Aufbau eines klassenbasierten Programmpaketes zur Molekulardynamischen Simulation von Gelen am Beispiel des N-Isopropylacrylamid Hydrogels

Dissertation

zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
– Dr. rer. nat. –
der Fakultät für Chemie
an der Universität Bielefeld

vorgelegt
von

Thorsten Tönsing

Bielefeld, im Juni 2004

Gedruckt auf alterungsbeständigem Papier
nach DIN-ISO 9706.

Danksagung

Die vorliegende Dissertation wurde an der Fakultät für Chemie der Universität Bielefeld unter Leitung von Herrn Prof. J. Hinze, PhD angefertigt, dem ich für die Aufnahme in seiner Arbeitsgruppe und für seine Unterstützung danke.
Mein Dank gilt Dr. Christan Oldiges für die fruchtbare Zusammenarbeit, so wie den inspirierenden Diskussion in den letzten Jahren. Danken möchte ich auch Herrn Prof. Dr. W. W. Schoeller und Herrn PD Dr. D. Andrae für ihre Anregungen und Diskussionen.
Den Mitgliedern der Arbeitsgruppe „Theoretische Chemie“ danke ich für das gute und kollegiale Arbeitsklima.
Meinen Eltern danke ich herzlich für ihre fortwährende Unterstützung.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Computersimulationen bieten die Möglichkeit strukturelle, dynamische und thermodynamische Eigenschaften von Molekülsystemen zu untersuchen. Hierzu wurden im wesentlichen zwei Verfahren entwickelt, die Monte Carlo (MC) Simulation und die Molekulardynamische (MD)-Simulation, die beide auf Erkenntnissen der Statistischen Physik aufbauen.

Die zu simulierenden Teilchen werden in einem rechteckigen Raum verteilt. Die dort auf die Teilchen wirkenden Wechselwirkungen lassen sich meist durch Paar-Potentiale beschreiben. Im Monte Carlo Verfahren wird während der Simulation ein Teilchen zufällig ausgewählt und um einen zufälligen Vektor verschoben. Der Metropolis-Algorithmus entscheidet dann, ob die durch diese Verschiebung neu entstandene Konfiguration als neuer Zustand an die Zustandskette angehängt wird, oder ob die Konfiguration verworfen wird. Der letzte Zustand in der Zustandskette bildet immer den Ausgangspunkt für den nächsten Simulationsschritt, in dem wiederum ein zufällig gewähltes Teilchen verschoben wird.

Die Molekulardynamische Simulation folgt der klassischen Mechanik, die Bewegungsgleichungen der Teilchen werden hierbei numerisch integriert, was der natürlichen Vorstellung der Bewegungen der Teilchen näher kommt und hier im weiteren betrachtet werden soll. Durch die Integration der Bewegungsgleichung werden Punkte im Phasenraum erzeugt, die in eine Datei abgespeichert werden und zusammen eine Trajektorie im Phasenraum bilden. Mit Hilfe dieser Trajektorie ist es z. B. möglich, die Bewegung einzelner Moleküle zu verfolgen oder auch die Eigenschaften einer Gruppe von Teilchen zu beschreiben.

Die erste MD-Simulation, eine Flüssigkeit aus harten Kugeln, wurde von Adler und Wainwright 1957 durchgeführt [1, 2]. 1964 folgte dann die erste reale Flüssigkeit, die von Rahman [3] simuliert wurde. Er simulierte flüssiges Argon, wobei die Teilchen über ein Lennard-Jones-Potential miteinander wechselwirkten. Woodcock [4] führte 1971 Ladungen und die Coulomb-Wechselwirkung in die Simulationen ein, und Rahman und Stillinger [5] simulierten Wasser.

