen, kann, sofern die der Materiparameterdatenbank entnommenen Parameter nicht an einem schlechten Verhalten schuld sind, das Problem der Parameterwahl auf creepGamma und yieldGamma eingeschränkt werden. Ein zu geringer creepGamma-Wert führt zu einer Oszillation der Partikel im Werkstück und folglich einer Explosion. Ein zu großer Wert, wie hier gewählt, führt dazu, dass das Material sich plastisch verformt, anstatt einen Span auszubilden, also zu reißen. Aufgrund der langen Simulationszeit, die nötig ist, um einen einzigen anderen Wert für creepGamma zu testen, ist eine Optimierung nicht trivial. Selbiges gilt für yieldGamma, das den Übergang zwischen Elastizität und Plastizität steuert. Eine Abhängigkeit von creepGamma von yieldGamma kann nicht ausgeschlossen werden, da es hier nicht explizit untersucht wurde. Dass auch andere Parameter optimiert werden müssen, kann ebenfalls nicht ausgeschlossen werden. Lediglich die Materialparameter können mit Vorsicht aus Materialtabellen übernommen werden.

Abbildung 7.27 zeigt, dass sich das Werkstück staucht und unterhalb der Schneidkante in regelmäßigen Abständen reißt. Dieses Verhalten ist zumindest optisch betrachtet vom Spanwinkel unbeeinflusst. Es entspricht dem erwarteten Verhalten der plastischen Verformung.

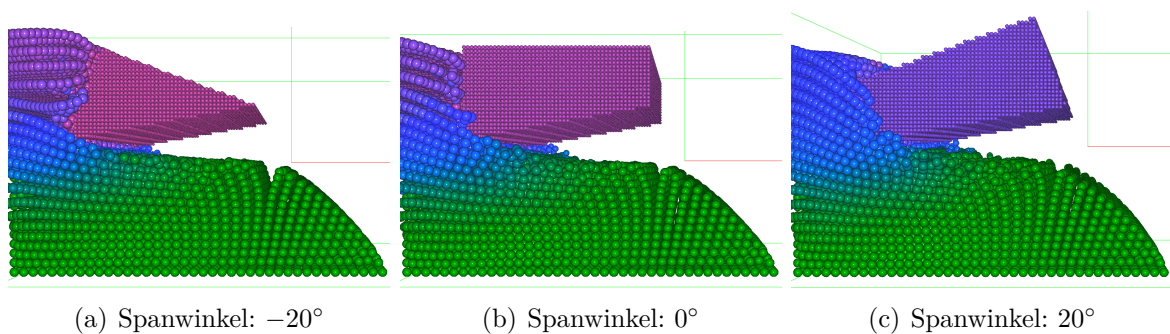


Abbildung 7.27.: Variation des Spanwinkels: Ergebnis nach 100 ms Simulationszeit

7.5. Diskussion der Ergebnisse

In diesem Abschnitt werden die Vorzüge und Nachteile von MCA- und SPH-Verfahren, welche in den beschriebenen Testfällen ersichtlich wurden, gegenübergestellt und bewertend verglichen. Als Erstes folgt eine Bewertung der Festkörpereigenschaften und eine Betrachtung, inwiefern verschiedene Materialeigenschaften realistisch dargestellt werden können. Hiernach wird ein Blick auf die Eignung von MCA- und SPH-Verfahren bei Schneid- und Spanbildungssimulationen geworfen. Abschließend erfolgt eine Gesamtbewertung mit einer Übersicht der darstellbaren Materialeigenschaften der beiden Verfahren. Soweit nicht anders angegeben, beziehen sich die folgenden Aussagen zu SPH und MCA auf die Implementierung der Projektgruppe, nicht generell auf SPH und MCA.

7.5.1. Grundlegende Materialeigenschaften

Mit SPH lassen sich Flüssigkeiten realistisch simulieren (siehe Versuch 7.2.1). Da SPH für Flüssigkeits- und Gassimulationen entwickelt wurde, stellt dies keine Überraschung dar. Bei MCA ist eine Simulation einer zähen Flüssigkeit bei geeigneter Parameterwahl ebenfalls möglich (siehe Versuch 7.1.2). Dies ist eine unerwartete Eigenschaft, da MCA für die Simulation von Festkörperreibung entworfen wurde.

SPH-Festkörper stellen allerdings ein Problem dar. Das beispielsweise in [Sol10] beschriebene Verhalten von Festkörpern konnte in der Projektgruppe nicht vollständig umgesetzt werden. Selbst in Ruhelage blieben die Festkörper nicht stabil (Versuch 7.2.2). Es ist aber nicht auszuschließen, dass bei geeigneter Parameterwahl eine Reproduzierbarkeit der Ergebnisse aus [Sol10] erreicht werden kann. Bis zum Abschluss der Projektgruppe ist dies allerdings nicht gelungen. Mit MCA ließen sich stabile Körper in Ruhelage (siehe Versuch 7.1.1) simulieren. Dies gelang auch nach einer Krafteinwirkung (siehe Versuch 7.1.2), hier ein Fall auf einen stabilen Boden, ohne Anzeichen einer plastischen Deformation. Bei der reinen Betrachtung von Festkörpern, ohne Einwirkung beispielsweise eines Schneidkeils, ist die MCA-Implementierung der Projektgruppe der SPH-Implementierung überlegen.

7.5.2. Schneid- und Spanbildungseigenschaften

Plastische Deformationen lassen sich mit MCA nur schwierig simulieren, da für einen stabilen Körper entsprechend gewählte Parameter den Körper zu starr abbilden (siehe Versuch 7.1.3), während bei einem weichen Körper im Laufe der Simulation extreme Instabilitäten entstehen (siehe Versuch 7.1.3). Desweiteren lässt sich folgern, dass das für ein Spanbildungsverhalten nötige Abtragen des Spans mit dieser Implementierung nicht realisierbar ist, da ein Kippverhalten eines Körpers nicht simuliert werden konnte (siehe Versuch 7.1.3).

Mit SPH lässt sich zwar ansatzweise ein Kippverhalten simulieren (siehe Versuch 7.2.3), jedoch wirkt der simulierte Körper extrem instabil. Dies liegt daran, dass der Green-St.-Venant-Spannungstensor nicht rotationsinvariant ist, was beim Kippen eines elastischen Objekts zu verfälschten Kräften führt [BIT09]. Eine plastische Deformation lässt sich ebenso simulieren (siehe Versuch 7.2.3). Allerdings lassen die im Laufe der Simulation auftretenden Instabilitäten vermuten, dass ein realistisch dargestelltes Spanbildungsverhalten mit der aktuellen Implementierung der Projektgruppe nur schwer realisierbar ist. Bei den Schnittversuchen konnte bei beiden Verfahren kein Abrollen eines Spans simuliert werden. In der SPH-Implementierung von [LES⁺09] wird genau dieses Problem beschrieben. Dort wird es auf die klassische SPH-Formulierung und den dort auftretenden Problemen mit Randpartikeln zurückgeführt, vor allem auf die Kräfte, die dort nur von einer Seite wirken. Eine dort als *Renormalized SPH formulation* bezeichnete Approximation liefert bessere Ergebnisse bei Randpartikeln sowie bei Partikeln, die völlig ungeordnet sind. Damit lässt sich dann ein Span simulieren, der auch abrollt und sich nicht nur entlang des Schneidkeils bewegt. Die Berechnungen wurden dort mit der Software LS-DYNA durchgeführt, die uns nicht zur Verfügung stand. Die oben erwähnte Rotationsvarianz des Spannungstensors wirkt offenbar auch gegen das Abrollen eines Spans. Der Einsatz der von Becker et al. entwickelte Zerlegung des Spannungstensors in Rotations- und Verformungskomponenten könnte somit eine Verbesserung des Systems darstellen [BIT09]. Weiterhin beschrieben Espinosa et al. in [LES⁺09] auch numerische Instabilitäten, die je nach Geschwindigkeit des Schneidkeils verschieden stark ausgeprägt sind. Mit verschiedenen Geschwindigkeiten des Werkzeugs konnten aufgrund der langen Laufzeiten für eine Simulation (siehe Abschnitt 7.5.3) in der Projektgruppe keine umfassenden Tests durchgeführt werden. Es ist demnach nicht auszuschließen, dass eine andere Geschwindigkeit bei der gegebenen Implementierung der Projektgruppe bessere Ergebnisse liefern kann.

