TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR TECHNISCHE INFORMATIK
PROFESSUR FÜR COMPUTERGRAPHIK UND VISUALISIERUNG
PROF. DR. STEFAN GUMHOLD

Bachelor-Arbeit

zur Erlangung des akademischen Grades
Bachelor of Science

Implementierung einer modularen Molakulardynamiksimulation mit Echtzeitvisualisierung

René Kaesler
(Geboren am 15. November 1989 in Dresden)

Betreuer: Dr. Sebastian Grottel

Dresden, September 09, 2013

Aufgabenstellung

Die Molekulardynamik ist eine partikelbasierte Simulationsmethode für Vielteilchensysteme. Atome, bzw. einfache starre Moleküle, werden als Partikel verwaltet. In einem Simulationszeitschritt werden die zwischen allen Paaren von Partikel wirkenden Kräfte mittels einfacher Modellen und gegebenen Funktionen berechnet. Aus den Kräften ergeben sich die Beschleunigungen, die Geschwindigkeiten und schließlich die Bewegungsbahnen der Partikel nach den einfachen Kraftgesetzen nach Newton. Moderne Simulationsprogramme basieren zwar auf diesen einfachen Regeln, nutzen aber sehr komplexe Algorithmen und Datenstrukturen um die Laufzeitkomplexität von $O(n^2)$ zu minimieren und möglichst schnelle und numerisch stabile Berechnungen durchführen zu können. Dies erlaubt zwar die Simulation großer Datenmengen bei hoher Geschwindigkeit, führt jedoch zu schwierig nachvollziehbaren Implementierungen.

Ziel dieser Arbeit ist es ein Molekulardynamiksimulationsprogram zu schreiben, welches möglichst einfach strukturiert ist. Die Verständlichkeit und Lesbarkeit des Codes und einfache Algorithmen sind hierbei wichtiger als die Ausführungsgeschwindigkeit. Ein sauberer Entwurf und eine saubere Implementierung in C++ sind das Ziel. Durch sinnvoll definierte Klassen und Interfaces sollte es möglich sein, einfache Optimierungen wahlweise einzubringen (z.B. Beschleunigungsstrukturen für Nachbarschaftssuche oder optimierte Methoden wie das Leapfrog-Verfahren). Allerdings wird sich die Simulation der Einfachheit halber ausdrücklich auf die lokale Ausführung auf einem einzelnen Computer beschränken.

Zur Anzeige der entstehenden Simulationsdaten steht ein Visualisierungsprogram am Lehrstuhl zur Verfügung. In ersten Schritten sollen die Daten in eine Datei geschrieben und anschließend visualisiert werden (klassisches Simulations-Post-Processing). Später soll die Simulation direkt an die Visualisierung gekoppelt werden, um die Daten in Echtzeit betrachten zu können. Diese Kopplung erfolgt indem der Simulationscode in die Visualisierungssoftware eingebracht wird und beide Aufgaben in einem gemeinsamen Prozess abgearbeitet werden.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Bachelor-Arbeit zum Thema:

Implementierung einer modularen Molakulardynamiksimulation mit Echtzeitvisualisierung

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den September 10, 2013

René Kaesler

Kurzfassung

Durch die stetig steigende Rechenleistung sind Molekulardynamiksimulationen aus der Forschnung nicht mehr wegzudenken. Allerdings fällt es Neueinsteigern oft schwer, sich mit den Methoden der Molekuladynamik auseinanderzusetzen, da der verfügbare Quellcode der meisten Simulatoren sehr unübersichtlich ist. In dieser Arbeit werden Grundlagen und Entwurf einer einfachen und modularen Molekulardynamiksimulation beschrieben. Die daraus entstandene Simulation mit Echtzeitvisualisierung soll Neueinsteigern dabei helfen, Ablauf und Funktionsweise von Molekulardynamiksimulationen zu verstehen.

Abstract

Due to steadily growing computing power, it is difficult to imagine scientific research without molecular dynamics simulators. Nevertheless, for beginners it often is a challenge to deal with the methods of the molecular dynamic. This is caused by the fact, that the available source codes of most simulators are rather unorganized. This work explains the basis and the model of a simple and modular molecular dynamics simulator. The resultung simulation with real-time vizualisation aims to help beginners to understand the process and the function of such molecular dynamic simulators.

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Problemstellung . . . 3
1.2 Zielsetzung der Arbeit . . . 4

2 Grundlagen der Molekulardynamik **5**
2.1 Ablauf einer MD-Simulation . . . 5
2.2 Zwischenmolekulare Wechselwirkungen . . . 6
2.2.1 Lennard-Jones-Paarpotential . . . 7
2.2.2 Gravitationspotential . . . 8
2.3 Lösung der Bewegungsgleichung . . . 9
2.4 Integrationsverfahren . . . 9
2.4.1 Euler-Methode . . . 10
2.4.2 Leapfrog-Methode . . . 10
2.4.3 Velocity-Verlet-Methode . . . 10
2.5 Wichtige Systemgrößen . . . 11
2.5.1 Energie . . . 11
2.5.2 Temperatur . . . 12
2.5.3 Druck . . . 12
2.6 Periodische Randbedingungen . . . 12
2.6.1 Minimum Image Convention . . . 13
2.7 Nachbarschaftslisten . . . 14
2.8 Ensembles . . . 15
2.9 Numerische Probleme . . . 15

3 Entwurf einer Modularen MD-Simulation **17**
3.1 Vorüberlegung . . . 17
3.2 Modulentwurf . . . 18
3.2.1 Abhängigkeiten . . . 19
3.2.2 Objektorientierter Ansatz . . . 19
3.3 Das Simulator-Modul . . . 22
3.3.1 Die run()-Methode . . . 22
3.3.2 Pre-, Immediate- und Postprocessor . . . 23
3.4 Das Box-Modul . . . 23
3.4.1 Particle-Wrapper . . . 24
3.4.2 Boxiterator und Particleiterator . . . 24
3.5 Das Interactor-Modul . . . 25

3.6 Das Integrator-Modul 26
3.7 Das Initializer-Modul 26
3.8 Der Render-Adapter 27

4 Evaluation **29**
4.1 Verfügbare Module 29
4.1.1 Initializer-Module 29
4.1.2 Interactor-Module 30
4.1.3 Integrator-Module 30
4.1.4 Box-Module 31
4.2 Der Lennard-Jones-Simulator 32
4.3 Der Planet-Simulator 32
4.4 Datenausgabe 33
4.5 Datenvisualisierung 34
4.6 Ausblick 35

Literaturverzeichnis **37**

Abbildungsverzeichnis **39**

1 Einleitung

Im Bereich der Naturwissenschaften sind Computersimulationen nicht mehr wegzudenken, bieten sie doch die Möglichkeit, Verhalten und Eigenschaften vieler Systeme auf rein virtueller Basis zu untersuchen. Eine Methode zur Untersuchung von Vielteilchensystemen ist die sogenannte Molekulardynamik (MD). Mit ihr lassen sich einfache Stoffe für einen kurzen Zeitraum simulieren und strukturelle, dynamische sowie auch thermodynamische Stoffeigenschaften bestimmen. Atome bzw. Moleküle werden in der Molekulardynamik als kleine Teilchen (Kugeln) angesehen, die den Gesetzen der Newtonschen Mechanik unterliegen und sich frei im Raum bewegen. Dabei wird ihre Bewegungsbahn (Trajektorie) von anderen Teilchen im unmittelbaren Umfeld beeinflusst. Der Grund dafür sind anziehende und abstoßende Kräfte zwischen den Teilchen. Die Aufgabe einer klassischen MD-Simulation ist es, die einzelnen Trajektorien der Teilchen innerhalb eines Systems für einen vorgegebenen Zeitraum zu bestimmen.

1.1 Problemstellung

Ist man auf der Suche nach MD-Simulationsprogrammen, wird man schnell fündig. So ist in fast jeder Fachliteratur der Quellcode für eine voll funktionsfähige Implementierung einer MD-Simulation beigelegt. Als Beispiel sei das Buch von PROF. DR. REINHOLD HABERLANDT *Molekulardynamik - Grundlagen und Anwendungen* [Hab95] erwähnt. Auch Im Internet finden sich unzählige Programme, die frei zum Download bereit stehen [Beu]. Die meisten dieser Programme sind in *Fortran* - einer imperativen Programmiersprache - geschrieben. Die maschinennahe Problembeschreibung ist zwar Vorteilhaft, wenn es um die Frage der Performance geht, jedoch schränkt sie sowohl Lesbarkeit als auch Erweiterbarkeit eines Programms stark ein. Ein Neueinsteiger hat es dadurch nicht gerade einfach, so ein Programm zu verstehen, sind doch die verwendeten Algorithmen und Datenstrukturen sehr komplex. Möchte er dann noch Teile des Programms umschreiben, um seine bereits erworbenen Kenntnisse im Gebiet der Molekulardynamik zu überprüfen, so ist sein Misserfolg regelrecht vorprogrammiert. Zwar existieren bereits objektorientierte MD-Simulationen, wie z.B. *NAMD* [NAM], die eine gewisse Modularität und Übersichtlichkeit in den Quellcode bringen, doch liegt auch ihr Augenmerk auf Performance und Geschwindigkeit der Simulation.

Die meisten MD-Simulationen besitzen keine Möglichkeit zur Echtzeitvisualisierung. Möchte man die Trajektorien der Teilchen beobachten, so müssen sie nachträglich mit einen Visualisierungsprogramm, wie z.B *VMD* [VMD], animiert werden. Neueinsteiger animieren ihre Simulationsdaten sehr oft, um herauszufinden, welche Auswirkungen Veränderungen im Programmcode auf die Simulation haben. Die nachträgliche Visualisierung ist dabei oft lästig und unkomfortabel. Einer Echtzeitvisualisierung würde den Aufwand für die nachträgliche Visualisierung vermeiden und das Experimentieren mit dem Simulationscode wesentlich angenehmer gestalten.

1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist es, dem Neueinsteiger eine Möglichkeit zu bieten, die Grundlagen der Molekulardynamik anhand einer voll funktionsfähigen MD-Simulation zu erlernen. Für diesen Zweck ist es notwendig, eine typische MD-Simulation in kleine Module aufzuteilen. Wie in einem Baukasten lassen sich die einzelnen Module nachher zu einem MD-Simulator zusammenstecken. Dabei hat der Nutzer die Möglichkeit eigene Module zu implementieren und diese anstelle vorgegebener Module für einen Simulator zu verwenden. Die in der Arbeit implementierten Module sollen zu einer C++ Bibliothek, genannt *MD-Sim*, zusammengefasst werden. Außerdem sollen bereits zwei Simulatoren als Beispiel in der Bibliothek zur Verfügung stehen. Wichtig hierbei ist, dass Simulatoren als eigenständige Programme zu sehen sind. Es ist also wünschenswert, Ausgabe und Visualisierung von der eigentlichen Simulation zu trennen, um eine individuelle Verwendung eines Simulators zu garantieren. Allein der Nutzer entscheidet, welche Simulationsdaten ausgegeben werden und wie bzw. ob sie überhaupt visualisiert werden sollen. Dem Nutzer sollen standardmäßig zwei Möglichkeiten für die Datenvisualisierung zur Verfügung stehen:

- **Visualisierung durch Post-Processing**
  Die für jeden Zeitschritt berechneten Positionen der Teilchen werden in eine Datei geschrieben und anschließend mit einem Visualisierungsprogramm animiert.
- **Echtzeitvisualisierung**
  Nach jedem Zeitschritt werden die Positionen der Teilchen an ein Visualisierungsprogramm geschickt und direkt auf den Bildschirm gerendert.

Für die Echtzeitvisualisierung der Teilchenbewegungen steht am Lehrstuhl das Visualisierungsprogramm *MegaMol* [Meg] zur Verfügung. Im Rahmen der Arbeit soll ein Plugin geschrieben werden, dass die Anbindung eines Simulators an MegaMol ermöglicht. Die Verwendung von MegaMol ist allerdings nicht trivial. Aus diesem Grund wird ein zweites Visualisierungsprogramm namens *IrrMol* implementiert, welches zwar nicht so hohe Frameraten wie MegaMol erzeugt, jedoch in der Simulationsanbindung einfacher zu handhaben ist. IrrMol ist nicht Teil der Aufgabenstellung und dient daher nur als persönliches Hilfswerkzeug für die Modulentwicklung.

Bei der Implementierung ist allgemein darauf zu achten, dass der Quellcode einfach strukturiert und übersichtlich ist. Dennoch soll die Performance der Simulation nicht komplett außer acht gelassen werden. So soll die Möglichkeit der Parallelisierung einzelner Module definitiv bestehen.