Der Lebenszyklus einer MD-Simulationen ist in Abbildung 1.1 dargestellt, er lässt sich grob in vier Phasen aufteilen: Die Erstellung der Kraftfelder, die Kon-

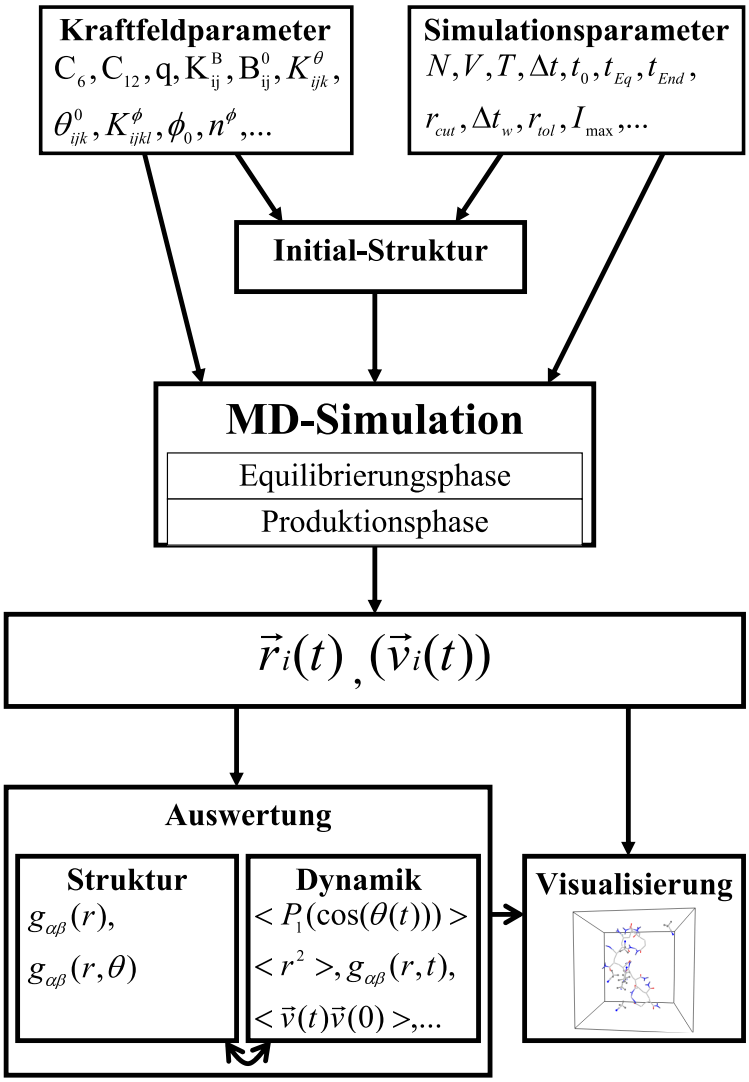


Abbildung 1.1: Die Komponenten eines Simulationssystems.

struktion einer Startstruktur, die Simulation des Systems und die Auswertung der aus der Simulation gewonnenen Daten.

Die Kraftfelder und Kraftfeldparameter können sowohl aus quantenchemischen Berechnungen als auch aus experimentellen Messungen bestimmt werden. Die Bestimmung der Parameter aus quantenchemischen Rechnungen wird zum Beispiel von Case u. a. [6] beschrieben. Die im Rahmen dieser Arbeit angefertigten Simulationen verwenden vorgegebene Kraftfeldparameter, die Teilweise von Berendsen u. a. im GROMOS [7] Parametersatz bestimmt wurden.

Die Kraftfelder lassen sich grob in zwei Klassen aufteilen: Die intramolekularen und die intermolekularen Wechselwirkungen. Während die intramolekularen Wechselwirkungen die Bindungsabstände und Bindungswinkel der Atome eines

Moleküls bestimmen, bestehen die intermolekularen Wechselwirkungen in der Regel aus den van der Waals- und den Coulomb-Wechselwirkungen. Die Kraftfelder sind nicht nur für die Simulation wichtig, sie können auch bei der Konstruktion der Moleküle in der Simulationsumgebung helfen. Die innere Struktur der Moleküle wird hauptsächlich von den intramolekularen Kraftfeldparametern bestimmt. Im Gegensatz dazu werden die Abstände der Moleküle untereinander durch die intermolekularen Parameter beschrieben. Zur Konstruktion einer Simulation werden außerdem weitere Parameter wie das Volumen des Simulationsraumes oder die Atom- bzw. Moleküldichte benötigt.