Beim MCA-Verfahren erscheint eine andere Kugelpackung durch den Generator ein geeigneter Lösungsansatz für das Hauptproblem der Inkompressibilität (siehe Abschnitt 7.4.1) zu sein. Versuche hierzu waren im gegebenen Zeitrahmen (siehe Anhang B) jedoch nicht mehr möglich. Bei der SPH-Methode verursacht die Stabilität des Materials Probleme, da realistische Werte der relevanten Parameter extrem kleine Zeitschritte im Bereich von $1 \cdot 10^{-8}$ s erforderlich machen. Die damit einhergehenden sehr hohen Laufzeiten (siehe Kapitel 7.4.2), bis ein verwertbares Ergebnis sichtbar ist, erlaubten auch hier

nicht, umfassende Parametertests durchzuführen. Trotz der genannten Einschränkungen lassen sich bei beiden Methoden in die richtige Richtung weisende Simulationsverhalten beobachten. Ein „Hochfließen“ eines Spans am Schneidkeil ist bei der MCA-Methode bei entsprechenden Parametern abbildbar (siehe Abbildung 6.10). Bei SPH besteht die Vermutung, dass bei geeigneter Wahl der Parameter `creepGamma` und `yieldGamma` dieses auch möglich sein könnte (siehe Abschnitt 7.4.2). Bei MCA-Simulationen lässt sich teilweise die Ausbildung einer Wellenstruktur auf der Werkstückoberfläche vor dem Schneidkeil beobachten (siehe Abbildung 7.28).

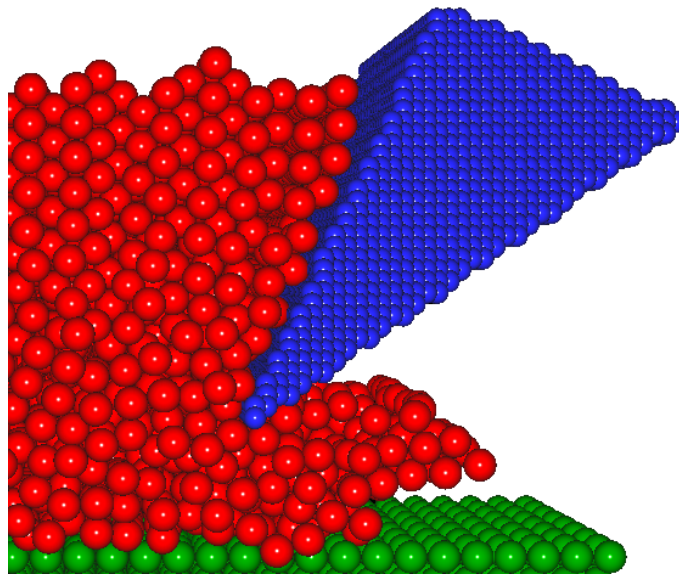


Abbildung 7.28.: Ausbildung einer Wellenstruktur auf der Werkstückoberfläche

Diese Ergebnisse ermöglichen noch keine grundlegende Aussage über die Möglichkeit akkurater Simulationen von Spanbildung, beispielsweise bei Fräsprozessen, sodass weitere Parametertests sinnvoll erscheinen. Beim SPH-Verfahren könnte die zusätzliche Verwendung der erwähnten Tensorzerlegung von Becker et al. in [BIT09], sowie die von Monaghan et al. in [Mon00] beschriebenen Methoden zur Abmilderung der tensilen Instabilität zu besseren Ergebnissen führen.

Die gewählte Modellierung des Werkzeugs aus Partikeln wirkt sich bei beiden Simulationsmethoden möglicherweise ungünstig auf die Energieübertragung zwischen Werkzeug und Werkstück aus. Das Spanbildungsverhalten wurde negativ durch Verkantungen der Partikel unterschiedlicher Objekte beeinflusst. Es wird vermutet, dass die in Kapitel 4.3 diskutierte Verwendung impliziter Werkzeugflächen, passend in das jeweilige Verfahren integriert, Abhilfe schaffen könnte.

7.5.3. Gesamtbewertung

Die während der Projektgruppendauer durchgeführten Tests lassen für beide Alternativmethoden ungefähre Schätzungen der Brauchbarkeit zu. Die gewählte MCA-Implementierung erscheint nach Betrachtung der Ergebnisse für eine annähernd realistische Simulation von Festkörpern geeigneter zu sein, auch wenn nicht alle Versuche absolut realistisches Verhalten aufwiesen, wie z.B. die Stabversuche aus Kapitel 7.1.4. Die Versuche der SPH-Implementierung zeigten oftmals zu große Schwierigkeiten mit Stabilitäten von Festkörpern, weswegen unter Anderem auch eine Simulation des Stabversuchs nicht möglich war. Flüssigkeiten sind mit beiden Verfahren prinzipiell simulierbar (siehe Abschnitt 7.5.1). Da allerdings das Hauptaugenmerk der Projektgruppe auf Festkörpern lag, wurde Flüssigkeitsverhalten nicht erschöpfend getestet und deshalb an dieser Stelle auch keine endgültige Aussage getroffen.

Beide Simulationsverfahren zeigen ein plastisches Deformationsverhalten, das aber im Vergleich zu realen Materialien zu ausgeprägt ist (siehe Abbildung 7.12(b)). Die gewählten Simulationsparameter sind nicht dafür geeignet, ein elastisches Simulationsverhalten darzustellen, wie es augenscheinlich zu erwarten ist (siehe Abbildung 7.27). Bei einer besseren Parameterwahl oder einer verbesserten Simulation durch zusätzliche Kräfte [MGS01] könnte dieses vermutlich erreicht werden. Jedoch liegen realistische Parameter für den Elastizitäts- bzw. Schubmodul von Metallen im Bereich von einigen GPa, was in Kombination mit den ansonsten sehr kleinen vorkommenden Größen numerische Instabilitäten begünstigt. Auch die Integrationsschemata werden bei zu großen Wertebereichen unzuverlässig und müssen gegebenenfalls durch rechenaufwändigere Varianten mit höherer numerischer Stabilität ersetzt werden [MKN⁺04]. Aufgrund der genannten methodischen Defizite der beiden Implementierungen ist eine realitätsnahe Spannbildung nicht simulierbar.

Zusätzlich weisen beide implementierte Methoden eine hohe Laufzeit und eine hohe Speicheranforderung auf, wie bereits in Kapitel 7.4.1 und Kapitel 7.4.2 festgestellt wurde. Als Beispiele können die in den erwähnten Kapiteln durchgeführten Spannbildungsversuche dienen, bei denen folgende Laufzeiten festgestellt wurden: Bei der MCA-Simulation mit 104913 Partikeln ergibt sich auf einem Intel Core 2 Duo mit 2,1 GHz und 4 GB Arbeitsspeicher eine Laufzeit von 46 Stunden und 47 Minuten für eine Simulationszeit von 0,04945 s. Eine SPH-Simulation von ca. 270000 Partikeln auf einem Intel i3-2100 mit 3,2 GB Speicher dauerte mehr als 48 Stunden.

Sowohl das Laufzeitverhalten, als auch der Speicherverbrauch ergeben sich stärker aus den Eigenschaften der Verfahren als aus Implementierungsdetails. Beide Verfahren müssen komplexe Zustandsbeziehungen von Elementpaaren abspeichern und aktualisieren. Im MCA-Ansatz werden Deformationswinkel (γ_{ij}) und Verbindungszustände von Automatenpaaren benötigt. Der SPH-Ansatz kommt für die Simulation von Festkörpern nicht ohne paarweise betrachtete Dehnungsfelder \vec{u}_{ij} aus. Die genannten Größen sind dafür verantwortlich, dass unabhängig von der gewählten Umsetzung der Platz- und Zeitbedarf

einer jeden Implementierung mindestens in quadratischer Größenordnung von der Sampleanzahl abhängt. Es existieren zwar nicht zwischen allen Elementen Beziehungen, aber je genauer die räumliche Diskretisierung erfolgt, desto mehr Samples gibt es und desto mehr Beziehungen pro Element müssen potentiell beachtet werden. Dementsprechend wächst für den Term $c \cdot n^2$, der qualitativ den Zusammenhang zwischen der Sampleanzahl (n) und dem Ressourcenbedarf beschreibt, nicht nur der quadratische Anteil, sondern auch der Gewichtsanteil c , durch den die nur partielle Verbundenheit von Elementen zum Ausdruck kommt. Ferner summieren sich Rundungsfehler und numerische Instabilitäten ebenfalls über Paarbeziehungen, weshalb ihr Anstieg qualitativ dem gleichen Term folgt. Die Kompensation von numerischen Fehlern muss also durch kleiner gewählte Zeitschritte erfolgen, was sich erneut negativ auf das Laufzeitverhalten auswirkt.