2 Grundlagen der Molekulardynamik

Für das allgemeine Verständnis ist es notwendig, sich näher mit den Grundlagen der Molekulardynamik zu befassen. Allerdings ist es nicht möglich auf alle Details genauer einzugehen, da dies den Rahmen der Arbeit sprengen würde. Daher sei auf *Molekulardynamik - Grundlagen und Anwendungen* [Hab95] hingewiesen, welches die hier angesprochenen Themen und Gleichungen ausführlich erklärt.

2.1 Ablauf einer MD-Simulation

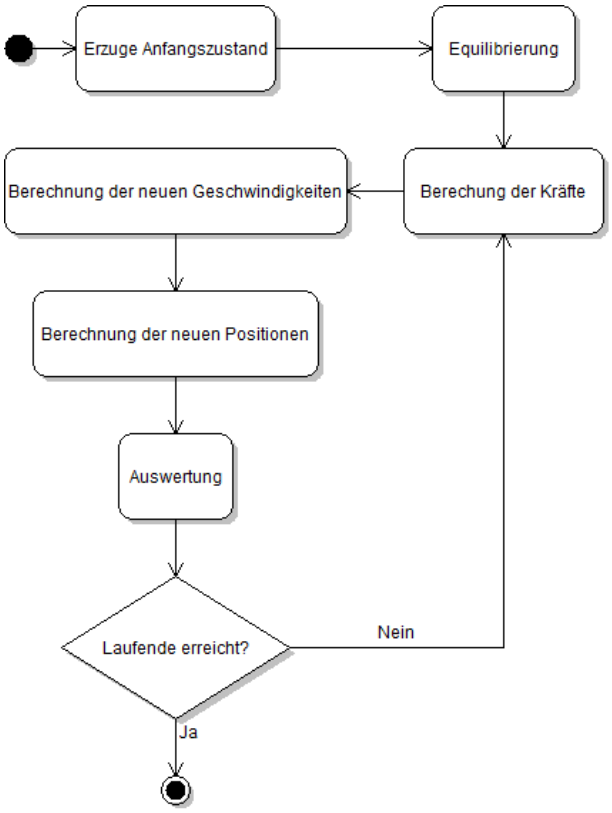

Abbildung 2.1: Prinzipieller Ablaufplan einer MD-Simulation

Zu Beginn ist es nötig, für das zu simulierende Vielteichensystem einen Anfangszustand zu herzustellen (Abb. 2.1). In MD-Simulationen ordnet man alle Teilchen $i \in \{1, \ldots, N\}$ innerhalb einer zuvor definierten Box (MD-Box) an. Ihr Volumen hängt dabei vom Druck ab, der im System herrschen soll (siehe 2.5.3). Es ist wünschenswert, dass alle Teilchen einen größtmöglichen Abstand zueinander besitzen. Liegen zwei Teilchen zu dicht aneinander, so ist der Betrag ihrer Anfangsenergie unnatürlich hoch. Das würde das Simulationsergebnis stark verfälschen. Jedem Teilchen wird außerdem eine zufällige Startgeschwindigkeit $\dot{r}_i(start)$ zugewiesen. Allerdings befindet sich nun das System als Ganzes mit hoher

Wahrscheinlichkeit in Bewegung, ähnlich einem Fluss, in dem das Wasser in eine bestimmte Richtung fließt. In MD-Simulationen ist solch eine Bewegung nicht erwünscht. Im allgemein sollte sich das Gesamtsystem in Ruhe befinden. Dies ist der Fall, wenn die Schwerpunktgeschwindigkeit v_s des Systems gleich null ist [Kie13].

$$v_s = \frac{\sum_{i=1}^{n} m_i \dot{r}_i}{\sum_{i=1}^{n} m_i} = 0 \tag{2.1}$$

Für die Erzeugung der Startgeschwindigkeiten muss demnach für jedes Teilchen die aktuelle Schwerpunktgeschwindigkeit wieder abgezogen werden, um das Gesamtsystem in Ruhe zu bringen:

$$\dot{r}_i(start) = \dot{r}_i(start) - v_s \tag{2.2}$$

Ist der Anfangszustand hergestellt, muss das System noch ins Gleichgewicht gebracht werden, d.h. die Simulation muss noch warmlaufen. Diesen Vorgang bezeichnet man als Equilibrierung und ist im Grunde genommen ein mehrfacher Simulationsschleifendurchlauf, vor der eigentlichen Simulation.

Die Simulation beginnt mit dem Schleifendurchlauf für den ersten Zeitschritt t_1. Zuerst ist die Berechnung der Kräfte, die zwischen den Teilchen wirken, notwendig. Anhand dieser Kräfte lassen sich die neuen Geschwindigkeiten $\dot{r}_i(t_1)$ für jedes Teilchen bestimmen. Mit den Teilchengeschwindigkeiten lassen sich wiederum die neuen Teilchenpositionen $r_i(t_1)$ bestimmen. Nach der Ermittlung der neuen Geschwindigkeiten und Positionen befindet sich das System in einem völlig neuen Zustand. Deshalb findet am Ende der Simulationsschleife eine Auswertung statt, in der Systemgrößen wie Energie, Druck und Temperatur neu berechnet und analysiert werden. In den meisten Fällen definiert man für eine Simulation eine maximale Anzahl vom Zeitschritten t_{max}. Ist t_{max} erreicht, wird die Simulation beendet. Andernfalls beginnt der nächste Schleifendurchlauf für den zweiten Zeitschritt t_2 [Hab95].

2.2 Zwischenmolekulare Wechselwirkungen

Die Teilchen in einem System befinden sich in ständiger Bewegung. Sie besitzen also kinetische Energie. Die Teilchen beeinflussen sich auch gegenseitig. Elektromagnetische Kräfte sogen dafür, dass sich zwei Teilchen abstoßen, sobald sie zu nah aneinander geraten. Ist der Abstand r_{ij} zweier Teilchen i, j hingegen zu groß, so ziehen sie sich an. Grund dafür sind die sogenannten Van-Der-Waals-Kräfte, die zwischen den beiden Teilchen herrschen. Die Teilchen besitzen demnach, abhängig von ihrer Position im Raum, auch potentielle Energie. Den Vorgang der gegenseitigen Anziehung und Abstoßung zweier Teilchen bezeichnet man als zwischen-molekulare Wechselwirkung. Sie lässt sich mit Hilfe von Potentialen beschreiben. Dabei gibt ein Potential die potentielle Energie zweier Teilchen in Abhängigkeit ihres Abstandes voneinander an [Zie].

Potentiale spielen bei MD-Simulationen eine wichtige Rolle, ermöglichen sie doch die Bestimmung der Wechselwirkungskräfte F_{ij} und F_{ji}, zweier Teilchen. Die räumliche Ableitung des Potentials $\Phi(r_{ij})$ entspricht dabei F_{ij}.

$$F_{ij} = \frac{\partial \Phi(r_{ij})}{\partial r} \tag{2.3}$$

Gemäß dem dritten newtonschen Axiom *Actio est Reactio* (lat.: Aktion ist gleich Reaktion) ist der Betrag der Wechselwirkungskräfte zweier Teilchen identisch. Sie unterscheiden sich nur in ihren Vorzeichen [Zie].

$$F_{ij} = -F_{ji} \tag{2.4}$$

Ein Teilchen i wechselwirkt nicht nur mit einem, sondern mit allen Teilchen innerhalb des Systems (Abb. 2.2). Die Summe all seiner Paarwechselwirkungskräfte F_{ij} entspricht der Gesamtkraft F_i.

$$F_i = \sum_{i=i,j\neq i}^{N} F_{ij} \tag{2.5}$$

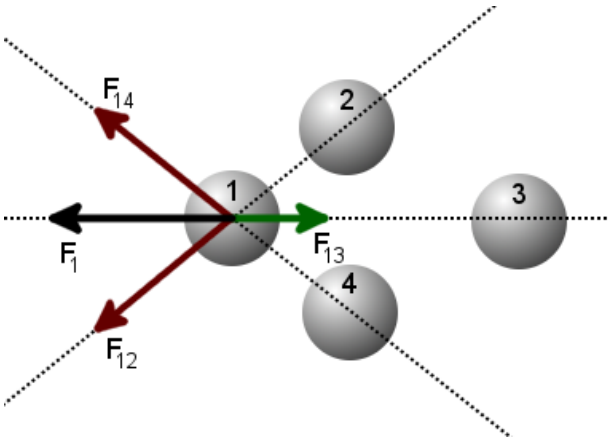

Abbildung 2.2: Wirkende Kräfte auf ein Teilchen

2.2.1 Lennard-Jones-Paarpotential

Das Lennard-Jones-Paarpotential modelliert die zwischen-molekularen Wechselwirkungen zwischen zwei Teilchen mit der selben Masse m. Es setzt sich aus den Termen $-\left(\frac{\sigma}{r_{ij}}\right)^6$ für die Anziehung und $\left(\frac{\sigma}{r_{ij}}\right)^{12}$ für die Abstoßung zweier Teichen zusammen [Zie].

$$\Phi(r_{ij}) = 4 \cdot \epsilon \cdot \left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right] \tag{2.6}$$

Entscheidend dafür, wie nahe sich zwei Teilchen kommen können, ist der Abstandsparameter σ . Er definiert die Nullstelle des Potentials (Abb. 2.3). Grundsätzlich kann man davon ausgehen, dass sich zwei

Teilchen nicht viel näher als σ kommen. Besitzen die Teichen allerdings große kinetische Energien, d.h ist der Betrag der Geschwindigkeit mit der sich die Teilchen aufeinander zubewegen groß genug, so kann der Abstand der Teilchen kleiner als σ werden. Aufgrund des extrem starken Anstiegs der Potentialfunktion ab dem Punkt r_{min} ist dies jedoch nur im geringen Maße möglich [Zie]. Der Energieparameter ϵ ist entscheidend für die Stärke der Wechselwirkungen zwischen zwei Teilchen. Er legt den Wendepunkt des Potentials fest und bestimmt somit die Größe der potentiellen Energie, die beide Teilchen bei einem Abstand von r_{ij} besitzen.

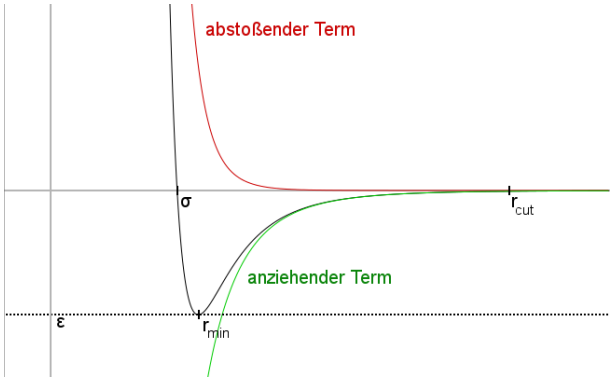

Abbildung 2.3: 2D-Plot des Lennard-Jones-Potentials. Die X-Achse entspricht dem Abstand und die Y-Achse der potentiellen Enegie beider Teilchen

Mit steigendem Teilchenabstand nähert sich die potentielle Energie immer mehr dem Nullpunkt der Y-Achse an. Sobald der Teilchenabstand die Größe von r_{cut} erreicht, sind die Wechselwikungen zwischen den beiden Teilchen so gering, dass sich eine Kraftberechung für die MD-Simulation nicht mehr lohnt. Deshalb wird das Lennard-Jones-Paarpotential ab einem Teilchenabstand von r_{cut} abgeschnitten. Die Wechselwirkungen zwischen Teilchen mit einem Abstand größer als r_{cut} werden ignoriert. Typischerweise wird das Lennard-Jones-Paarpotential bei $r_{cut} = 2.5\sigma$ abgeschnitten [Zie].

2.2.2 Gravitationspotential

Da MD-Simulationen auf den Gesetzen der Newtonschen Mechanik beruhen, könnte man mit ihnen auch Systeme, bestehend aus größeren Objekten, simulieren. Die Wechselwirkungen zwischen den jeweiligen Objekten müssten jedoch mit Hilfe des Gravitationspotentials beschrieben werden (Abb. 2.4). Anders als beim Lennard-Jones-Paarpotential existieren zwischen zwei Objekten, wie z.B. Planeten, keine Abstoßungskräfte. Allerdings können die Objekte unterschiedliche Massen m_1, m_2 besitzen. Die Massen der Objekte müssen also in die Potentialfunktion mit einbezogen werden.

$$\Phi(r_{ij}) = -g \cdot \frac{m_1 \cdot m_2}{r_{ij}} \tag{2.7}$$

Die Naturkonstante g heißt Gravitationskonstante. Es ist darauf zu achten, dass ab einen zu kleinem Abstand r_{ij} die Objekte miteinander kollidieren. Beim Gravitationspotential wird eine Kollision nicht mit berücksichtigt. Sie muss separat behandelt werden. Sobald die Objekte aufeinander stoßen, müssen

ihre Geschwindigkeiten neu bestimmt werden. Dies geschieht mit Hilfe des Impulserhaltungssatzes. Inwiefern sich die Geschwindigkeiten der Objekte verändern, hängt dabei von der verwendeten Stoßart (elastisch, unelastisch, real) ab [Dil].