Zur Erzeugung von Molekülen gibt es verschiedene Methoden. Problematisch ist hier aber die Konstruktion von Polymernetzen, in denen sich die großen Strukturen aus kleineren vernetzen Einheiten, den Monomeren bilden. Zu berücksichtigen ist, dass die Quervernetzer des Netzwerkes mehrere Teilketten mit einander verknüpfen müssen, dabei aber die Abstände der einzelnen Einheiten bis auf kleine Toleranzen festgelegt sind. Für die Konstruktion des NIPAM-Polymergerüstes wurden zwei verschiedenen Verfahren implementiert. Das eine Verfahren verteilt die Atome auf einem virtuellen Gitter mit festen Gitterabstand. Dieser stimmt, um bei der Platzierung der Atome die Bindungsabstände zu erhalten, mit der mittleren Bindungslänge der C-C Bindung überein. Das andere Verfahren ist gegenüber diesem weitaus flexibler. Es baut das Polymernetzwerk frei im Raum auf. Diese Flexibilität erkauft sich das zweite Verfahren durch eine leicht von der mittleren Länge abweichende Bindungslänge an den Verknüpfungspunkten der Teilketten des Polymers. Kleinere Moleküle lassen sich dem gegenüber relativ einfach im Simulationsraum platzieren. Diese werden zufällig platziert, bei einer Kollision des Moleküls mit einem anderen Molekül wird es einfach wieder entfernt und an einer anderen Stelle oder in einer anderen Lage neu platziert.

Die eigentliche MD-Simulation erfolgt in einer zweiten Phase, die wiederum in zwei Teile unterteilt werden kann. Im ersten Teil wird das System bei einer vorgegebenen Temperatur T in ein thermisches Gleichgewicht gebracht. Dies ist notwendig, da nach der Konstruktion des Simulationssystems die Atome nur Lageenergie besitzen, diese wird dann durch die Simulation zum Teil in kinetische Energie umgewandelt, die wiederum teilweise an ein Wärmebad abgegeben wird. Das Wärmebad wird mit Hilfe eines Thermostaten erzeugt, das die Temperatur im Simulationsraum durch Energietransfer an die gewünschte Temperatur angleicht. Bei zu hoher kinetischer Energie werden die Geschwindigkeiten der Atome reduziert, bei zu niedriger Energie erhöht. Hat das System die Soll-Temperatur erreicht, wird es noch einige Zeit weiter simuliert. Die Dauer der zusätzlichen Simulation richtet sich nach der Größe der Moleküle. Hiermit sollen etwaige Spannungen in den Molekülen gelöst werden und die damit frei werdende Energie ebenfalls an das Wärmebad abgegeben werden. Beendet ist die erste Phase der Simulation, wenn sich das System im thermischen Gleichgewicht befindet.

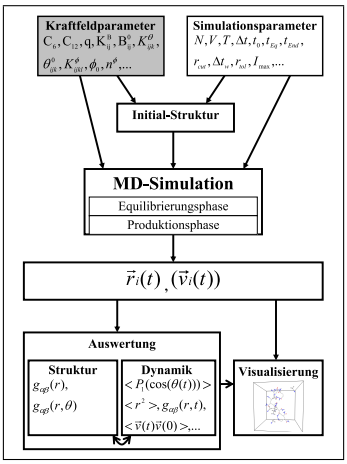
Im zweiten Teil der Simulation werden dann die Simulationsdaten gewonnen. Die Phasenraumpunkte, d. h. die Positionen und Geschwindigkeiten sämtlicher