Den hohen Ressourcenanforderungen der Simulationsverfahren kann nur bedingt durch die Verwendung von Rechenclustern begegnet werden. Rechnerisch ergibt sich, wenn die Elemente i und j , sowie j und k miteinander in Paarbeziehungen stehen, eine transitive Abhängigkeit des Zustands von Element i vom Zustand des Elements k . Solche transitiven Abhängigkeitsketten erstrecken sich jeweils mindestens über alle Elemente aus zusammenhängenden Objekten einer Szene. Eine parallele Verarbeitung durch mehrere Rechnerknoten kann dementsprechend nur erfolgen, wenn der Szenenzustand, der wie oben angeführt viel Speicherplatz benötigt, nahezu vollständig in jedem Berechnungsschritt über die Rechnerknoten propagiert wird. Die Implementierung der Projektgruppe hat gezeigt, dass der Zeitbedarf pro Rechenschritt mit aktueller Hardware und moderaten Szenengrößen den einstelligen Sekundenbereich nicht überschreitet. Es ist also anzunehmen, dass die Ressourcenkosten für die Zustandspropagation zwischen verteilten Rechnerknoten höchstens unter Anwendung trickreicher Optimierungsverfahren überhaupt auf ein Maß gesenkt werden können, das einen Performanzgewinn zulässt.

Die beschriebenen Laufzeit- und Speicherplatzprobleme könnten in Zukunft durch Weiterentwicklungen im Hardwarebereich weniger stark ins Gewicht fallen, sodass es keine Veranlassung gibt, die untersuchten Verfahren prinzipbedingt aus dem Fokus des Forschungsinteresses geraten zu lassen, selbst wenn sich ihre Umsetzung und Erprobung momentan noch als sehr große Herausforderung erweist.

Im Folgenden wird eine Einschätzung der Simulationseigenschaften der Implementierung der Projektgruppe auf Grundlage der durchgeführten Tests übersichtlich dargestellt.

Eigenschaften	SPH (PG)	MCA (PG)
Flüssigkeit	gut geeignet	mit Einschränkungen
Festkörperstabilität	kaum geeignet	geeignet
elastische Deformation	zu geringe Ausprägung	zu geringe Ausprägung
plastische Deformation	zu hohe Ausprägung, instabil	zu hohe Ausprägung
Spannbildung	nicht ansatzweise realitätsnah	nicht realitätsnah
Simulationsprobleme	Festkörperinstabilität	
Berechnungsprobleme	Speicher, Laufzeit	Speicher, Laufzeit

8. Zusammenfassung und Ausblick

Im Laufe der Projektgruppe sollte durch Benutzung von partikelbasierten Methoden eine Spanbildungssimulation entworfen und realisiert werden, mit der plasto-elastische Objektinteraktionen simuliert werden können. Speziell sollte die Simulation fähig sein, verschiedene Materialien und verschiedene Ausprägungen einer Spanbildung abzubilden. Dafür benötigte Komponenten, wie eine Nachbarschaftssuche zur lokalen Auswertung von Nachbarschaften von Partikeln und einer Visualisierungsumgebung, konnten von der Projektgruppe frei gewählt und in das Projekt eingebunden werden. Das Programm wurde mit der Programmiersprache C++ und der Klassenbibliothek Qt programmiert und mit der Grafikkbibliothek OpenGL visualisiert. Die eigentliche Modellierung der Simulation wurde letztlich durch Smoothed Particle Hydrodynamics (SPH) und Movable Cellular Automata (MCA) realisiert.

Die Simulation ist im Stande, primitive physikalische Tests, wie das Fallenlassen eines Würfels auf eine Oberfläche oder das Füllen eines Behälters mit einer Flüssigkeit, abhängig von den gewählten Simulationsparametern zu simulieren. Komplexe Prozesse wie eine Spanbildung lassen sich bislang kaum abbilden. Im Laufe der Auswertung zeigte sich, dass das Simulationsverfahren die Spanbildung als realen Prozess nicht abbilden konnte. Mögliche Ursachen, neben der Parameterfindung, könnten die Kollisionserkennung und -behandlung sowie die SPH-Berechnungsformeln sein.

Wie sich in der Diskussion herausgestellt hat, wird die Eignung vom MCA-Verfahren gegenüber dem SPH-Verfahren als etwas positiver bewertet. Durch intensives Parametersuchen könnten die bisherigen Ergebnisse beider Simulationsmodelle näher an ein optisch realistisches Ergebnis geführt werden.

Eine Reproduzierbarkeit der Ergebnisse anderer Forscher, deren Veröffentlichungen [Sol10, MKN⁺04, Mon05, Kel06, IAGT10, PHK⁺95] hier als Quellen genutzt wurden, war nicht möglich, da diese ihre Methoden und Parameter nicht veröffentlichen oder genau genug dokumentieren. Es konnte jeweils lediglich ein ähnliches Verhalten erreicht werden, das aber bei weitem nicht so ausgeprägt ist.

Ausblick

Für ein realistischeres Spanbildungsverhalten, das auch verschiedene Spantypen realitätsnah simulieren kann, sind einige Erweiterungen nötig. Das realistischere Spanbildungsverhalten könnte durch eine Verwendung weiterer physikalischer Größen, wie Temperatur,

erreicht werden.

Eine Optimierung der Laufzeit durch eine GPU-Implementierung der Nachbarschaftssuche ist eine weitere Möglichkeit der Verbesserung der hier beschriebenen Ergebnisse. Von einer vollständigen Implementierung auf der GPU wird abgeraten, da hierdurch die simulierbare Partikelanzahl zu sehr begrenzt wird.

Schließlich könnte durch eine bessere Kugelpackung eine weitere Verbesserung der Ergebnisse erreicht werden, da bei den aktuellen Simulationsergebnissen die Partikel ihre Anordnung in eine optimale Kugelpackung ändern, bevor überhaupt eine Deformation auftritt.

Zu den weiteren Möglichkeiten, die vielleicht zur Verbesserung der Simulationsergebnisse beitragen können, zählt zum Einen die von Monaghan et al. in [Mon00] und [MGS01] vorgeschlagene Methode zur Verminderung der tensilen Instabilität der SPH-Methode. Zum Anderen kann der von Becker et al. in [BIT09] entwickelte Vorgang, um die Rotationsvarianz des nichtlinearen Cauchy-St.-Venant Spannungstensors zu beseitigen, im Simulator genutzt werden.

Literaturverzeichnis

- [AM91] ABRAMOWSKI, S. ; MÜLLER, H.: *Geometrisches Modellieren*. BI-Wiss.Verl., 1991
- [AW09] ADAMS, B. ; WICKE, M.: Meshless Approximation Methods and Applications in Physics Based Modeling and Animation. In: *EG 09 Proceedings of the Eurographics Conference (2009)*, S. 213–239
- [Bal96] BALZERT, H.: *Lehrbuch der Software-Technik: Teil 1: Software-Entwicklung*. Spektrum Akademischer Verlag, 1996
- [BB07] BICKELHAUPT, F. M. ; BAERENDS, E. J.: Kohn-Sham Density Functional Theory: Predicting and Understanding Chemistry. In: *Reviews in Computational Chemistry* Bd. 15. John Wiley & Sons, Inc., 2007, S. 1–86
- [Bic07] BICANIC, N.: *Discrete element methods*. 2007
- [BIT09] BECKER, M. ; IHMSEN, M. ; TESCHNER, M.: Corotated SPH for Deformable Solids. In: *Eurographics Workshop on Natural Phenomena (2009)*
- [Cas03] CASTELLANI, M.: *CHM 448, Chapter 8 - Chemical Forces*. Version:2003. <http://science.marshall.edu/castella/chm448/chap8.pdf>
- [Cas08] CASTELLI, F.: *QJSON*. <http://qjson.sourceforge.net/>, 2008
- [CLMP95] COHEN, J. D. ; LIN, M. C. ; MANOCHA, D. ; PONAMGI, M.: I-COLLIDE: An Interactive and Exact Collision Detection System for Large-scaled Environments. In: *In Proc. of ACM Interactive 3D Graphics Conference, 1995*, S. 189–196
- [Cor12] CORPORATION, Intel: *Intel™Core i7-2600 Desktop Processor Series*. http://download.intel.com/support/processors/corei7/sb/core_i7-2600_d.pdf, 2012
- [Cro12] CROCKFORD, D.: *Einführung in json*. <http://www.json.org/json-de.html>. Version: August 2012
- [DG96] DESBRUN, M. ; GASCUEL, M.-P.: Smoothed Particles: A new paradigm for animating highly deformable bodies. In: *Eurographics Workshop on Computer Animation and Simulation (1996)*, S. 61–76