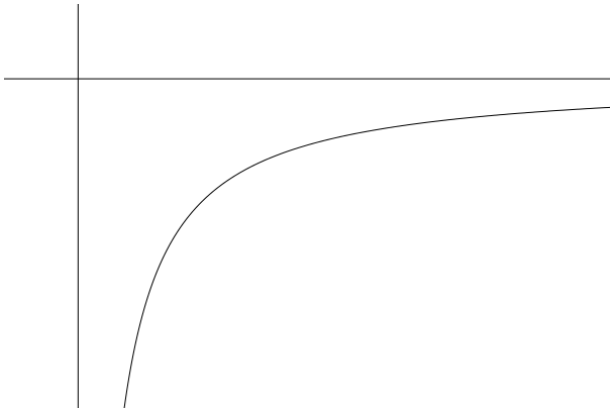

Abbildung 2.4: 2D-Plot des Gravitations-Potentials

2.3 Lösung der Bewegungsgleichung

Es sei $r_i \in \mathbb{R}^3, i \in \{1, \ldots, N\}$ der Positionsvektor der Teilchen. Um ein System von N Teilchen zu simulieren, müssen sowohl Teilchenpositionen r_i als auch Teilchengeschwindigkeiten $\dot{r}_i$ zu jedem Zeitpunkt t bestimmt werden. In der Molekulardynamik unterliegen die Teilchen den Gesetzen der Klassischen Mechanik. Mit Hilfe der Newtonschen Bewegungsgleichung lässt sich die Beschleunigung $\ddot{r}_i$ für das Teilchen i berechnen.

$$m_i \cdot \ddot{r}_i = F_i \qquad\qquad i \in \mathbb{N} \tag{2.8}$$

Geschwindigkeit und Beschleunigung entsprechen der ersten und zweiten zeitlichen Ableitung der Position.

$$\ddot{r}_i = \frac{\partial \dot{r}_i}{\partial t} \tag{2.9}$$

$$\dot{r}_i = \frac{\partial r_i}{\partial t} \tag{2.10}$$

Die Teilchengeschwindigkeiten und die Teilchenpositionen lassen sich durch zeitliche Integration der Geschwindigkeit herleiten [PCM].

2.4 Integrationsverfahren

Die Beschleunigung der Teilchen entspricht den zeitlichen Ableitungen der Geschwindigkeit und der Position. Um also Position und Geschwindigkeit für ein Teilchen zu ermitteln muss ihre Beschleunigung nach der Zeit t integriert werden. Da eine analytische Integration nicht ohne weiteres möglich ist, verwendet die Molekulardynamik Algorithmen der nummerischen Integration.

2.4.1 Euler-Methode

In MD-Simulationen sind die Anfangswerte für Position r und Geschwindigkeit $\dot{r}$ eines Teilchens zum Zeitpunkt $t = 0$ bereits bekannt. Auch die Beschleunigung $\ddot{r}$ ist durch die Position der Teilchen eindeutig bestimmt. Die explizite Euler-Methode betrachtet diese Anfangswerte und interpoliert sie konstant über die Zeitschritte Δt hinweg.

$$r(t + \Delta t) = r(t) + \dot{r}(t) \cdot \Delta t \tag{2.11}$$

$$\dot{r}(t + \Delta t) = \dot{r}(t) + \ddot{r}(t) \cdot \Delta t \tag{2.12}$$

Der Vorteil der expliziten Euler-Methode liegt in ihrer Einfachheit. Sie ist schnell zu implementieren und belegt nur wenig Speicher. Jedoch ist die Euler-Methode nicht sehr genau, da es um ein Verfahren erster Ordnung handelt. Alle Taylor-Terme ab einschließlich dem Δt^2-Term werden abgeschnitten. Das führt zu Approximationsfehlern, die das Ergebnis verfälschen [Kum10].

2.4.2 Leapfrog-Methode

Genau wie die explizite Euler-Methode ist der Leapfrog-Algorithmus eine einfache Möglichkeit für eine nummerische Integration in MD-Simulationen. Allerdings handelt es sich bei der Leapfrog-Methode um ein Verfahren zweiter Ordnung und liefert demnach bessere Ergebnisse als die Euler-Methode. Position r und Geschwindigkeit $\dot{r}$ werden dazu um einen Zeitschritt versetzt voneinander berechnet. Daher leitet sich auch der Name der Leapfrog-Methode (dt.: Bocksprung) ab.

$$r(t + \Delta t) = r(t) + \dot{r}\left(t + \frac{\Delta t}{2}\right) \cdot \Delta t \tag{2.13}$$

$$\dot{r}\left(t + \frac{\Delta t}{2}\right) = \dot{r}\left(t - \frac{\Delta t}{2}\right) + \ddot{r}(t) \cdot \Delta t \tag{2.14}$$

Eine besondere Eigenschaft der Leapfrog-Methode ist ihre Invarianz. Rechnet man also bis zu einem bestimmten Zeitpunkt zurück, so erhält man die selben Werte, die auch zuvor für den Zeitpunkt t ermittelt wurden. Die Leapfrog-Methode hat jedoch einen Nachteil, wenn es um die Berechnung der kinetischen Energie geht. Ist die Ermittlung der potentiellen und kinetischen Energie zum gleichen Zeitpunkt t erwünscht, so ist eine weitere Hilfsfunktion notwendig [Kum10].

$$\dot{r}(t) = \frac{\dot{r}\left(t - \frac{\Delta t}{2}\right) + \dot{r}\left(t + \frac{\Delta t}{2}\right)}{2} \tag{2.15}$$

2.4.3 Velocity-Verlet-Methode

Die Velocity-Verlet-Methode ist eine Erweiterung der Leapfrog-Methode. Im Grunde entsprechen beide Methoden dem selben Ansatz, Position und Geschwindigkeit zu unterschiedlichen Zeitpunkten zu ermitteln. Allerdings vermeidet die Velocity-Verlet-Methode den Nachteil, die bei der Leapfrog-Methode entsteht, wenn potentielle und kinetische Energie zur gleichen Zeit bestimmt werden sollen. Dafür wird

vor und nach der Positionsermittlung die Geschwindigkeit um einen halben Zeitschritt erhöht. Man erhält am Ende der Routine Geschwindigkeit $\dot{(r)}$ und Position r zum gleichen Zeitpunkt t.

$$\dot{r}\left(t+\frac{\Delta t}{2}\right) = \dot{r}(t) + \frac{1}{2}\cdot\ddot{r}(t)\cdot\Delta t \tag{2.16}$$

$$r(t+\Delta t) = r(t) + \dot{r}\left(t+\frac{\Delta t}{2}\right)\cdot\Delta t \tag{2.17}$$

$$\dot{r}(t+\Delta t) = \dot{r}\left(t+\frac{\Delta t}{2}\right) + \frac{1}{2}\cdot\ddot{r}(t+\Delta t)\cdot\Delta t \tag{2.18}$$

Die Velocity-Verlet-Methode ist die wohl meist verwendete nummerische Integrations-Methode. Sie überzeugt durch geringe Fehlerrate und die zeitliche Synchronität zwischen Position und Geschwindigkeit [Kum10].

2.5 Wichtige Systemgrößen

In der statistischen Physik wird das Verhalten eines Systems durch die Bewegung der einzelnen Teilchen erklärt. Wie sich ein Stoff verhält, hängt von seinen Eigenschaften ab. Diese Eigenschaften lassen sich mit Hilfe von makroskopischen Größen, wie z.B. Gesamtenergie, Temperatur und Druck beschreiben.

2.5.1 Energie

Jedes Teilchen besitzt sowohl potentielle als auch kinetische Energie. Die mittlere Gesamtenergie eines Systems lässt sich wie folgt bestimmen.

$$\overline{E} = \overline{E}_{kin} + \overline{E}_{pot} \tag{2.19}$$

Für die mittlere kinetische Energie pro Teilchen gilt

$$\overline{E}_{kin} = \frac{1}{2N}\sum_{i}^{N} m_i\dot{r}_i. \tag{2.20}$$

Die mittlere potentielle Energie ist

$$\overline{E}_{pot} = \frac{1}{N}\sum_{i,j>i}^{N}\Phi(r_{ij}). \tag{2.21}$$

Die Energieänderung ΔE in einem System ist gleich der Summe der Wärme ΔT, die einem System zugeführt wird, und der Arbeit ΔW, die am System verrichtet wird. Demnach lässt sich mit Hilfe der Systemenergie bestimmen, wie leicht sich ein Stoff erwärmen lässt und wie schnell er wieder abkühlt [PCM, Gol].

$$\Delta E = \Delta T + \Delta W \tag{2.22}$$

2.5.2 Temperatur

Ob ein Stoff im gasförmigen, flüssigen oder festen Zustand auftritt, hängt von der Temperatur ab. Für ein einatomig, reines, ideales Gas steht steht die Temperatur im direkten Zusammenhang mit der kinetischen Energie. Sie ist definiert über

$$T = \frac{2}{3k_B}\overline{E}_{kin}, \tag{2.23}$$

wobei $k_B = 1,308648 \cdot 10^{-}23 J/K$ die *Bolzmann*-Konstante bezeichnet. Mit Hilfe von MD-Simulationen können demnach sowohl Siede- als auch Schmelztemperatur für einfache Stoffe ermittelt werden [PCM].

2.5.3 Druck

Die Systemtemperatur ist abhängig vom Systemdruck. So siedet z.B. Wasser bei einem niedrigen Druck eher als bei einem höheren Druck. Eine Angabe der Siede- oder Schmelztemperatur hat daher keinen Aussagewert ohne die Angabe des verwendeten Systemdrucks. Die Berechnung des Druckes P erfolgt über den sogenannten Virialsatz. Dies ermöglicht eine einfache Bestimmung von P, ohne sich mit den Impulsänderungen innerhalb des Systems auseinander setzen zu müssen. Unter der Verwendung des Lennard-Jones-Paarpotentials gilt für P

$$PV = Nk_BT + \frac{1}{3}W, \tag{2.24}$$

wobei V das Volumen ist, dass die Teilchen einschließt. Das Virial W berücksichtigt die Wechselwirkungen der Teilchen untereinander [PCM].

$$W = -\frac{1}{N}\sum_{i,j>i}^{N} F_{ij} \cdot r_{ij} \tag{2.25}$$

2.6 Periodische Randbedingungen

MD-Simulationen sollen in der Regel einen Ausschnitt aus einer Flüssigkeit oder einem Gas simulieren. Wechselwirkungen mit den Wänden der MD-Box sollen grundsätzlich nicht in die Simulation mit einfließen. Deshalb führt man in der Molekulardynamik periodische Randbedingungen für die MD-Box ein. Die Idee dahinter ist, Abbilder der Box um die eigentliche MD-Box herum anzuordnen (Abb. 2.5). Ein Teilchen i wechselwirkt nun neben den anderen Teilchen j auch mit dessen Abbildungen. Da in allen Abbildern das Gleiche geschieht, ist die Berechnung von Geschwindigkeit und Position nur für die Teilchen innerhalb der eigentlichen MD-Box notwendig. Für die Teilchen innerhalb der MD-Box haben die periodischen Randbedingungen folgende Konsequenz: Fliegt ein Teilchen i über eine Seite der Box

hinaus, so tritt es auf der gegenüberliegenden Seite wieder ein. Die Anzahl der Teilchen innerhalb der Box bleibt somit immer konstant [Hab95].

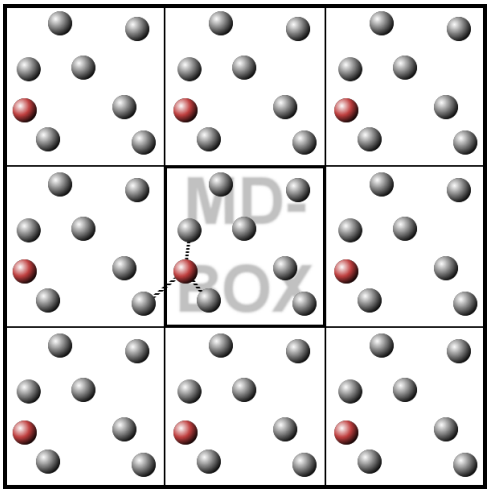

Abbildung 2.5: Periodische Randbedingungen

Bei den Periodischen Randbedingungen sind alle Abbildungen identisch zueinander. Daher erfahren auch alle Teilchen stets identische Kräfte. Diese Periodizität kann unerwünschte Effekte hervorrufen. Sind z.B Teilchen stark miteinander gekoppelt, kann die Vibration eines Teilchens eine Welle auslösen, die sich durch das System fortpflanzt und aufgrund der Periodizität wieder auf das Teilchen auftritt [HB12].