Atome, werden in festen Abständen abgespeichert. Die Intervalle zur Speicherung reichen dabei von 1 fs zur Analyse von schnellen Bewegungen bis hin zu 50 fs für die Analyse von Trends, wie z. B. der mittleren quadratischen Verschiebung. Während der Datengewinnung kann die Simulation in unterschiedlichen Ensemblen ablaufen. Gebräuchlich sind das mikrokanonische Ensemble (NVE), in dem die Anzahl N der Atome, das Volumen V und die Gesamtenergie E fixiert sind, so wie das kanonische Ensemble (NVT), bei dem statt der extensiven Gesamtenergie E die intensive Temperatur T festgehalten wird, oder das großkanonische Ensemble (NPT), das zusätzlich den Druck P statt des Volumens V konstant hält. Das großkanonische Ensemble ist auf Grund der vernetzten Polymerstruktur und den später beschriebenen periodischen Randbedingungen für die hier durchgeführten Simulationen ungeeignet. Dort würden die Bindungsabstände im Polymer durch die Schwankungen im Volumen zusätzliche Schwankungen ausführen.

Aus den in der Simulation abgespeicherten Phasenraumpunkten lassen sich in der Auswertung strukturelle und dynamische Daten gewinnen, die durch Visualisierungen ergänzt werden können. Die Bestimmung der Struktur basiert im Wesentlichen auf der Auswertung von Paarkorrelationen. Hierbei wird die Aufenthaltswahrscheinlichkeit eines Teilchens in der Umgebung eines Bezugsteilchens berechnet. Aufwendigere Korrelationen beziehen zur genauen Lagebestimmung weitere Teilchen, die in der Regel an dem Bezugsteilchen gebunden sind, in die Berechnungen mit ein. Für die Dynamik eines Teilchens gibt es verschiedene Analyseverfahren, die im allgemeinen jeweils einen Aspekt der Dynamik beschreiben. So beschreibt die Geschwindigkeitsautokorrelation die Verteilung der Geschwindigkeiten im zeitlichen Verlauf. Aus ihr wird durch Fourier-Transformation die Spektraldichte eines Teilchens berechnet, die wiederum Auskunft über die Frequenzen gibt mit denen das Teilchen schwingt. Dem verwandt ist die Rotationsautokorrelation, die sich nicht auf die translativen Geschwindigkeiten bezieht, sondern auf die der Rotation. Einem der Diffusion eines Teilchens entsprechende Analysemethode ist die mittlere quadratische Verschiebung, die in Abschnitt 7.2.3 behandelt wird.

Kapitel 2

Kräfte



Grundlage jeder Simulation bilden die zwischen den Atomen wechselwirkenden Kräfte. Die in den Simulationen verwendeten Kraftmodelle werden in intramolekulare und intermolekulare Kräfte unterteilt. Die intramolekularen Kräfte repräsentieren die Wechselwirkungen der Atome innerhalb eines Moleküls, sie lassen sich über die Anzahl der an der Wechselwirkung beteiligten Atome systematisieren. Diese Wechselwirkungen auf festgelegte atomare Verbände im Molekül lassen sich am einfachsten durch einen harmonischen Oszillator beschreiben, allerdings führt diese Näherung nicht für alle intramolekularen Wechselwirkungen zu akzeptablen Potentialen.

Im Gegensatz dazu wirkt bei den langreichweitigen Wechselwirkungen prinzipiell jedes Atom auf jedes andere Atom, d. h. eine Berechnung zeigt ein $O(N^2)$ verhalten. In einer Simulation entfallen deshalb etwa 90% der Rechenzeit auf die Bestimmung der Energien und Kräfte. Die auf die Atome i und j wirkenden Kräfte werden aus den unterschiedlichen Wechselwirkungspotentialen U bestimmt, die wiederum vom Differenzvektor $\vec{\mathbf{r}}_{ij}$ abhängen:

$$\vec{\mathbf{F}}\left(\vec{\mathbf{r}}_{ij}\right) \;=\; -\nabla U\left(\vec{\mathbf{r}}_{ij}\right) \tag{2.1}$$