- [Edg12] EDGEWALL: *The Trac Project*. <http://trac.edgewall.org/>.
Version: August 2012
- [Eri05] ERICSON, C.: *Real-Time Collision Detection*. Morgan Kaufmann, 2005
- [Ful79] FULLER, R. B.: *SYNERGETICS - Explorations in the Geometry of Thinking*.
Macmillan Publishing Co. Inc. 1975, 1979
- [GM77] GINGOLD, R. A. ; MONAGHAN, J. J.: Smoothed particle hydrodynamics:
theory and application to non-spherical stars. In: *Monthly notices of the
royal astronomical society* 181 (1977)
- [HCL⁺05] HERRANZ, S. ; CAMPA, F. J. ; LACALLE, L. L. N. ; RIVERO, A. ; LAMIKIZ, A. ;
UKAR, E. ; SANCHEZ, J. A. ; BRAVO, U.: The milling of airframe components
with low rigidity: a general approach to avoid static and dynamic problems.
In: *Proceedings of the Institution of Mechanical Engineers* (2005), S. 789–801
- [HKK07] HARADA, T. ; KOSHIZUKA, S. ; KAWAGUCHI, Y.: Smoothed particle hydro-
dynamics on GPUs. In: *Computer Graphics International*, 2007, S. 63–70
- [IAGT10] IHMSEN, M. ; AKINCI, N. ; GISSLER, M. ; TESCHNER, M.: Boundary handling
and adaptive time-stepping for PCISPH. In: *Workshop on Virtual Reality
Interaction and Physical Simulation VRIPHYS* (2010)
- [Jag08] JAGODZINSKI, F.: *Molecular Dynamics Equations*. Version: 2008. <http://www.cs.umass.edu/~filip/MDEquations.pdf>
- [Kel06] KELAGER, M.: *Lagrangian Fluid Dynamics Using Smoothed Particle Hydro-
dynamics*, Universität Copenhagen, Diplomarbeit, 2006
- [Kie11] KIERFELD, J.: *Vorlesungsskript: Computational Physics*. http://t1.physik.tu-dortmund.de/kierfeld/teaching/CompPhys_11/. Version: 2011
- [KK08a] KLOCKE, F. ; KÖNIG, W.: *Fertigungsverfahren 1*. 8.0. Springer Verlag, 2008
- [KK08b] KÖNIG, W. ; KLOCKE, F.: *Fertigungsverfahren - Drehen, Fräsen, Bohren*.
Berlin/Heidelberg : Springer Verlag, 2008
- [LES⁺09] LIMIDO, J. ; ESPINOSA, C. ; SALAÜN, M. ; MABRU, C. ; CHIERAGATTI,
R.: High speed machining modelling : SPH method capabilities. In: *4th
Smoothed Particle Hydrodynamics European Research Interest Community
(SPHERIC) workshop* (2009)
- [LESL07] LIMIDO, J. ; ESPINOSA, C. ; SALAUN, M. ; LACOME, J.: SPH method applied
to high speed cutting modelling. In: *International Journal of Mechanical
Sciences* 49 (2007), S. 898–908
- [LMCG01] LIN, M. C. ; MANOCHA, D. ; COHEN, J. ; GOTTSCHALK, S.: Collision
Detection: Algorithms and Applications. In: *Proceedings of Algorithms of*

Robotic Motion and Manipulation, 2001, S. 129–142

- [Luc77] LUCY, L. B.: A numerical approach to the testing of the fission hypothesis. In: *The Astronomical Journal* 82 (1977), Nr. 12
- [Lur05] LURIE, A. I.: *Theory of Elasticity*. Springer, 2005
- [MB05] McREYNOLDS, T. ; BLYTHE, D.: *Advanced Graphics Programming Using OpenGL*. Elsevier Morgan Kaufmann Publishers, 2005
- [MFJK12] MÜLLER, H. ; FISSELER, D. ; JOLIET, R. ; KERSTING, P.: *Projektgruppe ChipSim – WS 2011/2012, SS 2012*. <http://www.isf.de/de/studium/seminare/chipsim.html>. Version: August 2012
- [MGS01] MONAGHAN, J. J. ; GRAY, J. P. ; SWIFT, R. P.: SPH Elastic Dynamics. In: *Computational Methods of Applied Mechanical Engineering* 190 (2001), Nr. 49 – 50, S. 6641 – 6662
- [MKN⁺04] MÜLLER, M. ; KEISER, R. ; NIELEN, A. ; PAULY, M. ; GROSS, M. ; ALEXA, M.: Point Based Animation of Elastic, Plastic and Melting Objects. In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2004), S. 141–151
- [Mon00] MONAGHAN, J.J.: SPH without a Tensile Instability. In: *Journal of Computational Physics* 159 (2000), Nr. 2, S. 290 – 311
- [Mon05] MONAGHAN, J. J.: Smoothed particle hydrodynamics. In: *Reports on Progress in Physics* 68 (2005), Nr. 8, S. 1703
- [Mor99] MORTENSON, M. E.: *Mathematics for Computer Graphics Applications*. 2. Industrial Press, 1999
- [Muj12] MUJA, M.: *FLANN – Fast Library for Approximate Nearest Neighbors*. <http://www.cs.ubc.ca/~mariusm/index.php/FLANN/FLANN>. Version: August 2012
- [NMK⁺05] NEALEN, A. ; MÜLLER, M. ; KREISER, R. ; BOXERMAN, E. ; CARLSON, M.: Physically Based Deformable Models in Computer Graphics. In: *Computer Graphics Forum* 25 (2005), S. 809–836
- [OBH02] O’BRIEN, James F. ; BARGTEIL, Adam W. ; HODGINS, Jessica K.: Graphical Modeling and Animation of Ductile Fracture. In: *ACM Transactions on Graphics* 21 (2002), Nr. 3, S. 291–294
- [PHK⁺95] PSAKHIE, S. G. ; HORIE, Y. ; KOROSTELEV, S. Y. ; SMOLIN, A. Y. ; DMITRIEV, A. I. ; SHILKO, E. V. ; ALEKSEEV, S. V.: Method of Movable Cellular Automata as a Tool for Simulation with the Framework of Mesomechanics. In: *Russian Physics Journal* 38 (1995), Nr. 18, S. 1157 – 1168

- [Pop09] POPOV, V. L.: *Kontaktmechanik und Reibung*. 2. Springer, 2009
- [SA06] SEGAL, M. ; AKELEY, K.: *The OpenGL Graphics System: A Specification*. <http://www.opengl.org/registry/doc/glspec21.20061201.pdf>. Version: 2006
- [San11] SANDHU, T.: *ASUS (NVIDIA) GeForce GTX 560 graphics-card review*. <http://hexus.net/tech/reviews/graphics/30427-asus-nvidia-geforce-gtx-560-graphics-card-review/>, 2011
- [SD09] SEKERCIOGLU, A. S. ; DUYSAK, A.: Application of molecular modeling with mass-spring systems for computer simulation and animation. In: *International Journal of Physical Sciences* Bd. 4, 2009, S. 500–504
- [Smi06] SMITH, C.: *On Vertex-Vertex Systems and Their Use in Geometric and Biological Modelling*, University of Calgary, Diplomarbeit, 2006
- [Sol10] SOLENTHALER, B.: *Incompressible fluid simulation and advanced surface handling with SPH*, Universität Zürich, Diss., 2010
- [SS12] SUTHERLAND, Jeff ; SCHWABER, Ken: *The Scrum Papers: Nuts, Bolts, and Origins of an Agile Process*. 2012
- [Sta10] STAUBACH, D.: *Lagrangian Fluid Dynamics Using Smoothed Particle Hydrodynamics*. Erlangen-Nuernberg, Friedrich-Alexander-Universität, Bachelorarbeit, 2010
- [The12] THE KHRONOS GROUP: *OpenGL – The Industry Standard for High Performance Graphics*. <http://www.opengl.org>. Version: August 2012
- [VF08] VILLUMSEN, M. F. ; FAUERHOLDT, T. G.: *Simulation of Metal Cutting using Smooth Particle Hydrodynamics*. LS-Dyna Anwenderforum, Bamberg, 2008
- [VSC01] VASSILEV, T. ; SPANLANG, B. ; CHRYSANTHOU, Y.: Fast Cloth Animation on Walking Avatars. In: *Computer Graphics Forum*, 2001, S. 260–267
- [Wei05] WEIZEL, C.: *Kollisionserkennung mit Raytracing*, Universität Koblenz Landau, Diplomarbeit, 2005
- [WMK⁺02] WEINERT, K. ; MÜLLER, H. ; KREIS, W. ; SURMANN, T. ; AYASSE, J. ; SCHÜPPSTUHL, T. ; KNEUPNER, K.: Diskrete Werkstückmodellierung. Zur Simulation von Zerspanprozessen. In: *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb* (2002), Nr. 7/8, S. 385–389
- [WSM02] WAGNER, C. ; SCHILL, M. A. ; MÄNNER, R.: Collision detection and tissue modeling in a VR-simulator for eye surgery. In: *Proceedings of the workshop on Virtual environments 2002*, Eurographics Association, 2002 (EGVE '02), S. 27–36