2.6.1 Minimum Image Convention

Aufgrund der periodischen Randbedingungen ergeben sich für die Abstandsberechnung zweier Teilchen neue Regeln. So entspricht der minimale Abstand des Teilchens i zum Teilchen j immer dem Abstand des nächst nähren Bildes von j (Abb. 2.6).

Die Minimum Image Convention beruht auf der Tatsache, dass nur eines der Teilchenabbilder j mit Teilchen i wechselwirkt, sobald der Cutoffradius kleiner als die Hälfte der Boxlänge ist. Ist dies nicht der Fall, kann die Minimum Image Convention nicht angewendet werden. Alle Abbilder müssten einzeln auf mögliche Wechselwirkungen mit Teilchen i überprüft werden [Hab95].

2.7 Nachbarschaftslisten

Die Berechnung der Paar-Wechselwirkungen erfordert bei Teilchenanzahlen $N > 100$ den größten Teil der Computerzeit bei einem MD-Lauf, weil die Zahl der Teilchenpaare $\frac{N(N-1)}{2}$ mit dem Quadrat der Teilchenzahl wächst. Nicht alle Paar-Wechselwirkungen sind von Relevanz, da durch den Cutoffradius r_{cut} Kraftberechnungen für einige Teilchenpaare wegfallen. Die Abstandsberechnung der Teilchenpaare kostet allerdings auch Rechenzeit. Mit Hilfe von Nachbarschaftslisten können diese jedoch enorm reduziert werden (Abb. 2.7). Die MD-Box ist in kleinere Zellen unterteilt. Die Größe der Zellen entspricht

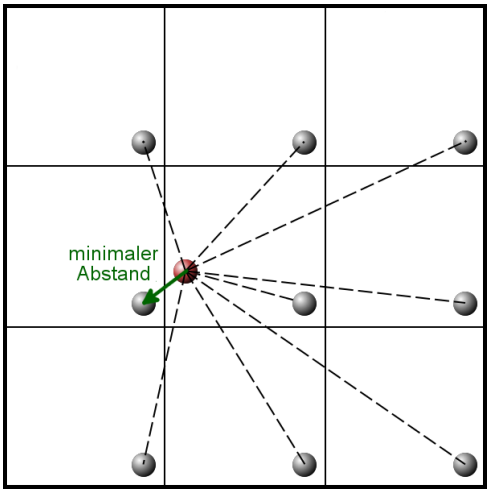

Abbildung 2.6: Minimum Image Convention

dabei r_{cut}. Jede Zelle repräsentiert eine Teilchenliste. Für die Berechnung der Paar-Wechselwirkungen werden nur die Zelle des jeweiligen Teilchens i und deren die anliegenden Nachbarzellen herangezogen. In einem dreidimensionalen Modell sind also 27 Teilchenlisten abzuarbeiten. Alle anderen Zellen sind für das Teilchen i bedeutungslos, da keine Wechselwirkungen ab einem Teilchenabstand von r_{cut} statt finden. Befindet sich die Zelle des Teilchens i am Rand der MD-Box, so müssen die umliegenden Abbilder der anderen Zellen zur Berechnung herangezogen werden [Hab95].

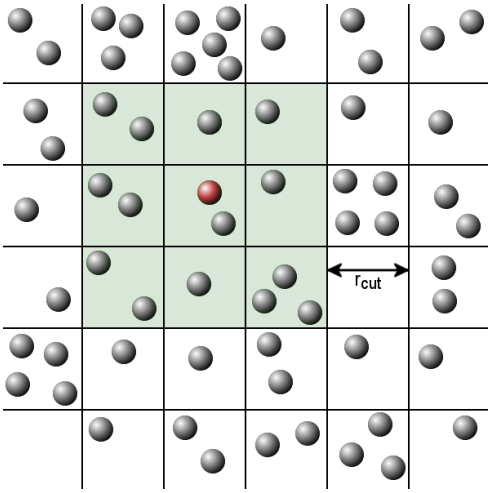

Abbildung 2.7: Nachbarschaftsliste

Durch die Verwendung von Nachbarschaftslisten steigt der Aufwand der Teilchenverwaltung. Jede Teilchenliste beansprucht weiteren Speicher. Verlässt außerdem ein Teilchen seine Zelle, so muss es sowohl aus seiner alten Teilchenliste entfernt, als auch in die Teilchenliste für die neue Zelle eingetragen werden. Das erhöht die Laufzeitkosten. Diese können allerdings ein wenig reduziert werden, indem die Aktualisierung der Teilchenlisten nur aller zehn Simulationsschritte durchgeführt wird.

2.8 Ensembles

In der Molekulardynamik ist es üblich, mit sogenannten Ensembles zu arbeiten. Sie beschreiben, welche Systemparameter während einer MD-Simulation konstant gehalten werden [Buc09].

mikrokanonisches Ensemble Teilchenanzahl N, Volumen V und Energie E

kanonisches Ensemble Teilchenanzahl N, Volumen V und Temperatur T

isothermales-isobares Ensemble Teilchenanzahl N, Druck P und Energie E

2.9 Numerische Probleme

Die Größen für Masse M, Länge L und Energie E werden im allgemeinen in Kilogramm, Meter und Joule angegeben. Dies ist jedoch für Computer ein sehr großes Problem. Sie rechnen intern mit Gleitkommazahlen, die die Menge der reellen Zahlen nicht komplett abdecken können. Die Fehlerwahrscheinlichkeit einer Fließkommaberechnung steigt mit dem Betrag des Exponenten der Gleitkommazahl. Es bietet sich also in MD-Simulationen an, alle Größen in dimensionslose Einheitsgrößen umzuwandeln.

$$M^* = \frac{M}{m} \tag{2.26}$$

$$L^* = \frac{L}{\sigma} \tag{2.27}$$

$$E^* = \frac{E}{\epsilon} \tag{2.28}$$

Für einatomige Lennard-Jones-Simulationen ergeben sich folgende Größen: Die Masse der Moleküle als Masseneinheit, die Nullstelle σ des Potentials als Längeneinheit und die Tiefe ϵ des Potentials als Einheit für die Energie [Beu].

3 Entwurf einer Modularen MD-Simulation

Das folgende Kapitel soll einen Einblick auf den Entwurfsprozess geben, der für die Entwicklung der modularen MD-Simulation notwendig war.

3.1 Vorüberlegung

Die Implementierung einer modularen MD-Simulation mit Echtzeitvisualisierung lies sich in drei Teilprobleme unterteilen (Tbl. 3.2).

Teilschritt	Teilproblem	Priorität
1	MD-Simulator	hoch
2	Einfacher Teilchenrenderer	niedrig
3	MegaMol-Plugin	mittel

Tabelle 3.1: Teilprobleme für eine MD-Simulation mit Echtzeitvisualisierung

Der MD-Simulator war Kern des Projektes. Der Teilchenrenderer und das MegaMol-Plugin waren vollkommen abhängig von ihm. Im Gegensatz dazu sollte der MD-Simulator nicht auf Teilchenrenderer oder Plugin angewiesen sein. Da der Simulator aus zusammensteckbaren Modulen bestehen sollte, bot es sich an, das Teilproblem *MD-Simulator* als C++-Bibliothek (MD-Sim) zu realisieren. MD-Sim sollte sowohl die einzelnen Module, als auch zwei vordefinierte Simulatoren beinhalten. Im zweiten Teilschritt war die Implementierung eines simplen Teilchenrenderers geplant. Für das Endresultat war der Renderer zwar irrelevant, allerdings erleichterte er durch seine Einfachheit das Debugging der MD-Bibliothek. Der Zeitaufwand für die Implementierung sollte daher so gering wie möglich sein. Die Performance des Renderers spielte keine Rolle. Für die eigentliche Teilchenvisualisierung kam vom Lehrstuhl das Visualisierungsprogramm MegaMol zum Einsatz. Dafür wurde im letzten Teilschritt ein Plugin zur Anbindung von MD-Sim an MegaMol geschrieben. Für die Implementierung aller Teilprobleme waren folgende Kriterien zu beachten (Tbl. 3.2).

Performance Ausführungsgeschwindigkeit, Framerate, Maximale Teilchenanzahl

Modularität Austauschbarkeit einzelner Komponenten

Übersichtlichkeit Strukturierter, sowie einfach zu lesender Quellcode

Portabilität Übertragbarkeit auf andere Systeme

Korrektheit Fähigkeit, Aufgaben exakt zu erfüllen

Robustheit Toleranz gegenüber Fehlerzustände

Kriterium	Priorität		
	MD-Bibliothek	Teilchenrenderer	Megamol-Plugin
Performance	mittel	niedrig	mittel
Modularität	hoch	niedrig	niedrig
Übersichtlichkeit	hoch	niedrig	mittel
Portabilität	niedrig	niedrig	niedrig
Korrektheit	hoch	niedrig	hoch
Robustheit	mittel	niedrig	mittel

Tabelle 3.2: Teilprobleme für eine MD-Simulation mit Echtzeitvisualisierung

3.2 Modulentwurf

Für die Modularisierung von MD-Sim war es notwendig, die Simulation in kleine Teilprobleme aufzuspalten. Danach wurde überprüft, ob und wie sich die einzelnen Teilprobleme zu Modulen formen lassen. Als Inspiration diente die Bibliothek *Object-MD 3.0* des Fachbereichs Physik der Technischen Universität Kaiserslautern [Ros]. Die übernommenen Teilprobleme sind mit einem Stern gekennzeichnet.

- **Simulator***

  Allgemeiner Simulationsablauf (siehe 2.1).

- **Particle***

  Datenstruktur, die Geschwindigkeit, Position, Masse und Radius eines Teilchen zusammenfasst.

- **MD-Box**

  Verwaltung der Teilchen, Verwendung von Beschleunigungsstrukturen, Einhaltung der Periodischen Randbedingungen, Abstandsberechnung zweier Teilchen mit der Minimum Image Convention (siehe 2.6)

- **Box-Iterator**

  Iteration über alle Teilchen innerhalb der MD-Box

- **Particle-Iterator**

  Iteration über alle wechselwirkenden Teilchen, ausgehend von einem Teilchen

- **Initializer**

  Erzeugung eines Anfangszustandes für die MD-Simulation

- **Integrator***

  Integration der Teilchenbeschleunigung für jeden Zeitschritt, Ermittlung der kinetischen Energie, Verwaltung der ermittelten Kräfte zwischen den Teilchen

- **Interactor***

  Kraftberechnung zwischen zwei Teilchen, sowie die Teilebrechungen der potentiellen Energie und des Virials (siehe 2.5)

3.2.1 Abhängigkeiten

Die einzelnen Teilprobleme hängen voneinander ab. Vor einem modularen Entwurf war eine Untersuchung der Abhängigkeiten zwischen den Teilproblemen notwendig (Abb. 3.1).

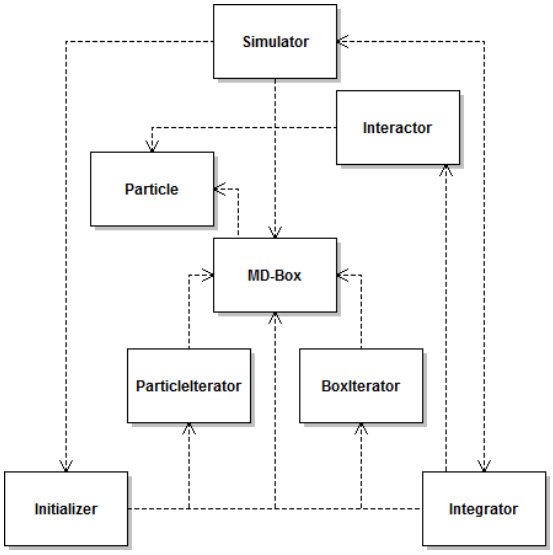

Abbildung 3.1: Abhängigkeiten zwischen den Teilproblemen

Ein Simulationsablauf setzt sich immer aus der Teilcheninitialisierung, der Teilchenverwaltung und der Integration für die einzelnen Zeitschritte zusammen. Demnach benötigt der Simulator für eine korrekte Simulationsdurchführung den Initializer, die MD-Box und den Integrator. Außerdem muss der Simulator die Systemgrößen (siehe 2.5) verwalten. Da sich Druck, Temperatur und Gesamtenergie in jeden Simulationsschritt ändern können, muss der Integrator Zugriff auf den Simulator haben. Integrator sowie Initializer ändern die Positionen und Geschwindigkeiten der Teilchen innerhalb der MD-Box. Dafür benötigen sie den Box- und den Particle-Iterator. Ändern sich die Teilchenpositionen, so müssen die periodischen Randbedingungen beachtet werden. Deswegen verwenden Integrator und Interactor die MD-Box auch direkt. Für die Kräfteverwaltung ist der Integrator zuständig. Damit er die wirkenden Kräfte zwischen den Teilchen bestimmen kann, benötigt er den Interactor. Der Interactor nimmt für die Kraftberechung zwei Teilchen als Argumente entgegen. Dabei muss er auch mit der MD-Box kommunizieren, da für die Abstandsbestimmung zweier Teilchen die Minimum Image Convention verwendet wird. Es sei zu erwähnen, dass die MD-Box das Kernstück einer Simulation darstellt. Alle Module benötigen einen direkten Zugriff auf die Box. Sie selber ist nur von der Datenstruktur der Teilchen abhängig.