Die in die Kraftfeldmodelle eingehenden Parameter basieren auf quantenchemischen Berechnungen für diese Modelle, die zu Parametersätzen führen, die im allgemeinen für einen bestimmten Satz von Atomen bestimmt sind. Die bekanntesten Parametersätze sind das CHARMM (Chemistry at HARvard Macromolecular Mechanics) von Brooks u. a. [8], das MM3 [9, 10, 11] und das MM4 [12, 13, 14, 15, 16] Kraftfeld von Allinger u. a., das AMBER System von Kollman u. a. [6, 17, 18, 19, 20, 21] und das hier verwendete GROMOS System von van Gunsteren u. a. [7]. Weniger verbreitet ist das Universal force field (UFF) von Rappé u. a. [22] und das von Halgrens entwickelte Merck Molekül Kraftfeld [23, 24, 25, 26, 27]. Die hier aufgeführten Parametersätze verwenden die in den kommenden Abschnitten beschriebenen Kraftfeldmodelle. Die Parametersätze bestimmen auch die Handhabung der Wasserstoffatome in den Simulation, wobei zwei Ansätze verfolgt werden. In dem einen Ansatz gibt es eine atomar

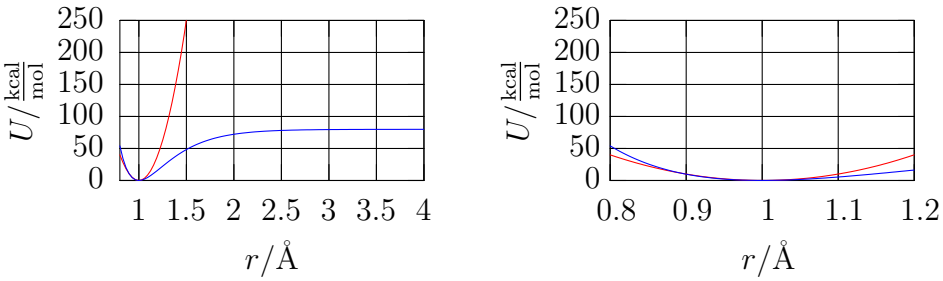


Abbildung 2.1: Der harmonische Oszillator (—) und das Morse-Potential (—).

detaillierte Darstellung, in der jedes Atom explizit in der Simulation erzeugt wird. Der andere Ansatz verwendet, wie auch das benutzte GROMOS System, die sogenannten „united atoms“, hier werden Wasserstoffatome und Kohlenstoffatome als eine atomare Einheit betrachtet, so dass es in den Simulationen neben den Atomen wie H, C, N oder O auch zusammengesetzte Einheiten wie CH,CH_2 oder CH_3 gibt. Vereinfachend werde ich im Folgenden sowohl die einzelnen Atome, als auch die zusammengesetzten Einheiten als „Atome“ betrachten.

2.1 Intramolekulare Wechselwirkungen

Die Ausbildung kovalenter Bindungen, beschrieben durch die Quantenmechanik, erfolgt durch die Überlappung von Elektronenschalen. Diese Überlappungen sind maßgeblich für die Struktur des Moleküls verantwortlich, in klassischen MD-Simulationen werden diese aus komplexen quantenmechanischen Effekten resultierenden Strukturen, durch einfachere Wechselwirkungen der klassischen Mechanik ersetzt.

Die einfachste Wechselwirkung, die sogenannte **Paarwechselwirkung** bestimmt die Bindungslänge r_0 und wird durch eine Streckschwingung beschrieben, sie beschreibt das Verhalten zweier gebundener Atome i und j in Abhängigkeit von ihrem Abstand r_{ij}. Ein Federmodell bzw. der harmonischer Oszillator repräsentieren die hier verwendete Streckschwingung, die durch eine Federkonstante K_b und einen Gleichgewichtsabstand r_0 charakterisiert ist (Abbildung 2.1):