- [ZLZ10] ZHAO, Xiangkun ; LI, Fengxia ; ZHAN, Shouyi: A New GPU-Based Neighbor Search Algorithm for Fluid Simulations. In: *DBTA'10*, 2010, S. 1–4

A. Pflichtenheft

Dieser Abschnitt fasst die wichtigsten Komponenten des Pflichtenheftes grob zusammen und gibt im späteren Verlauf einen Ausblick darüber, welche Ziele des Pflichtenheftes inwieweit erfüllt wurden.

A.1. Zielbestimmungen

Zu Anfang der Projektgruppendauer wurden gemeinschaftlich die folgenden Kriterien erarbeitet und in Muss-, Soll-/Wunschkriterien und spezielle Kriterien für den Prototyp unterteilt. Diese Kriterien sollten sich vor allem an den von der Projektgruppenleitung vorgegebenen Funktionsrahmen orientieren.

A.1.1. Muss-Kriterien

Die Muss-Kriterien beschreiben Anforderungen, die das Programm mindestens erfüllen muss, um den Mindestanforderungen der Projektgruppenziele gerecht zu werden. Dafür soll das Programm, unter Berücksichtigung eines spezifizierten Eingabeformats, Simulationsparameter aus einer externen Datei (z.B. einer Projektdatei) auslesen können. Dieser Simulationsrahmen soll mindestens mittels eines Generators aus primitiven geometrischen Objekten (Flächen, Quader) aufgebaut werden. Unbedingt variierbare Parameter der Simulationsumgebung sollen aus dem Zeitschrittintervall, der Partikelanzahl und einem Abbruchkriterium der Simulation bestehen. Zusätzlich soll sich das Programm an die vorhandene Hardware anpassen (im Sinne einer Abwärtskompatibilität für ältere Maschinen). Außerdem soll das Programm imstande sein, die während der Simulation erzielten Ergebnisse exportieren zu können, um so ggf. zu einem späteren Zeitpunkt eine weitere Auswertung zu gewährleisten.

A.1.2. Prototyp

Die Kriterien des Prototyps stellen im Grunde eine Teilmenge der Muss-Kriterien dar. Es war eine selektive Realisierung von zum Ende mindestens benötigter Features geplant. So sollte ein Prototyp noch keine externen Daten notwendigerweise auslesen müssen. Der Prototyp soll aus einer rudimentären Simulationsumgebung ohne spezielle Nutzer-

schnittstelle bestehen, die in der Lage ist (mindestens) eine Primitive (z.B.: ein Quader) als festen Körper darzustellen. Dieser Körper soll eine Art von Elastizität aufweisen, und mindestens einen von der Projektleitung festgelegten Demonstrationsfall darstellen können (z.B. einen springenden Block).

A.1.3. Optionale Kriterien

Unter den optionalen Kriterien sind solche Ziele zusammengefasst, die nur wenn möglich in das Programm integriert werden. Darunter können auch Funktionalitäten fallen, die während der Projektgruppendauer aufgrund von Zeitgründen nicht mehr realisierbar waren. So sollen zum Ende der Projektgruppendauer alle für die Spanbildung benötigten Geometrien (Werkzeug, Werkstück, inklusive ihrer Variationen) konstruiert werden können. Die durch eine Projektdatei geladenen Parameter sollen vollständig veränderbar sein, im Programm nach erfolgreichem Laden ebenfalls veränderbar und neu exportierbar sein. Das Programm soll eine Multicore CPU- und GPU Konfiguration anbieten, um Performance der Simulation zu verbessern. Die Visualisierung der Simulation soll verschiedene Visualisierungsarten anbieten können (punktbasierte Visualisierung, Festkörpervisualisierung), und während der Simulation alle vorhandenen Parameter (wie Geschwindigkeit, Beschleunigung, ...) farblich darstellen können. Zusätzlich soll das Programm eine Stapelverarbeitung sowie eine nachträgliche Visualisierung gespeicherter Simulationen anbieten können.

A.1.4. Demonstrationsfälle / Minimalziele

Von der Projektgruppenleitung wurde ein Satz von Demonstrationsfällen spezifiziert, die die minimale Funktionalität der Simulation eingrenzen. Darunter fallen folgende Fälle:

- Quader fällt auf Boden.
- Quader fällt auf Keil.

Dabei sollen die Materialien der simulierten Objekte weiche, gummiartige und harte Stoffeigenschaften bezüglich der Elastizität und Plastizität aufweisen. Die genaue physikalische Korrektheit ist dabei sekundär. Ein zusätzlicher Fall war eine auf den Boden fallende Kugel. Dieser Testfall wurde lediglich im Zwischenbericht knapp erwähnt, da er für die Spanbildung keine primäre Bewandnis darstellte.

A.1.5. Anmerkungen zum Pflichtenheft

Im Laufe der Projektgruppendauer wurden alle Muss-Kriterien erfüllt. Lediglich von den Kann-Kriterien wurde eine Visualisierung im Sinne von Festkörpern und eine GPU Konfiguration bzw. eine GPU Unterstützung nicht realisiert. Trotzdem wurde im Sinne der

Grafikkartenunterstützung eine leichtgewichtige Visualisierung angeboten für leistungsschwächere Grafikkarten. Die Visualisierung im Sinne eines Festkörpers (als Polygonnetz dargestellt) wurde vernachlässigt, da sonst rechenintensive Probleme wie das Remeshing auftreten könnten, das z.B. auch bei FEM-Simulationen für lange Iterationszeiten sorgt.

B. Zeitplan

Projektgruppe 562: Chipsim Zeitplan des SoSe 2012

Zeitraum	Implementierung SPH	Implementierung MCA	Endbericht
07.04-13.04	Altballast bewältigen. (Zwischenbericht Korrektur, Terminplan absegnen)		
14.04-20.04	Formeln überprüfen, Grundformeln komplettieren, internes Modell fixieren. Strukturdiagramm des Programmaufbaus.	Überprüfung des Systems	Struktur, Diskussion Übernehmen wiederverwendbarer Inhalte aus dem Zwischenbericht
21.04-27.04	Implementierung neuer Funktionalitäten: Werkzeug, Bewegung des Werkzeugs		
28.04-04.05	Kollisionsdetektion, sofern nötig		
05.05-11.05	GUI und Schnittstellen komplettieren		Formulierung: Testfälle
12.05-18.05	Konstruktion der ersten Spanbildung		(Was ist möglich, was kann dargestellt werden)
17.05-25.05	Spanbildung austesten, Testsznarien als Projektdateien erstellen		Formulierung: Grundlagenkapitel
26.05-01.06	Deadline Funktionalitäten und weiteren Aufwand einschätzen		Korrektur: Grundlagenkapitel
ab 07.06	Exportfunktionalität, <i>GPU / Multi-Core Support</i>		
09.06-15.06	Feinschliff / Bugfixing Implementierung letzter Features: <i>nachträgliche Visualisierung,</i> <i>Stapelverarbeitung</i>		Formulierung: Umsetzungskapitel, erste Teile des Analysekapitels, Zusammenfassung
15.06	Implementierungsstop		
16.06-22.06	Analyse der Testfälle erste Formulierungen Auswertungskapitel		
23.06-29.06	Evaluierung der Analyse, Formulierung: Auswertung / Fazit		
29.06	erste Finalversion des Endberichts		
06.07-13.07	Korrektur (und Vortragsvorbereitung)		
13.07	geplante Abgabe des Endberichts		

Der Zeitraum ab dem 15.06 erlaubt einen ganzen Monat effektiv nutzbarer Zeit zur Analyse und Verfassung des Endberichts.