3.2.2 Objektorientierter Ansatz

Genauso wie in der Biliothek Object-MD wurden die einzelnen Teilprobleme jeweils zu einer Klasse zusammengefasst. Die Modularität ließ sich durch den Entwurf geeigneter Schnittstellen erreichen (Abb. 3.2). Die Geschwindigkeit und Position eines Teilchens sind Vektorgrößen. Sie ließen sich durch eine einfache Vektor-Klasse Vec3d repräsentieren.

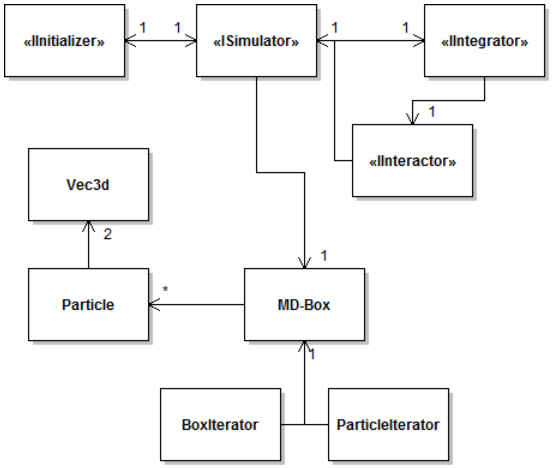

Abbildung 3.2: Objektorientierter Entwurf mit geeigneten Schnittstellen

Anders als in Object-MD sollten Beschleunigungsstrukturen austauschbar sein. Demnach konnte die MD-Box keine fest implementierte Klasse sein. Die Aufteilung der MD-Box in zwei Teilprobleme war keine Lösung. Zu stark hingen Teilchenverwaltung und Beschleunigungsstruktur voneinander ab. Wollte man die Modularität ebenfalls durch Vererbung erreichen, so hätten auch die Iteratoren geeigntete Schnittstellen benötigt. Das hätte sich negativ auf die Performance der MD-Simulation ausgewirkt, da Polymorphie mehr Verwaltungsaufwand für das Laufzeitsystem bedeutet. Für einzelne Methodenaufrufe fiel das nicht ins Gewicht. Allerdings werden die Methoden der Iteratoren sehr häufig benutzt, sind sie doch essentiell für Schleifendurchläufe. Dieses Problem lies sich mit Hilfe von Templates lösen. Jede Klasse, die die MD-Box verwendet, wurde als Template realisiert (Abb. 3.3). Außerdem wurde der Particle-Wrapper als eine weitere Klasse hinzugefügt. Er kapselt die Datenstruktur Particle in sich und war notwendig für die interne Teilchenverwaltung der MD-Box (siehe 3.4.1). Nachdem der Entwurf für MD-Sim stand, wurde der genaue Ablauf eines Simulationsschrittes festgelegt (Abb. 3.4).

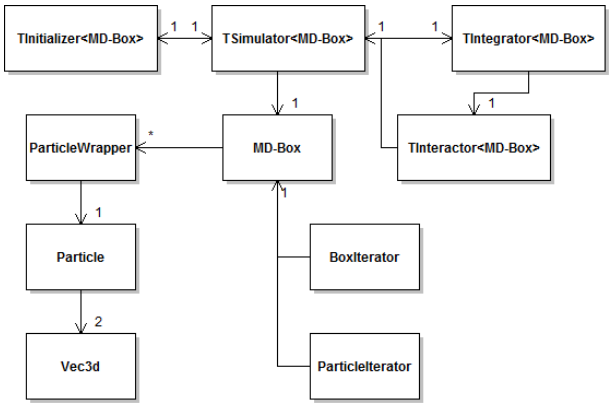

Abbildung 3.3: Objektorientierter Entwurf mit Particle-Wrapper und geeigneten Templates

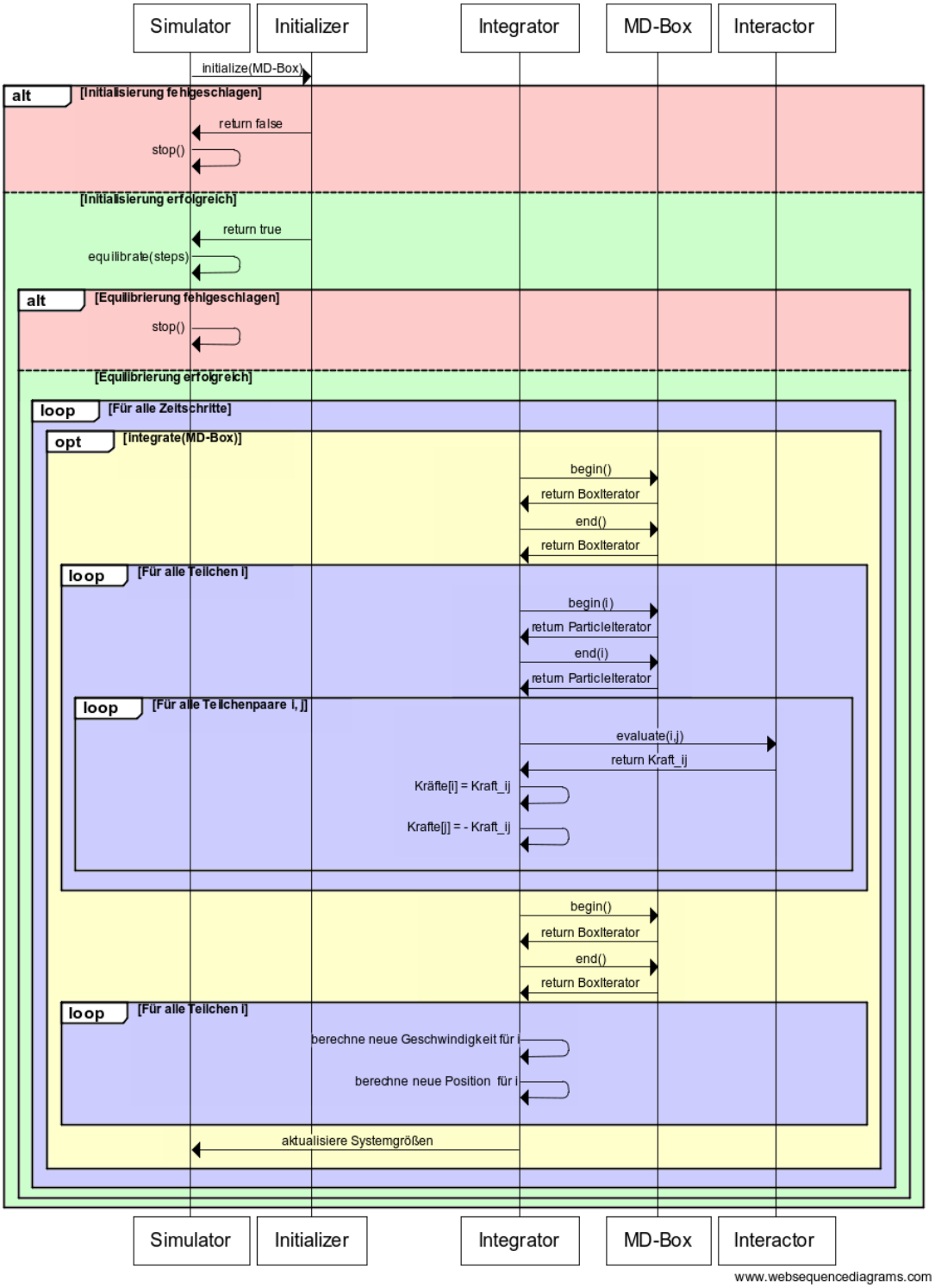

Abbildung 3.4: Interaktionen der Module, dargestellt durch ein Sequenzdiagramm

3.3 Das Simulator-Modul

Das Simulator-Modul für den allgemeinen Ablauf der MD-Simulation zuständig. Die meisten der dafür notwendigen Methoden wurden bereits implementiert (Abb. 3.5). Für die Erstellung eines Simulator-Moduls ist lediglich die Implementierung die Konstruktors und der abstrakten routine()-Methode notwendig. Innerhalb der routine()-Methode befindet sich der Code, der für jeden Simulationsschritt ausgeführt werden soll. Obwohl der Ablauf der Simulationsschritte immer der selbe ist, wurde er nicht fest implementiert. Der Grund dafür ist die optionale Verwendung von Ensembles (siehe 2.8). Der Nutzer soll selber entscheiden, wie bzw. ob überhaupt Systemparameter konstant gehalten werden sollen. Innerhalb des Konstruktors findet die Initialisierung der MD-Box mit Hilfe eines Initializer-Moduls statt. Außerdem werden die gewünschten Module für Initialisierung, Integration und Kraftberechnung definiert und anschließend zusammengesteckt. Bevor eine Simulation starten kann, muss sie noch equilibriert werden (siehe 2.1). Dafür ist die equilibrate()-Methode zuständig. Als Parameter nimmt sie die Anzahl der Schritte entgegen, die der Simulator vor der eigentlichen Simulation durchlaufen soll.

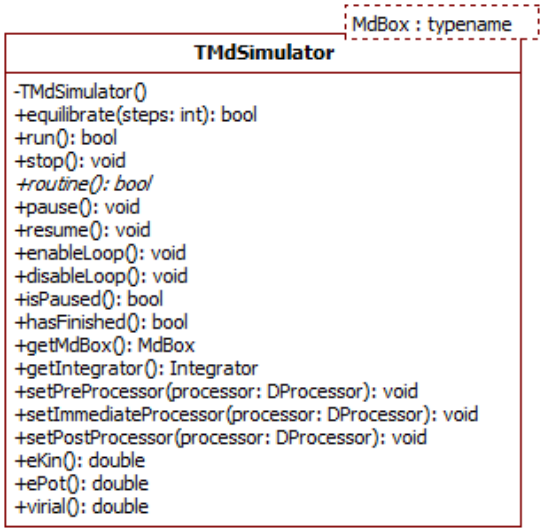

Abbildung 3.5: TMdSimulator<MdBox>

3.3.1 Die run()-Methode

Im Folgenden soll etwas genauer auf die run()-Methode eingegangen werden. Für jeden Simulations-Zeitschritt ruft sie die routine()-Methode auf und verwendet dafür eine eigene Programm-Schleife. In einigen Fällen ist das nicht erwünscht. Haben andere Anwendungen eigene Programmschleifen (z.B. Freeglut-Anwendungen), so ist ihre Kopplung mit dem Simulator nur mit zwei verschiedenen Threads möglich. Aus diesem Grund lässt sich Schleife innerhalb der run()-Methode abschalten. Die run()-Methode simuliert dann nur einen Zeitschritt. Für jeden weiteren Zeitschritt muss sie manuell aufgerufen werden. Ihr Rückgabewert gibt an, ob für die Simulation noch weitere Zeitschritte notwendig sind. So lässt sich auch innerhalb eines Threads Simulator und Anwendungsprogramm miteinander koppeln.

3.3.2 Pre-, Immediate- und Postprocessor

Eine Besonderheit des Simulator-Modul ist die Möglichkeit Quellcode vor, während und nach der Simulationsschleife nachträglich zu injizieren. So kann der Simulationsablauf für einen bereits fertig implementierten Simulator verändert werden, ohne seinen Quellcode zu verändern. Das ist möglich durch die Verwendung von Deleganten. Deleganten sind Typen, die eine Methodensignatur definieren. Für eine Codeinjektion reicht es aus eine Lambda-Funktion mit einer Simulator-Modul-Referenz als Parameter zu schreiben, und sie dem Simulator vor der Ausführung zu übergeben. Es stehen drei Methoden zur Verfügung:

- **setPreProcessor(DProcessor p)**
  Die übergebene Delegation wird vor dem ersten Simulationsschritt ausgeführt.
- **setImmediateProcessor(DProcessor p)**
  Dir übergebene Delegation wird nach jedem Simulationsschritt ausgeführt
- **setPostProcessor(DProcessor p)**
  Die übergebene Delegation wird ausgeführt, nachdem die Simulation beendet bzw. abgebrochen wurde.

Ein typischer Anwendungsfall wäre das Schreiben der Teilchentrajektorien in eine Datei für ein Post-Processing. In MD-Sim steht bereits ein File-Writer zur Verfügung. Man definiert also eine Lambda-Funktion für den Immediateprocessor und ruft in ihr den File-Writer auf. Nach jedem Simulationsschritt werden nun die aktuellen Trajektorien in die Datei geschrieben.