$$U(r_{ij}) = \frac{K_b}{2}(r_{ij} - r_0)^2 \tag{2.2}$$

Die klassischen Wechselwirkungen sind, wie oben bereits erwähnt, nur eine vereinfachte Beschreibung der quantenmechanischen Effekte. Auf Grund der Quantenmechanik repräsentiert ein exponentielles Wechselwirkungspotential die wahren Bindungsverhältnisse besser, deshalb ist das Morse-Potential (Abbildung 2.1) ein weitaus realistischeres Modell als das Federmodell, insbesondere wird das Bindungsverhalten für große Abstände genauer beschrieben. Es ist verglichen mit dem einfachen Federmodell jedoch viel rechenintensiver, weshalb es

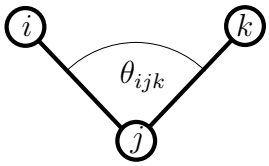


Abbildung 2.2: Bindungswinkel θ_{ijk} zwischen den Atomen i, j und k.

nur selten verwendet wird:

$$U\left(r_{ij}\right) \;=\; E_0\left(\left(1-e^{-k(r_{ij}-r_0)}\right)^2-1\right) \tag{2.3}$$

Einen Kompromiss zwischen Geschwindigkeit und Genauigkeit bildet die Kombination aus dem Federmodell für kleine Abstände und dem Morse-Potential für große Abstände. Sind allerdings nur kleine Elongationen zu erwarten, kann man beim einfachen Federmodell bleiben. In der Anfangsphase der Simulation, der Equilibrierungsphase, in der sehr große kinetische Energien auftreten können, bietet das Federmodell aber den Vorteil, dass festgelegte Bindungen durch große kinetische Energien nicht gebrochen werden können.

Bei der Wechselwirkung zwischen drei Atomen i, j, k eines Moleküls kommt zur Bindungslänge zwischen jeweils zwei Atomen der Bindungswinkel θ_{ijk} hinzu, der den Winkel zwischen den Vektoren $\vec{\mathbf{r}}_{ij}$ und $\vec{\mathbf{r}}_{jk}$ bestimmt (Abbildung 2.2). Weit verbreitet ist hierfür wieder die Beschreibung der Wechselwirkung durch einen harmonischen Oszillator:

$$U(\theta_{ijk}) \;=\; \frac{K_\theta}{2}\left(\theta_{ijk}-\theta_0\right)^2 \tag{2.4}$$

Der Winkel θ_{ijk} wird ursächlich festgelegt durch die ungleichmäßige räumliche Verteilung der Valenzschalenelektronen ($p, sp^3, d, ...$). Der harmonische Oszillator ist in diesem Fall nicht die einzige Methode zur Bestimmung der Biegeschwingung, es gibt zahlreiche Varianten, eine besteht zum Beispiel aus der Kombination des harmonischem Oszillators mit der Kosinusfunktion:

$$U(\theta_{ijk}) \;=\; \frac{k}{2}\big(\cos\left(\theta_{ijk}\right)-\cos\left(\theta_0\right)\big)^2 \tag{2.5}$$

Zur Vervollständigung der Beschreibung der in einem Molekül verwendeten intramolekularen Kräfte fehlen die Wechselwirkungen zwischen vier Atomen. Die Torsionswechselwirkung beschreibt die Verdrehung von vier wie in Abbildung 2.3 verknüpften Atome (i,j,k,l). Die Wechselwirkung wird zum Einen bestimmt durch den Bindungscharakter zwischen den Atomen j und k, eine Doppelbindung an dieser Stelle legt zum Beispiel eine andere Phasenverschiebung ϕ_0 des Torsionswinkels fest, als eine Einfachbindung. Beide Atome legen durch die Struktur ihrer Valenzschalen zusätzlich eine Periodizität m fest. Zur Modellierung dieses Wechselwirkungspotentials wird nur noch selten der Harmonische Oszillator verwendet:

$$U(\phi_{ijkl}) \;=\; \frac{K_\phi}{2}\left(\phi_{ijkl}-\phi_0\right)^2 \tag{2.6}$$

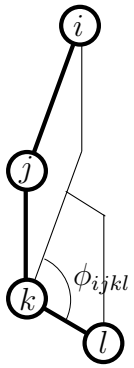


Abbildung 2.3: Torsionswinkel ϕ_{ijkl} zwischen den Ebenen ijk und jkl der Atome i, j, k und l.