Legende: *optionale Aufgabe*, **zwingende Aufgabe**

C. Benutzerhandbuch

Das Benutzerhandbuch ist in englischer Sprache verfasst worden, um das Programm für alle Interessenten zugänglich und verständlich zu machen.

ChipSimulator User Guide and Manual

*TU Dortmund
Faculty of Computer Science, Department VII
Institute of Machining Technology
Projectgroup 562, 2011/2012*

*Members:
Jim Bergmann, Jan Bessai, Andre Droschinsky,
Oliver Jungeilges, Armin Kazmi, Daniel Neugebauer,
Ercan Özdemir, Andrew Quinn, Christoph Schikora,
Christian Stossno, Stefan Voss*

This is the User Guide and Manual for **ChipSimulator**, developed by Projectgroup562: ChipSim, which took place in 2011/2012 at the TU Dortmund. It's purpose is to simulate chip formation during milling processes. For further information, see the final report (in German).

Contents

1	Introduction	2
2	Simulation Methods	2
2.1	SPH	2
2.2	MCA	2
2.3	Simulation Scene Types	2
3	Controls	3
3.1	Simulation Control	3
3.2	Visualization	4
3.3	Export	4
4	Project files	5
4.1	Parameter Types	5
4.2	Necessary parameters for all simulations	6
4.3	Parameters for Block-Type scenes	6
4.4	Parameters for Cutting-Type scenes	7
4.5	MCA Simulation Parameters	8
4.6	SPH Simulation Parameters	10
5	Batches	12

1 Introduction

This document serves solely as a technical manual for working with the program `ChipSimulator`. It is not meant to give any detailed background information on any of the methods used. For further information please see the final report [1].

For a brief introduction to the available simulation methods, see section 2. The controls are described in section 3. Instructions for creating a project file are given in section 4. An explanation of other miscellaneous functionality can be found in section 5.

2 Simulation Methods

`ChipSimulator` comes with two available particle-based simulation methods. In contrast to finite element simulations, particle based simulations do not make use of meshes to define scene geometry, thus eliminating the need for costly remeshing operations.

For further information on the methods below, see the provided references or the final report [1].

2.1 SPH

→ See also [2].

SPH, or *Smoothed Particle Hydrnamics*, is a simulation method which originates from the field of computational fluid dynamics. It was intially used in the field of astrophysics to simulate liquids and gases. Since then, the method has been extended to incorporate physical behavior such as viscosity, elasticity and plasticity, thus making it suitable for solid body simulations.

The central idea of SPH is to simulate materials by approximating a continuum with a finite set of particles. These sets of particles are not meant to serve as representations of atoms or molecules, but rather as sampling points over which the behavior of an object is interpolated.

2.2 MCA

→ See also [3].

In the MCA, or *Movable Cellular Automata*, method, objects are similarly sampled into sets of points which are referred to as automata, over which object behavior is interpolated. In contrast to the SPH method, however, there exists a state parameter for every pair of points indicating whether they are contacted, linked or unlinked. These states determine whether and to what extent these automata can influence each other.

2.3 Simulation Scene Types

The program provides three different types of simulated scenes: `BLOCK`, `CUTTING` and `BOUNDARY`.

The `BLOCK`-Type scene, in which a cube falls to the floor, was used to evaluate and tune the basic plastoelastic behavior of the simulation methods. This scene type additionally served to intially test the connection of particles.

A `CUTTING`-Type scene consists of a workpiece and a tool. The tool cuts through the work piece and should, depending on the parameters, initiate chip formation.

A deprecated but still included scene type is the **BOUNDARY**-type scene, where the permeability of a tool was tested. Additionally, a **BOUNDARY**-type scene provides the functionality to perform a test in which a bar of an arbitrary material rests upon two immovable anchors so that the material's plastoelastic properties may be observed.

3 Controls

This section describes the controls provided by the user interface. Below is a screenshot of the program's user interface.

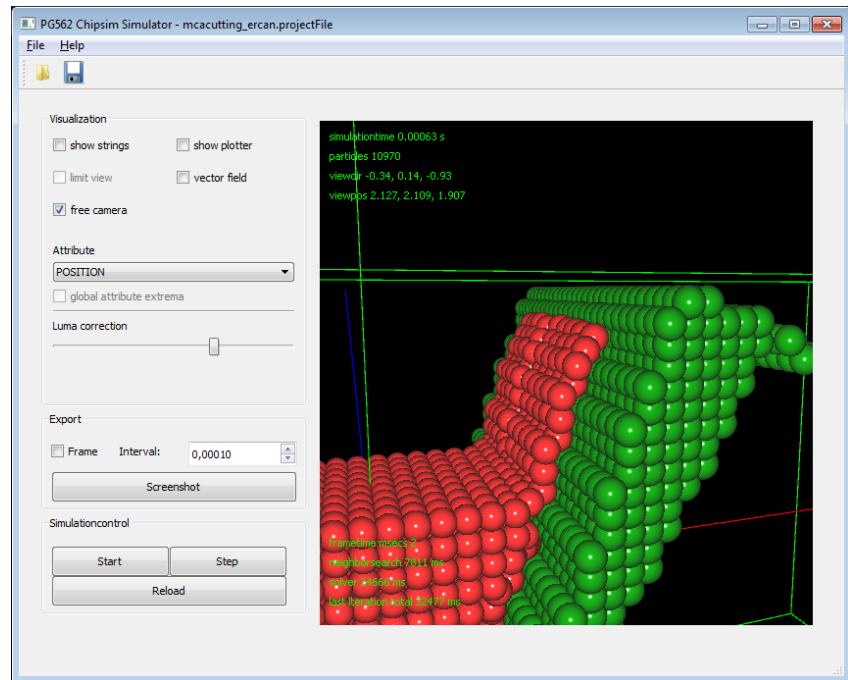


Figure 1: User Interface with sample simulation.

3.1 Simulation Control

The simulation controls (see figure 2) can be found in the lower left corner of the user interface. With these three buttons, the user is able to start and stop the simulation, simulate a single step and reset the entire simulation by reloading the project file.

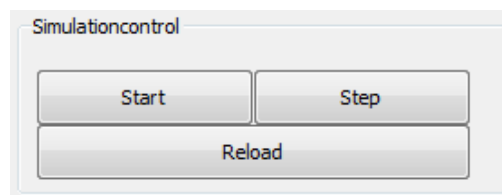


Figure 2: Simulationcontrol

! → The *Step* button executes a single iteration through the simulator's methods for solving for particle attributes and integrating these into the current scene. This corresponds to moving forward in time by an amount of `stepTime`, which is defined in the project file, see section 4.2.

3.2 Visualization

The upper left corner contains the visualization controls (see figure 3). By checking or unchecking *free camera*, the user is able to choose between having the camera's focal point bound to the center of the scene or allowing the user to move the camera freely through space, thus making it possible to focus on arbitrary parts of the scene.

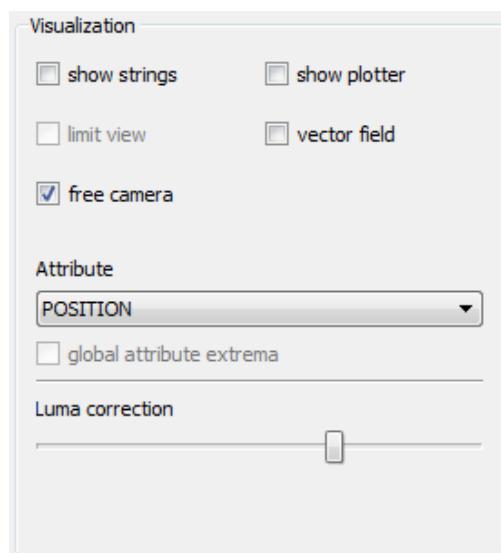


Figure 3: Visualizationcontrol

For better simulation analysis, a plotter can be activated by checking *show plotter*. The plotter will then appear above the simulation area, allowing the visualization of force trends which arise during the simulation.

The set of points can be displayed as spheres (default) or as a vector field by checking the *vector field* box. Additionally, the spheres (and vector fields) can be colored according to their properties by changing the selection in the *Attribute* combo-box. The colorization corresponds either to local minima/maxima or, when activated by the user, a global minima/maxima spectrum.

The *Luma correction* slider at the bottom provides gamma-correction, which is often useful when working in poor lighting or with overhead projectors.

3.3 Export

The export control can be found between the visualization and simulation controls.