3.4 Das Box-Modul

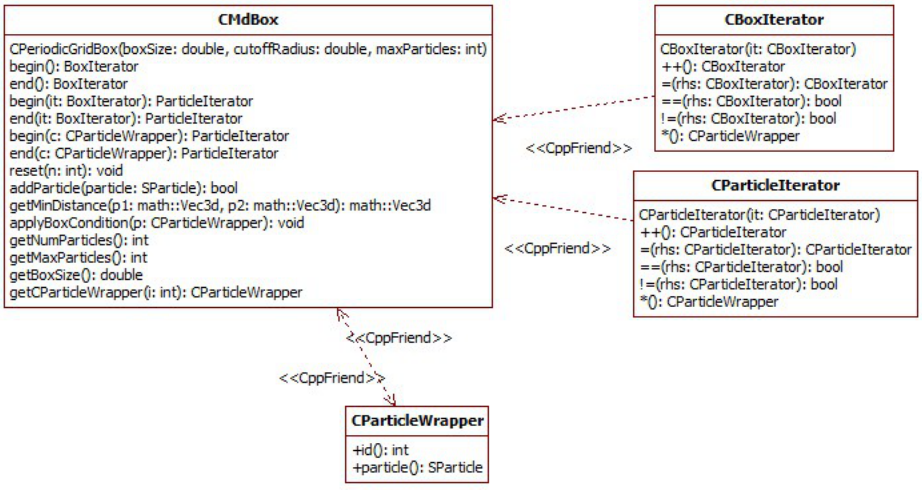

Abbildung 3.6: TMdSimulator<MdBox>

Für das Box-Modul gibt es keine Templatevorlage. Um die Spezialisierung anderer Module für eine eigens entwickelte Box zu vermeiden, sollte sich daher an den Methoden bereits implementierten Boxen

orientiert werden (Abb. 3.6). Das Box-Modul ist hauptsächlich für die Verwaltung der Partikel zuständig. In der applyBoxCondition()-Methode befindet sich der Code für die Einhaltung der Periodischen Randbedingungen (siehe 2.6). Sie nimmt ein Teilchen entgegen und erneuert seine Position, sobald es sich außerhalb der Box befindet. Die getMinDistance()-Methode ist zuständig für die Berechnung des minimalen Abstandes zweier Teilchen. Sie verwendet dafür die Minimum Image Convention (siehe 2.6.1). Die maximale Anzahl der Teilchen, die die MD-Box verwalten kann, muss bereits im Konstruktor festgelegt werden. Zur Laufzeit kann sie über die getMaxParticles()-Methode abgefragt werden. Über getNumParticles() lässt sich die Anzahl der Teilchen ermitteln, die sich aktuell in der MD-Box befinden. Möchte man die Teilchenkapazität der MD-Box zur Laufzeit erhöhen, so kann die reset()-Methode dafür verwendet werden. Hierbei ist zu beachten, dass alle Teilchen, die sich vor dem Aufruf in der MD-Box befanden, gelöscht werden.

3.4.1 Particle-Wrapper

Damit die Teilchen von der MD-Box verwaltet werden können, benötigen sie sogenannte Verwaltungsparameter. Die benötigten Parameter sind jedoch von Box zu Box unterschiedlich. So benötigt die *CPeriodicBox* nur eine ID. Die *CPeriodicGridBox* hingegen muss zusätzlich den aktuellen Zellindex für die Verwaltung der Nachbarschaftslisten abspeichern (siehe 4.1.4). Die Speicherung der Verwaltungsparameter innerhalb der Teilchen-Struktur ist also nicht möglich. Abhilfe verschafft der Particle-Wrapper, der Teilchen und nötige Parameter in sich kapselt. Jede Box implementiert dafür seinen eigenen Wrapper. Die Verwaltungsparameter besitzen höchstens einen Lesezugriff und können nur von der MD-Box verändert werden. So kann ein versehentliches Überschreiben der Verwaltungsparameter vom Nutzer ausgeschlossen werden.

3.4.2 Boxiterator und Particleiterator

Für die Erstellung der Iteratoren ist die MD-Box zuständig. Wie es auch in der Standard-Template-Library für C++ üblich ist, erhält man Start- und End- Iterator mit den Methoden begin() und end(). Für die Erstellung eines Particle-Iterators müssen den Methoden zusätzlich ein Box-Iterator übergeben werden, der auf ein Partikel innerhalb der MD-Box verweist. Durch die Verwendung des Iterator-Patterns bleiben Implementierungsdetails der MD-Box dem Nutzer verborgen. Dadurch gestaltet sich der Zugriff auf die Partikel einfach und intuitiv. Allerdings haben Iteratoren auch Nachteile. Beispielsweise muss der Paticle-Iterator, unter der Verwendung von Nachbarschaftslisten, Referenzen für 27 Teilchenlisten speichern und verwalten. Die Folgen sind erhöhte Speicher- und Laufzeitkosten. Außerdem erschweren Iteratoren die Parallelisierung von Schleifen. Wahrscheinlich ist das der Grund, warum Object-MD sich gegen die Verwendung von Iteratoren entschieden und alle Schleifen fest implementiert hat. Im Fall dieser Arbeit steht jedoch das Kriterium *Übersichtilchkeit* über dem Kriterium *Performance*.

In der entstandenen Bibliothek MD-Sim, sollte dennoch die Möglichkeit bestehen, einzelne Module zu parallelisieren. Der Zeitaufwändigste Teil einer MD-Simulation ist die Berechnung der Kräfte, die zwischen den Teilchen wirken. Zum Verständnis sei der C++-Code für eine typische Kraftberechnung gegeben. Der Rückgabewert der evaluate()-Methode ist ein Typedef und beschreibt ein Tupel, in dem alle Ergebnisse, einschließlich der Kraft abgespeichert sind (siehe 3.5).

```
// Iteration ueber alle Teilchen innerhalb der MD-Box
auto iEnd = mdBox.end();
for(auto it = mdBox.begin() ; it != iEnd ; ++it){
  // Iterator liefert Particle-Wrapper, in dem das Teilchen gekapselt ist
  SParticle p1 = (*it).particle();

  // Iteration ueber alle wechselwirkenden Teilchen, ausgehend vom aktuellen Teilchen
  auto jEnd = mdBox.end(it);
  for(auto jt = mdBox.begin(it) ; jt != jEnd ; ++jt){
  SParticle p2 = (*jt).particle();
  // Kraefteberechnung
  Result result = interactor.evaluate(p1,p2);
  }
}
```

Bezüglich der Parallelisierung ist das Iterator-Pattern denkbar ungeeignet. So lassen sich beispielsweise mit *OpenMP* [Ope] nur for-Schleifen parallelisieren, die einen Integertyp als Zähler aufweisen. Am effizientesten wäre eine Parallelisierung der äußeren Schleife der Kraftberechnung. Die Ausführung der inneren Schleife könnte so in verschiedenen Threads ausgeführt werden. Für den Entwurf bedeutet das, dass der Box-Iterator ersetzt werden muss. Anstelle mit Iteratoren über alle Partikel zu traversieren, ist es deshalb auch möglich, über Indices direkt auf die einzelnen Particle-Wrapper innerhalb der MD-Box zuzugreifen. Außerdem muss sich der Particle-Iterator auch mit einem Particle-Wrapper erstellen lassen können. Eine Kraftberechnung kann also auch wie folgt durchgeführt werden.

```
// Iteration ueber alle Teilchen innerhalb der MD-Box
getNumParticles() liefert Anzahl der Teilchen innehalb der Box
for(int i = 0 ; i < mdBox.getNumParticles() ; ++i){
  // Beschaffung des Particle-Wrappers, in dem das Teilchen gekapselt ist
  auto c = mdBox.getParticleWrapper(i);

  // Iteration ueber alle wechselwirkenden Teilchen, ausgehend vom aktuellen Teilchen
  auto jEnd = mdBox.end(c);
  for(auto jt = mdBox.begin(it) ; jt != jEnd ; ++jt){
  SParticle& p2 = (*jt).particle();
  // Kraefteberechnung
  Result result = interactor.evaluate(c.particle(),p2);
  }
}
```

3.5 Das Interactor-Modul

Das Integrator-Modul ist neben der Kraftberechnung auch für die Berechnung der Teilviriale und der potentiellen Energie der Teilchen zuständig. Eine Methode gestattet allerdings nur einen Rückgabewert. Für den aktuellen Entwurf bieten sich zwei Varianten zur Ergebnisrückgabe an.

- **Rückgabe mit Referenzparametern**

  Der Methode werden beim Aufruf Variablen als Referenzen mitgegeben und innerhalb der Methode entsprechend verändert.

- **Rückgabe als Tupel**

  Die Methode liefert ein Tupel zurück, in dem Kraftvektor, Teilvirial und potentielle Energie gespeichert sind.

Die Entscheidung fiel auf die Ergebnisrückgabe mit Hilfe eines Tupels. Diese Variante ist langsamer als die Rückgabe mit Hilfe von Referenzparametern, da das Tupel bei der Rückgabe kopiert werden muss. Allerdings ist die Verwendung der Funktionsparameter als Rückgabewert weniger intuitiv und somit weniger benutzerfreundlich. Aus Gründen der Übersichtlichkeit wurde der Typ-Name *Result* für das Tupel eingeführt (Abb. 3.7).

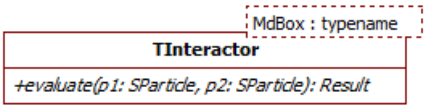

Abbildung 3.7: TInteractor<MdBox>

3.6 Das Integrator-Modul

Das Integrator-Modul ist für die nummerische Integration der Teilchenbeschleunigung für jeden Zeitschritt sowie für die Ermittlung der kinetischen Energie zuständig. Das Modul besitzt nur die integrate()-Methode, die es zu implementieren gilt (Abb. 3.8). Als Parameter bekommt sie die MD-Box der aktuellen Simulation übergeben. Für die Berechnung der Teilchenbeschleunigungen ist immer eine Kraftberechnung notwendig. Daher benötigt das Integrator-Modul immer einen Interactor, um seine Aufgabe korrekt auszuführen. Bei allen Integrator-Modulen von MD-Sim wird der Interactor beim Konstruktoraufruf mit übergeben.

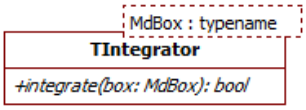

Abbildung 3.8: TIntegrator<MdBox>

3.7 Das Initializer-Modul

Das Initializer-Modul ist für die Erzeugung eines Startzustandes für die Simulation zuständig. Das Modul besitzt nur die init()-Methode, die es zu implementieren gilt (Abb. 3.9). Als Parameter bekommt sie die MD-Box der aktuellen Simulation übergeben. Im Gegensatz zur Equilibrierung sollte die Initialisierung austauschbar sein. Deswegen wurde für die Initialisierung eine separate Klasse angelegt. So kann die Initialisierung entweder über einen Algorithmus oder über das Auslesen einer Datei, mit Teilchentrajektorien als Inhalt, erfolgen.

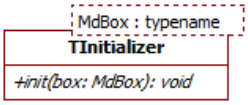

Abbildung 3.9: TInitializer<MdBox>

3.8 Der Render-Adapter

Beginnt man mit dem Bau eines Visualisierungsprogrammes für einen Simulator, so fällt ein Problem im Entwurf auf. Es existiert keine feste Schnittstelle für die Simulatorklasse. Das kann es auch nicht, da der Simulator eine Templateklasse ist, die von der jeweils gewählten MD-Box abhängt. Das Visualisierungsprogramm benötigt aber eine Referenz auf einen Simulator, um an die Simulationsdaten zu gelangen. Man möchte allerdings nicht für jeden neuen Simulator ein neues Visualisierungsprogramm bauen. Mit Hilfe des Adapter-Patterns konnte dieses Problem gelöst werden. Innerhalb einer Struktur werden eine Liste mit Zeigern auf alle Teilchen und Zeiger auf alle Systemgrößen, die für die Visualisierung notwendig sind, abgespeichert. Außerdem werden alle Methoden, die für die Bedienung des Simulators notwendig sind, als Delegationen in der Struktur gespeichert (Abb. 3.10). Diese Struktur, genannt Render-Adapter, dient nun dem Visualisierungsprogamm als eine Art Kommunikationsbrücke. Obwohl das Visualisierungsprogramm den Simulator selber nicht kennt, hat es trotzdem Zugriff auf alle wichtigen Simulationsgrößen und Simulationsmethoden.

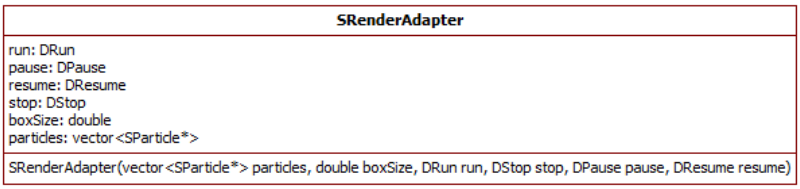

Abbildung 3.10: SRenderAdapter

Aktuell kann der Renderer-Adapter vier Methoden zur Bedienung des Simulators abspeichern.