Diese Beschreibung der Torsionswechselwirkung ist nur sehr eingeschränkt verwendbar, da bei ihm der periodische Charakter der Torsionswechselwirkung verloren geht. Besser geeignet sind periodische Wechselwirkungsfunktionen wie die Harmonische Kosinusfunktion:

$$U(\phi_{ijkl}) \quad = \quad \frac{K_\phi}{2}\big(\cos(\phi_{ijkl}) - \cos(\phi_0)\big)^2 \tag{2.7}$$

oder das von mir verwendete **Kosinuspotential**:

$$U(\phi_{ijkl}) \quad = \quad K_\phi\big(1 + \cos(m\phi_{ijkl} - \phi_0)\big) \tag{2.8}$$

mit ihrer periodischen Abhängigkeit vom Drehwinkel. In Abbildung 2.4 ist ein entsprechendes Potential mit Periodizität drei dargestellt. Beim Kosinuspotential ist zu beachten, dass abweichend von den beiden anderen Potentialen ein Energieminimum bei $\phi_{ijkl} = 0$ nur dann vorliegt, wenn der Winkel ϕ_0 bei Werten aus dem Intervall $[0; 2\pi]$ den Wert π hat und nicht Null, wie in den anderen Potentialen.

Die Wechselwirkung des uneigentlichen Öffnungswinkels (improper dihedral) ist die letzte nur zwischen den Atomen eines Moleküls wirkende Wechselwirkung die in das simulierte Potential eingeht. Sie beschreibt die Kräfte, die wie in Abbildung 2.5 gezeigt, auf ein Atom (l) wirken, das sich außerhalb einer von den Atomen i,j und k gebildeten Ebene befindet. Zur Beschreibung des hierfür erforderlichen Potentials wird in der Regel wieder ein Federmodell mit Federkonstante K_ϵ und Gleichgewichtswinkel ϵ_0 verwendet:

$$U(\epsilon_{ijkl}) \quad = \quad \frac{1}{2}K_\epsilon\left(\epsilon_{ijkl} - \epsilon_0\right)^2 \tag{2.9}$$

Das Federmodell erzeugt, falls der Winkel ϵ_{ijkl} vom Gleichgewichtswinkel ϵ_0 abweicht, eine rücktreibende Kraft.

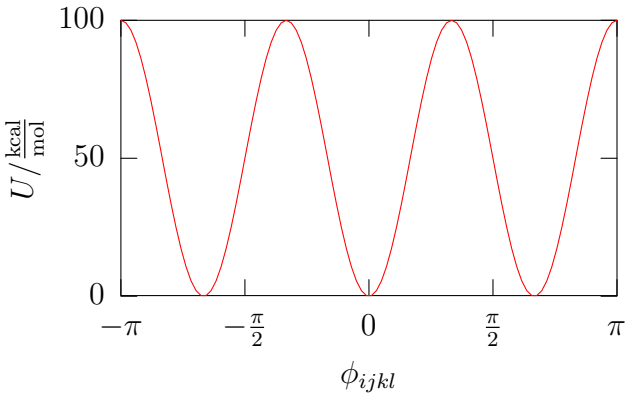


Abbildung 2.4: Kosinuspotential (Gleichung 2.8) zur einfachen Beschreibung der Torsionswechselwirkung ($K_\phi = 50$ $^{\text{kcal}}/_{\text{mol}}$, $m = 3$, $\phi_0 = \pi$).