By checking *Frames*, the program will periodically export a frame after the given interval, which is specified in the spin-box. These frames can then be used to generate a video of the simulated scene. The default export path is set to **build/export**.

The *Screenshot*-Button saves a single screenshot of the present simulation.

The *Savedisk*-Button on the upper toolbar creates a snapshot of the entire scene, including current positions, forces and interparticle connections.

A snapshot can be loaded into the program and the simulation can be resumed at any time.

4 Project files

Project files are needed to specify the simulation method that should be used, and, depending on the simulation type, provide the characteristics of the needed components. A sample of a project file can be seen below.

```
{
  "sceneType"      : "SPH_BLOCK",
  "blockDimensions" : [0.4, 0.4, 0.4],
  "blockPosition"  : [0.0, 0.017, 0.0],
  "blockMaterial"  : "Metal",
  "floorDimensions" : [0.1, 0.01, 0.1],
  "floorMaterial"  : "None",
  "stepTime"       : 1e-4,
  "smoothingLength" : 0.038077880,
  ...
}
```

A project file should provide all necessary parameters for the corresponding simulation (see below). The additional parameters affect the characteristics of the scene's components, as well as the physical environment. Additionally, the project file can provide world states of a simulation, including positions, forces and interparticle connections, when using the *Save*-functionality described in Section 3.3. Manually determining the world states of a simulation manually is not intended and will therefore not be discussed in this section. The program does not include default parameters for any parameters.

! →

4.1 Parameter Types

There are two types of parameters used when creating project files. First, there are simple parameters, which consist of a keyword and a value. The

```
{
  "exampleNumber" : 1e-4,
  "exampleString" : "sampleString",
  "exampleArray"  : [0.1, 0.2, 0.3],
}
```

Figure 4: Example of simple parameters

keyword is an identifier to determine which property should be set to the given value. The value can either be a floating point number, a string or an array, depending on the parameter (see listing 4).

The second parameter type is a parameter table, which consists of an array of objects.

Listing 5 is an example of a parameter table. An object is enclosed in braces and consists of a set of keys and their respective values.

```

{
  "exampleTable" : [
    { "key1-1" : "value1-1", "key1-2" : "value1-2" },
    { "key2-1" : "value2-1", "key2-2" : "value2-2" }
  ]
}

```

Figure 5: Example of a parameter table

4.2 Necessary parameters for all simulations

These two parameters should always be included in a project file. All other parameters default to zero.

<p>→ Section 2.1 for more information on SPH-Scenes</p>	<p>sceneType : defines the type of scene to be created when loading the project file. Possible types are:</p> <p>SPH_BLOCK SPH_CUTTING SPH_BOUNDARY (Deprecated)</p>
<p>→ Section 2.2 for more information on MCA-Scenes</p>	<p>MCA_BLOCK MCA_CUTTING MCA_BOUNDARY (Deprecated)</p>

See sections 4.3 and 4.4 for further information on scene-specific parameters and sections 4.5 and 4.6 for method-specific parameters.

stepTime : the interval between each iteration in *seconds* as a **floating point number**. A smaller **stepTime** results in more precise results during simulation.

! → Caution should be used when simulating with a very large **stepTime**, as objects may become unstable due to a lack of precision.

4.3 Parameters for Block-Type scenes

A short explanation on Block-Type Scenes can be found in section 2.3.

! → All sizes and positions are to be specified in *metres*.

blockDimensions : defines the height, width and depth of the block. Sizes are specified as [x, y, z].

blockPosition : the position of the center of the block. Position is specified as [x, y, z].

blockMaterial : the material of the block. Since there are no default values, always specify every needed parameter for each of the used materials.

floorDimensions : the dimensions of the floor, where x and z dimensions should at least be as big as the block's width and depth. Dimensions are given as [x, y, z].

floorMaterial : the material of the floor, analogous to **blockMaterial**.

sampleDistanceTable : a parameter table which contains material-specific particle sampling intervals. The value defines the distance in *metres* between orthogonal neighboring particles within the material. Example:

```

"sampleDistanceTable" : [
  { "material" : "mat1", "value" : 0.123 },
  { "material" : "mat2", "value" : 1.22e-1 }
],

```

wedgePosition : analogous to **blockPosition**, see section 4.3

wedgeDimensions : analogous to **blockDimensions**, see section 4.3

wedgeAngle : defines the wedges angle in degree.

wedgeMaterial : defines the wedges material, analogous to **blockMaterial**, see section 4.3

obstacleBlocksCenter : analogous to **blockPosition**, see section 4.3

obstacleBlocksDistance : determines the separation distance between obstacle blocks.

obstacleBlocksDimensions : analogous to **blockDimensions**, see section 4.3

obstacleBlocksMaterial : analogous to **blockMaterial**, see section 4.3

useFixedVelocity : determines whether blocks are moving with a fixed velocity at the beginning of the simulation.

fixedVelocity : defines a fixed velocity for the block, given in *metres/second*.

4.4 Parameters for Cutting-Type scenes

A short explanation of Cutting-Type Scenes can be found in section 2.3.

! → All sizes and positions are to be specified in *metres*.

workpieceDimensions : analogous to **blockDimensions**, see section 4.3

workpieceRippleFrequency : makes it possible to create a workpiece with a rippled surface. For non-rippled surfaces, ignore this parameter and set **workpieceRippleAmplitude** to zero.

workpieceRippleAmplitude : defines the height and depth of the ripples of a rippled surface in *metres*.

workpieceMaterial : analogous to **blockMaterial**

fixedLayerHeight : defines the height of the fixed layers in the workpiece. Fixed layers are used to fasten the workpiece to its original location, only allowing the upper region mobility.

fixedLayerRippleFrequency : (deprecated) analogous to **workpieceRippleFrequency**

fixedLayerRippleAmplitude : (deprecated) analogous to **workpieceRippleAmplitude**

toolDimensions : defines the height, width and depth of the tool. Sizes are specified as [x, y, z].

cuttingEdgeCenter : the position of the tool's edge, measured by its center. Position is specified as [x, y, z].

cuttingAngle : defines the tool's cutting angle in degree as a **floating point number**. See image 6 for a visualization of the cutting angle.

wedgeAngle : defines the tool's wedge angle in degree as a **floating point number**. See image 6 for a visualization of the wedge angle.

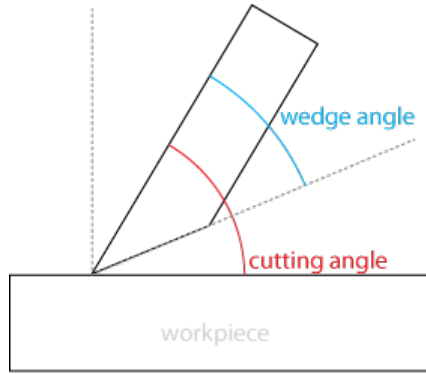


Figure 6: Tool's cutting angle (red) and wedge angle (blue)

- feedRate** : specifies the direction and speed for the tool's movement. Specified as $[x, y, z]$, where x, y, z are to be given in *metres/second*.
- toolMaterial** : analogous to **blockMaterial**
- sampleDistanceTable** : a parameter table, which contains the used distances for sampling a material with particles. The value defines the distance in *metres* between orthogonal neighboring particles within the material. Example:

```
"sampleDistanceTable" : [
  { "material" : "mat1", "value" : 0.123 },
  { "material" : "mat2", "value" : 1.22e-1 }
],
```

4.5 MCA Simulation Parameters

- densityTable** : a parameter table which contains the densities in *kilogram · metres⁻³* of various materials. Example:
- ```
"sampleDensityTable" : [
 { "material" : "someMaterial", "value" : 1e6 }
]
```
- gravityTable** : a parameter table, which contains the gravity in *metres/second<sup>2</sup>* affecting the material. Example for Earth's gravity affecting **someMaterial**:
- ```
"gravityTable" : [
  { "material" : "someMaterial", "value" : 9.81 }
]
```
- shearModulusTable** : a parameter table which contains the shear modulus in *pascal* for a specified material.
- ```
"shearModulusTable" : [
 {
 "material" : "Floor",
 "valueImp" : 350e-2,
 "value" : 79.3e9
 }
]
```
- ljpAlphaTable** : a parameter table which contains the Lennard-Jones-potential  $\alpha$ -value for two specified materials.

This is a scalar value which has no unit.

When specifying the `ljpAlphaTable` for two varying materials, you should define the value for `MaterialA`  $\rightarrow$  `MaterialB`, as well as for `MaterialB`  $\rightarrow$  `MaterialA`.