- **DRun für run()-Methode**
  Delegation für die manuelle Ausführung der Simulationsschritte, falls Simulationsschleife abgeschaltet ist (disableLoop()-Methode).
- **DPause für pause()-Methode**
  Delegation für das Pausieren der Simulation
- **DResume für resume()-Methode**
  Delegation für die Aufhebung einer Simulationspause
- **DStop für stop()-Methode**
  Delegation für das Beenden der Simulation

4 Evaluation

In diesem Kapitel wird die Bibliothek MD-Sim, die während der Arbeit entstanden ist, einer Evaluation unterzogen. Es soll gezeigt werden, dass Module sich beliebig austauschen lassen und MD-Sim durch neue Module erweitert werden kann. Außerdem soll gezeigt werden, dass die Parallelisierung einzelner Module grundsätzlich möglich ist. Neben der Austauschbarkeit der Module soll auch die Unabhängigkeit der Simulatoren bezüglich der Datenausgabe und der Datenvisualisierung bewiesen werden.

4.1 Verfügbare Module

Für die Evaluation wurden insgesamt zehn Module implementiert. Um näher zu bringen, wie vielseitig der Entwurf von MD-Sim ist, wurde darauf geachtet, dass sich die Module so wenig wie möglich ähneln. In diesem Abschnitt sollen die Module vorgestellt und auf ihre Besonderheiten eingegangen werden.

4.1.1 Initializer-Module

Ein Initializer-Modul ist nicht zwangsläufig für einen Simulator notwendig. Die Teilchen können auch manuell über die MD-Box gesetzt werden. Bei einer Simulation mit mehreren Teilchen kommen die Initializer-Module zum Einsatz. Für die Teilcheninitialisierung stehen zwei unterschiedliche Module zur Verfügung.

- **Single-Atomic-Initializer**

  ~ordnet alle Teilchen gleichmäßig innerhalb der MD-Box mit Hilfe eines Rasters an. Lassen sich die Teilchen nicht gleichmäßig aufteilen, können *leere Reihen* entstehen (Abb. 4.1). Unter Berücksichtigung der gewünschten Systemenergie wird für jedes Teilchen eine Zufallsgeschwindigkeit bestimmt. Gewünschte Systemenergie, Teilchenradius und Teilchenmasse müssen im Konstruktor angegeben werden.

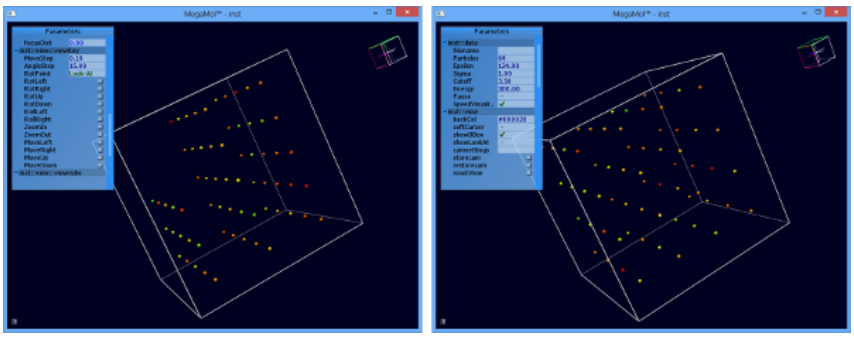

Abbildung 4.1: Boxinitialisierung mit SingleAtomicInitializer, Links: 65 Teilchen, Rechts: 64 Teilchen

- **XYZ-Initializer**

  ~liest Position und optional Geschwindigkeit, Masse und Radius der Teilchen aus einer erweiterten XYZ-Datei aus. Im Konstruktor muss nur der Pfad der Datei angegeben werden. Ist der Dateipfad fehlerhaft, wird die MD-Box nicht befüllt und es erscheint eine Fehlermeldung auf der Konsole. Die Datei sollte nicht fehlerhaft sein, da es dafür keine Benachrichtigung auf der Konsole ausgibt. Statdessen versucht Initializer die Datei weiter zu parsen. Die Folge sind falsche Teilchenwerte.

4.1.2 Interactor-Module

Für die Ermittlung der Wechselwirkungskräfte zweier Teilchen stehen zwei unterschiedliche Module zur Verfügung.

- **Lennard-Jones-Interactor**

  ~berechnet die wirkenden Kräfte zweier Teilchen mit Hilfe des Lennard-Jones-Potentials. Dabei wird angenommen, dass alle Teilchen die gleiche Masse besitzen. Die Lennard-Jones-Parameter σ, ϵ, sowie der Cutoffradius r_{cut} müssen im Konstruktor angegeben werden.

- **Gravity-Interactor**

  ~berechnet die wirkenden Kräfte zweier Objekte mit Hilfe des Gravitations-Potentials. Die unterschiedlichen Massen zweier Objekte werden dabei berücksichtigt.

4.1.3 Integrator-Module

Für die nummerische Integration stehen vier Integrator-Module zur Verfügung.

- **Euler-Integrator**

  ~ermittelt Position, sowie Geschwindigkeit mit Hilfe des Euler-Verfahrens.

- **Leapfrog-Integrator**

  ~ermittelt Position, sowie Geschwindigkeit mit Hilfe des Leapfrog-Verfahrens.

- **Velocity-Verlet-Integrator**

  ~ermittelt Position, sowie Geschwindigkeit mit Hilfe des Velocity-Verlet-Verfahrens.

- **Velocity-Verlet-Multicore-Integrator**

  ~ermittelt Position, sowie Geschwindigkeit mit Hilfe des Velocity-Verlet-Verfahrens. Die dafür notwendigen Kraftberechnungen wurden mit Hilfe der Open-MP API [Ope] parallelisiert. Box-Iterator und Particle-Iterator sind weiterhin verfügbar, können aber durch den direkten Zugriff auf die Patikel-Wrapper ersetzt werden.

Bei den Integratoren ist der Velocity-Verlet-Multicore-Integrator besonders hervorzuheben. Er ist der Beweis für die Möglichkeit der Parallelisierung einzelner Module innerhalb von MD-Sim. Mit der Aufspaltung der Kräfteberechnung in mehrere Threads kann die Rechenzeit spürbar reduziert werden. So konnte auf dem Lenovo Ideapad Y500 die Simulationszeit eines Lennad-Jones-Simulators mit einer PeriodicGridBox als MD-Box um ca. ein Drittel gesenkt werden (Tbl. 4.1). Dabei sei zu erwähnen, dass beim Ideapad die Taktfrequenz des Prozessors im Singlecore-Betrieb (3.4 GHz) kleiner ist als im Multicore-Betrieb (2.4 GHz).

Integrator	Rechenzeit, PeriodicBox (in sec)	Rechenzeit, PeriodicGridBox (in sec)
Euler	2098.23	33.27
Leapfrog	2096.11	33.32
Velocity-Verlet	2095.86	32.47
Velocity-Verlet-Multicore	1911.42	20.18

Tabelle 4.1: Geschwindigkeitstest der Integratoren mit dem Lennard-Jones-Simulator
Testsystem: 16 GByte DDR3, 2x GeForce GT 650M ,Intel Core i7-3630QM, Windows 8
$\epsilon = 124, \sigma = 1, r_{cut} = 3.5, \Delta t = 0.01, E = 300,$
BoxSize $= 50, 10000$ *Teilchen* , 1000 *Zeitschritte*

Trotz möglicher Parallelisierung lassen sich Schleifendurchläufe auch weiterhin mit Iteratoren realisieren. Für Neueinsteiger ist das ein großer Vorteil gegenüber Object-MD, da Iteratoren zur Verständlichkeit des Quellcodes beitragen.

4.1.4 Box-Module

Für die Teilchenverwaltung stehen zwei Box-Module zur Verfügung.

- **Periodic-Box**

  ~verwaltet die Teilchen mit Hilfe eines STL-Vektors. Die Box verwendet keine Beschleunigungsstruktur. Für die Iteration stehen die Iteratoren des STL-Vektors zur Verfügung. Sie wurden einfach im Box-Iterator und Particle-Iterator gekapselt.

- **Periodic-Grid-Box**

  ~verwaltet die Teilchen mit Nachbarschaftslisten als Beschleunigungsstruktur. Anfangs wurde versucht, die Nachbarschaftslisten mit einer Linked List aus der STL-Bibliothek zu realisieren. Der Particle-Iterator musste 27 List-Iteratoren in sich kapseln und verwalten. Leider war das Ergebnis nicht zufriedenstellend. So war die Periodic-Box teilweise schneller, obwohl sie keine internen Beschleunigungsstrukturen besitzt. Die Implementierung einer eigenen Linked List war notwendig. Dadurch ließen sich die Iteratoren der MD-Box besser auf die Nachbarschaftslisten anpassen, da so der Aufbau der Linked List bekannt war.

Beide Boxen unterscheiden sich sehr stark voneinander. Während die Periodic-Box im Grunde genommen nur einen STL-Vektor in sich kapselt, vewendet die Periodic-Grid-Box eine selbst implementierte Linked List. Im Simulationstest mit 10.000 Teilchen war die Periodic-Grid-Box ca. 100 mal schneller

als die Periodic-Box (Tbl. 4.1). Da sich die Boxen beliebig austauschen lassen, ist es sogar möglich eine völlig neue Box zu entwickeln, in der statt periodischer Randbedingungen Wände für die Box existieren. Mit Object-MD ist der Boxaustausch nicht möglich.

4.2 Der Lennard-Jones-Simulator

Mit dem Lennard-Jones-Simulator lassen sich einfache Gase oder Flüssigkeiten simulieren (Abb. 4.2). Für die Kräfteberechung verwendet der Simulator das Lennard-Jones-Interactor-Modul und für die Teilchenverwaltung das Periodic-Grid-Box-Modul. Das Integrationsmodul wurde nicht fest implementiert und kann mit Hilfe von Enumerationen beim Konstruktor-Aufruf ausgewählt werden. Dadurch ist der Simulator auch ein Beweis für die Austauschbarkeit einzelner Module. Der Simulator arbeitet mit dem mikrokanonischen Ensamble. Während der Simulation wird die Gesamtenergie des Systems auf einem konstanten Wert gehalten.

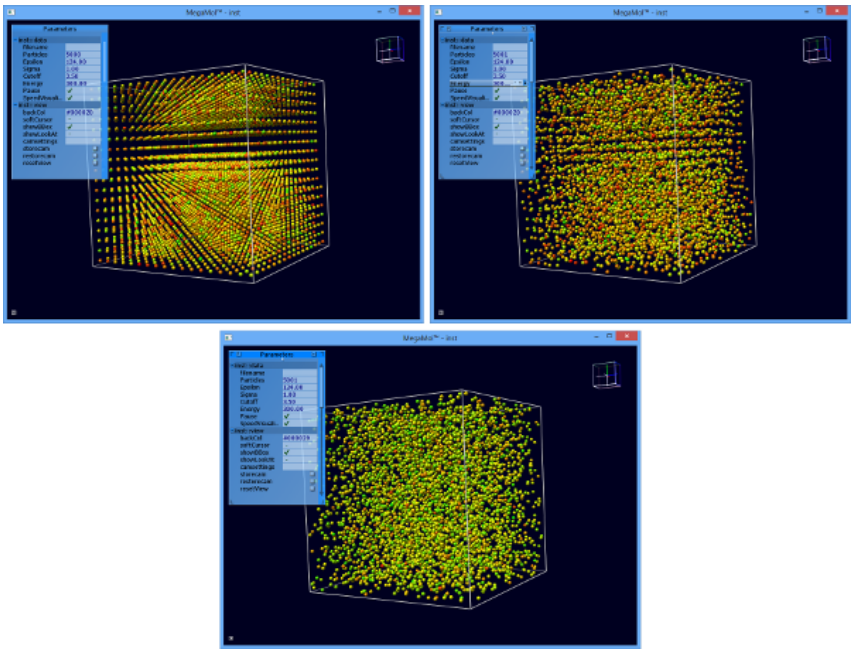

Abbildung 4.2: Simulation einer Flüssigkeit mit dem Lennard-Jones-Simulator

4.3 Der Planet-Simulator

Der Aufbau der Bibliothek MD-Simulation ist so allgemein gehalten, dass sie sogar zweckentfremdet werden kann. Als Beweis dient ein kleiner Planeten-Simulator (Abb. 4.3). Für den Simulator musste nur ein eigenes Interactor-Modul (der Gravity-Interactor) geschrieben werden. Die restlichen Module konnten einfach übernommen werden. Für die Initialisierung verwendet der Planet-Simulator den XYZ-Initializer. Positionen, Startgeschwinigkeiten, Größen und Massen der Planeten werden in einer XYZ-Datei definiert und vor dem Simulationsstart ausgelesen. Für die Planetenverwaltung kommt die Periodic-Box zum Einsatz. Als Integrator verwendet der Planet-Simulator den Leapfrog-Integrator.