!  $\rightarrow$

These two values do not need to be the same.

```
"ljpAlphaTable" : [
 {
 "material1" : "Steel",
 "material2" : "Steel",
 "value" : 9.5e-200
 }
]
```

`frictionEtaTable` : a parameter table which contains the friction  $\eta$ -value for two specified materials.

This is a scalar value which has no unit.

Follows the same principle as `ljpAlphaTable`.

```
"frictionEtaTable" : [
 {
 "material1" : "Steel",
 "material2" : "Steel",
 "value" : 5.75e-8
 }
]
```

`yieldAngleTable` : a parameter table which contains the yield angle, in radians, for two specified materials.

Follows the same principle as `ljpAlphaTable`.

```
"yieldAngleTable" : [
 {
 "material1" : "Steel",
 "material2" : "Steel",
 "value" : 6.5e-10
 }
]
```

`contactedRadiusTable` : a parameter table which contains the contacted radius in *metres* for two specified materials.

Follows the same principle as `ljpAlphaTable`.

```
"contactedRadiusTable" : [
 {
 "material1" : "Steel",
 "material2" : "Steel",
 "value" : 0.0005
 }
]
```

`minLinkedRadiusTable` : a parameter table which contains the minimal linked radius in *metres* for two specified materials.

Follows the same principle as `ljpAlphaTable`.

```

"minLinkedRadiusTable" : [
 {
 "material1" : "Floor",
 "material2" : "Floor",
 "value" : 0.0003
 }
]

```

**maxLinkedRadiusTable** : a parameter table which contains the maximal linked radius in *metres* for two specified materials.

Follows the same principle as `ljpAlphaTable`.

```

"maxLinkedRadiusTable" : [
 {
 "material1" : "Floor",
 "material2" : "Floor",
 "value" : 0.03
 }
]

```

**dryFriction** : a friction parameter based on the normal force exerted by one cellular automaton onto another.

#### 4.6 SPH Simulation Parameters

The following parameters affect the SPH-Type Simulations. For a more detailed explanation on the parameters' effects, please see [1].

**smoothingLength** : the smoothing length defines the radius in *metres* in which neighboring particles are taken into account for force calculations. See figure 7 for a two dimensional illustration.

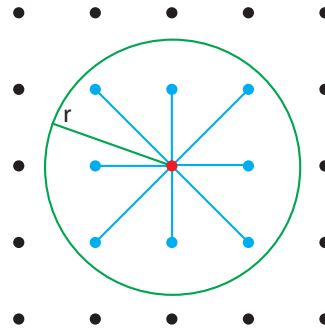


Figure 7: Smoothing Length (with radius  $r$ ) of a particle (red) with neighboring particles (blue)

! → The smoothing length should always be large enough to have a certain amount of particles in it's radius. A value of twice the sampling distance is recommended.

**densityTable** : a parameter table which contains the densities in *kilogram · metres<sup>-3</sup>* of various materials. Example:

```

"sampleDensityTable" : [
 { "material" : "someMaterial", "value" : 1e6 }
]

```

- gravityTable** : a parameter table which contains the gravity in *metres/s<sup>2</sup>* affecting various materials. Example for Earth's gravity affecting *someMaterial*:
- ```

    "gravityTable" : [
      { "material" : "someMaterial", "value" : 9.81 }
    ]

```
- gammaTable** : (deprecated)
- viscosityTable** : a parameter table which contains the viscosity attributes in *pascal*seconds* of various materials. Example:
- ```

 "viscosityTable" : [
 { "material" : "someMaterial", "value" : 100.0 }
]

```
- viscosityStiffnessTable** : a parameter table which contains the viscosity stiffness coefficient of the material. Example:
- ```

    "viscosityStiffnessTable" : [
      { "material" : "someMaterial", "value" : 100.0 }
    ]

```
- poissonRatioTable** : a parameter table which contains the Poisson's ratios of various materials. Example:
- ```

 "poissonRatioTable" : [
 { "material" : "someMaterial", "value" : 100.0 }
]

```
- youngsModulusTable** : a parameter table which contains the Young's moduli in *pascal* of various materials. Example:
- ```

    "youngsModulusTable" : [
      { "material" : "someMaterial", "value" : 100.0 }
    ]

```
- yieldGamma** : parameter specifies the portion of a material's sampling distance a particle may be forced out of its initial position before transitioning into a plastic deformation mode.
- creepGamma** : parameter specifies the threshold of particle displacement (relative to the material's sampling distance) above which the material fails and interparticle bonds are broken.

5 Batches

`ChipSimulator` comes with a Ant-based batch script to create and simulate batches of configurations. For the Ant script to run properly, the Java-Development-Kit¹ (JDK) must be installed on your system.

To configure the script, copy the `batchRunner.properties`, rename it to `privateBatchRunner.properties` and configure the property-file according to the included comments. To generate batches of project files, use the parameters `key`, `searchForValue`, `valueList` and `valueDelimiter` to replace certain keys and produce project files with replaced values.

Example:

To generate project files where the key `smoothingLength` with initial value (`searchForValue`) 0.034422 is replaced with a list of new parameters (e.g. `valueList: 0.03,0.04,0.05`), see the example provided below.

```
key = smoothingLength
searchForValue = 0.034422
valueList = 0.03, 0.04, 0.05
valueDelimiter = ,
```

Figure 8: Example-snippet of `privateBatchRunner.properties`

Using the command `ant generate`, the ant-script generates 3 project files containing varying `smoothingLengths` with values 0.03,0.04 and 0.05. The generated project files should include the parameter `maxSimulationTime` and `screenshotIntervall` to generate screenshots and end the simulation at a certain time.

! → The `valueDelimiter` only needs to be changed if the delimiter is contained in the `valueList`.

Finally, to simulate all batches from `/testdata/generated`, use the command `ant runThem`. You can then use the (optional) screenshots to create videos for each simulation.

¹ For the JDK see the official Oracle-Site:

<http://www.oracle.com/us/technologies/java/overview/index.html>

References

- [1] J. Bergmann and J. Bessai and A. Droschinsky and O. Jungeilges and A. Kazmi and D. Neugebauer and E. Özdemir and A. Quinn and C. Schikora and C. Stossno and S. Voss, Final report: Modellierung und Simulation plastoelastischer Objektinteraktion, Dortmund (2012)
- [2] M. F. Villumsen and T. G. Fauerholdt: Simulation of Metal Cutting using Smooth Particle Hydrodynamics. LS-Dyna Anwenderforum, Bamberg (2008)
- [3] S. G. Psakhie and Y. Horie and S. Y. Korostelev and A. Y. Smolin and A. I. Dmitriev and E. V. Shilko and S. V. Alekseev: Method of Movable Cellular Automata as a Tool for Simulation with the Framework of Mesomechanics. In: Russian Physics Journal 38 (1995)

Index

MCA-Parameter

- contactedRadiusTable, 9
- densityTable, 8
- dryFriction, 10
- frictionEtaTable, 9
- gravityTable, 8
- ljpAlphaTable, 8
- maxLinkedRadiusTable, 10
- minLinkedRadiusTable, 9
- shearModulusTable, 8
- yieldAngleTable, 9

- viscosityTable, 11
- yieldGamma, 11
- youngsModulusTable, 11

Parameter

- blockDimensions, 6
- blockMaterial, 6
- blockPosition, 6
- cuttingAngle, 7
- cuttingEdgeCenter, 7
- feedRate, 8
- fixedLayerHeight, 7
- fixedLayerRippleAmplitude, 7
- fixedLayerRippleFrequency, 7
- fixedVelocity, 7
- floorDimensions, 6
- floorMaterial, 6
- obstacleBlocksCenter, 7
- obstacleBlocksDimensions, 7
- obstacleBlocksDistance, 7
- obstacleBlocksMaterial, 7
- sampleDistanceTable (Block), 7
- sampleDistanceTable (Cutting), 8
- sceneTypes, 6
- stepTime, 6
- toolDimensions, 7
- toolMaterial, 8
- useFixedVelocity, 7
- wedgeAngle, 7
- wedgeDimensions, 7
- wedgeMaterial, 7
- wedgePosition, 7
- workpieceDimensions, 7
- workpieceMaterial, 7
- workpieceRippleAmplitude, 7
- workpieceRippleFrequency, 7

Projectfile Sample, 5

SPH-Parameter

- creepGamma, 11
- densityTable, 10
- gammaTable, 11
- gravityTable, 11
- poissonRatioTable, 11
- smoothingLength, 10
- viscosityStiffnessTable, 11