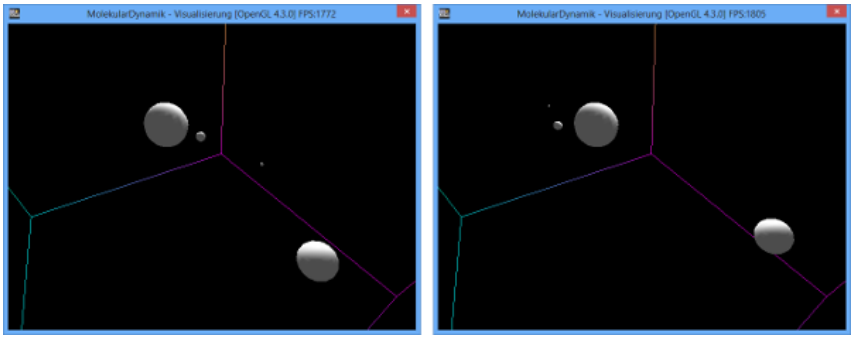

Abbildung 4.3: Simulation der Planetenbahnen für ein Sonnensystem

Die Planetensimulation sollte nicht zu ernst genommen, und eher als Beweis für die Vielseitigkeit der Bibliothek aufgefasst werden.

4.4 Datenausgabe

Eine benutzerdefinierte Ausgabe von Simulations-Daten ist mit Hilfe des Pre- , Immediate- und Postprocessor möglich. Ist z.B. die Ausgabe aller Teilchenpositionen für jeden Integrationsschritt gewünscht, so muss lediglich eine Funktion für den Immediateprocessor geschrieben werden, in der über alle Teilchenpositionen in eine Datei geschrieben werden. Für das Schreiben der Teilchenpositionen in eine Datei steht bereits ein File-Writer in der Bibliothek zur Verfügung. Er speichert die Positionen im XYZ-Format ab. Die so geschriebene Datei lässt sich mit dem Viewer VMD-Viewer [VMD] nachträglich visualisieren und animieren (Abb. 4.4). Mit Hilfe des Pre- und Postprocessors sind auch die Benchmarks für den Lennard-Jones-Simulator entstanden (Tbl. 4.1). Im Preprocessor wurde die aktuelle Zeit vor dem Simulationsstart gemmessen und im Postprocessor aktuelle Zeit nach dem Simulationsende.

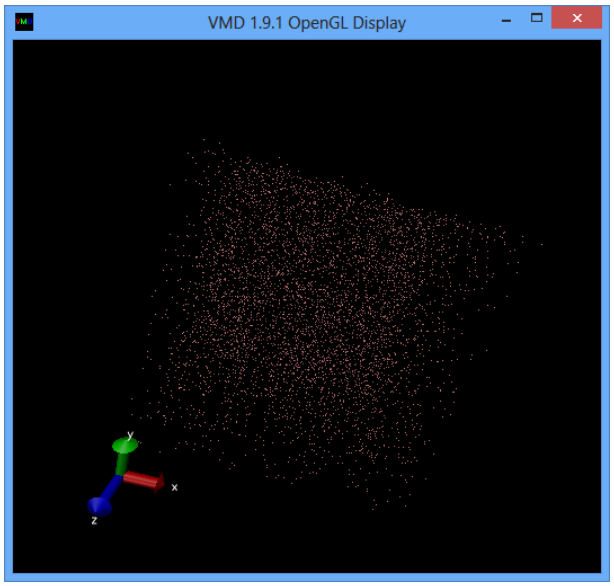

Abbildung 4.4: Visualisierung einer Lennard-Jones-Simulation durch ein Postprocessing mit VMD

4.5 Datenvisualisierung

Anders als Object-MD lassen sich MD-Simulationen mit MD-Sim in Echtzeit visualisieren. Für die Datenvisualisierung während der Modulentwicklung diente ein einfaches Visualisierungsprogramm namens *IrrMol*, beruhend auf der Realtime 3D Engine Irrlicht. Möchte man aber sehr viele Teilchen (< 5.000) simulieren, so reicht IrrMol nicht mehr aus. Mit MegaMol können auch große Teilchenmengen (< 1.000.000) problemlos visualisiert werden. Die Datenvisualisierung wurde komplett vom Simulator entkoppelt. Das ermöglicht eine Visualisierung mit verschiedenen Programmen. Es stehen zwei Visualisierungsprogramme zur Verfügung (Abb. 4.5).

- **IrrMol**

  ~ist ein einfaches Visualisierungsprogramm und eignet sich daher für das Debugging. Mit IrrMol lassen sich bis zu 2.500 Teilchen rendern. Für Teilchenanzahlen < 5.000 ist IrrMol nicht mehr geeignet, da die Framerate extrem niedrig wird. Auf dem Ideapad Y 500 betrug sie durchschnittlich nur noch 3 FPS.

- **MegaMol-Plugin**

  ~ist ein Visualisierungsprogramm, speziell ausgelegt für die Visualisierung von Molekularen Strukturen. Mit dem MegaMol-Plugin können bis zu 20.000 Teilchen gerendert werden. Außerdem lassen sich die Teilchengeschwindigkeiten mit Farben darstellen. Die Farbskala geht von Grün über Gelb zu Rot. Grün repräsentiert keine Teilchenbewegung. Rot steht für die größte Geschwindigkeit, die während der Simulation registriert wurde. Zudem besitzt das MegaMol-Plugin ein Panel, in dem sich z.B Teilchenanzahl, Systemenergie oder die Lennard-Jones-Parameter einstellen lassen.

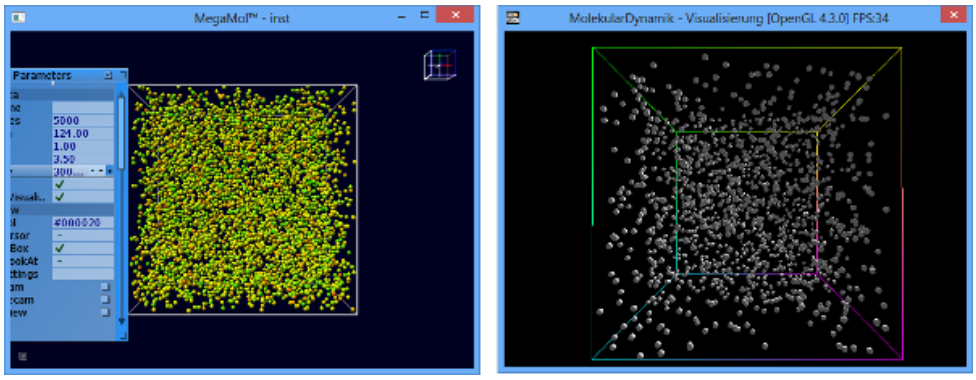

Abbildung 4.5: Visualisierung mit MegaMol (links) und IrrMol (rechts)

4.6 Ausblick

Aus zeitlichen Gründen konnten während der Arbeit nicht alle Ideen verwirklicht werden. So war z.B. die Ausführung der Kräfteberechnung auf der Grafikkarte geplant. Die prinzipielle Möglichkeit der Parallelisierung einzelner Module wurde bereits mit dem Velocity-Verlet-Multicore-Integrator bewiesen. Für eine Weiterentwicklung von MD-Sim könnte also ein Integrator-Modul entwickelt werden, das eine Kraftberechnungen mit Hilfe von *CUDA* [CUD] durchführt. Um die Performance noch weiter zu steigern, wäre auch die Implementierung weiterer Boxen mit anderen Beschleunigungstrukturen wünschenswert. Des weiteren kann der Filewriter von MD-Sim momentan nur Dateien im XYZ-Format schreiben. Gleiches gilt für das Auslesen von Dateien mit dem XYZ-Initializer. Die Unterstützung weiterer Dateiformate könnten zukünftig noch realisiert werden.

Literaturverzeichnis

[Beu] BEU, Titus: *Nanosim Laboratory for Simulations of Nanostructured Systems - Fortran Based*. http://phys.ubbcluj.ro/~tbeu/teaching.html, . – Accessed: 2013-09-09

[Buc09] BUCHELE, Patric: *Klassische Molekulardynamik Simulationen*. http://www.icp.uni-stuttgart.de/~icp/mediawiki/images/5/50/Hs0910_buechele_ausarbeitung.pdf, 2009. – Accessed: 2013-09-09

[CUD] *Nvidia Parallele Berechnungen mit CUDA*. http://www.nvidia.de/object/cuda-parallel-computing-de.html, . – Accessed: 2013-09-09

[Dil] DILGER, Franziska: *Molekulardynamik*. http://www-m2.ma.tum.de/foswiki/pub/M2/Allgemeines/HauptseminarWohlmuthWS10/Ausarbeitung_Molekulardynamik.pdf, . – Accessed: 2013-09-09

[Gol] *GoldBook - Compendium of Chemical Terminology*. http://goldbook.iupac.org/I03103.html, . – Accessed: 2013-09-09

[Hab95] HABERLANDT, Reinhold: *Molekulardynamik, Grundlagen und Anwendungen*. Braunschweig/Wiesbaden : Vieweg, 1995

[HB12] HARTZ-BEHREND, Karsten: *Einsatz der klassischen Molekulardynamik fuer die quantitative Vorausberechnung des Benutzungsverhaltens beim Loeten*. http://athene.bibl.unibw-muenchen.de:8081/doc/90053/90053.pdf, 2012. – Accessed: 2013-09-09

[Kie13] KIERFELD, Jan: *Computational Physics*. http://t1.physik.tu-dortmund.de/kierfeld/teaching/CompPhys_13/, 2013. – Accessed: 2013-09-09

[Kum10] KUMMEROW, Jakob: *Implementierung und Vergleich weiterer Zeit-Integrationsverfahren IDP Numerische Aspekte bei Molekulardynamik-Simulationen*. http://www5.in.tum.de/pub/kummerow2010.pdf, 2010. – Accessed: 2013-09-09

[Meg] *MegaMol Project Website*. https://svn.vis.uni-stuttgart.de/trac/megamol/, . – Accessed: 2013-09-09

[NAM] *NAMD Scalable Molecular Dynamics*. http://www.ks.uiuc.edu/Research/namd/, . – Accessed: 2013-09-09

[Ope] *OpenMP API specification for parallel programming*. http://openmp.org/wp/, . – Accessed: 2013-09-09

[PCM] *PCM - Molekulardynamik*. https://downloads.physik.uni-muenchen.de/praktikum/jessen/versuche/PCM.pdf, . – Accessed: 2013-09-09

[Ros] ROSANDI, Y.: *Object-MD 3.0 The Object Oriented Molecular Dynamics Library*. http://merapi.physik.uni-kl.de/objectmd/, . – Accessed: 2013-09-09

[VMD] *VMD Visual Molecular Dynamics*. http://www.ks.uiuc.edu/Research/vmd/, . – Accessed: 2013-09-09

[Zie] ZIESSOW, Dieter: *Zwischenmolekulare Wechselwirkungen Lerneinheit*. http://irrlicht.sourceforge.net/, . – Accessed: 2013-09-09

Abbildungsverzeichnis

2.1 Prinzipieller Ablaufplan einer MD-Simulation 5
2.2 Wirkende Kräfte auf ein Teilchen 7
2.3 2D-Plot des Lennard-Jones-Potentials. Die X-Achse entspricht dem Abstand und die Y-Achse der potentiellen Enegie beider Teilchen 8
2.4 2D-Plot des Gravitations-Potentials 9
2.5 Periodische Randbedingungen 13
2.6 Minimum Image Convention 13
2.7 Nachbarschaftsliste 14

3.1 Abhängigkeiten zwischen den Teilproblemen 19
3.2 Objektorientierter Entwurf mit geeigneten Schnittstellen 20
3.3 Objektorientierter Entwurf mit Particle-Wrapper und geeigneten Templates 20
3.4 Interaktionen der Module, dargestellt durch ein Sequenzdiagramm 21
3.5 TMdSimulator<MdBox> 22
3.6 TMdSimulator<MdBox> 23
3.7 TInteractor<MdBox> 26
3.8 TIntegrator<MdBox> 26
3.9 TInitializer<MdBox> 27
3.10 SRenderAdapter 27

4.1 Boxinitialisierung mit SingleAtomicInitializer, Links: 65 Teilchen, Rechts: 64 Teilchen . 29
4.2 Simulation einer Flüssigkeit mit dem Lennard-Jones-Simulator 32
4.3 Simulation der Planetenbahnen für ein Sonnensystem 33
4.4 Visualisierung einer Lennard-Jones-Simulation durch ein Postprocessing mit VMD . . . 33
4.5 Visualisierung mit MegaMol (links) und IrrMol (rechts) 34