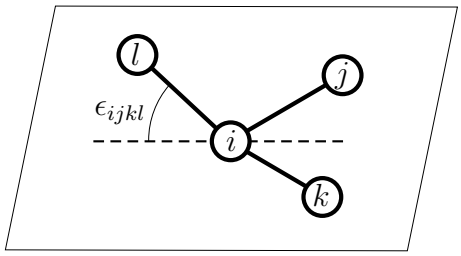


Abbildung 2.5: Der uneigentliche Öffnungswinkel zwischen der Ebene der Atome i,j,k und dem Atom l.

2.2 Intermolekulare Wechselwirkungen

Die intermolekularen Wechselwirkungen sind die Wechselwirkungen der ungebundenen Potentiale, sie werden als Zentral Zweikörperpotentiale modelliert. In einer Simulation werden in der Regel die van der Waals- und die Coulomb-Kräfte verwendet, in besonderen Fällen können auch Metallpotentiale, Mehrkörperpotentiale oder externe Felder berücksichtigt werden, für das langreichweitige Potential U_L gilt somit:

$$U_L\left(r_{ij}\right) = U_{vW}\left(r_{ij}\right) + U_C\left(r_{ij}\right) + \cdots \tag{2.10}$$

Im folgenden beschränke ich mich auf die Beschreibung der van der Waals- und der Coulomb-Wechselwirkungen, da in meinen Simulationen die anderen Potentiale nicht verwendet wurden.

2.2.1 Van der Waals-Wechselwirkung

Das van der Waals-Potential setzt sich aus einem anziehenden und einem abstoßenden Teilpotential zusammen. Das anziehende Teilpotential wird durch Dipol-

momente in den Atomen hervorgerufen, jedoch geht die Wechselwirkung zweier permanenter Dipole zum größten Teil in das Coulomb-Potential ein.

Besitzt, bei der Wechselwirkung von zwei Atomen, nur eines der Beiden ein permanentes Dipolmoment, so influenziert es im Anderen ein entgegengesetztes Dipolmoment, dessen Potential in Abhängigkeit vom Abstand r mit r^{-6} abfällt. Besitzt keiner der beiden Atome ein permanentes Dipolmoment, so entsteht trotzdem eine Wechselwirkung zwischen den Atomen, die London-Dispersionswechselwirkung. Die Elektronen der Atome sind nicht fixiert, deshalb entsteht durch Schwankung ihrer Verteilung ein temporäres Dipolmoment, welches, wie bei einem mit permanenten Dipol, ein entgegengesetztes Dipolmoment im anderen Atom influenziert. Die temporären Dipolmomente erzeugen ebenfalls anziehende Potentiale, die mit r^{-6} abfallen, aber schwächer sind als das Potential eines permanenten Dipols.

Wegen des Pauli-Prinzips dürfen zwei voll besetzte Elektronenschalen nicht überlappen (Überlappungsverbot). Hieraus resultiert ein sehr steiles Potential, welches durch einen r^{-12} oder $e^{-r/a}$ Term modelliert wird. Im van der Waals-Potential werden beide Wechselwirkungen in einem gemeinsamen Potential vereint. Entscheidet man sich bei der Modellierung des Überlappungsverbots für eine r^{-12} Abhängigkeit, so erhält man das Lennard-Jones-Potential:

$$U_{vW}\left(r_{ij}\right) = 4\epsilon_{ij}\left(\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right) \tag{2.11}$$

Die Wechselwirkungsparameter ϵ_{ij} und σ_{ij} für zwei Atome i und j lassen sich dabei nach der Mischungsregel von Lorentz-Berthelot mit einer geometrischen Mittelung von Berthelot und einer arithmetischen Mittelung von Lorentz aus den Wechselwirkungsparametern der reinen Stoffe bestimmen:

$$\epsilon_{ij} = \sqrt{\epsilon_{ii}\epsilon_{jj}} \qquad \sigma_{ij} = \frac{\sigma_{ii} + \sigma_{jj}}{2} \tag{2.12}$$

Eine einfache Umformung des Lennard-Jones-Potentials (2.11) führt zu neuen Parametern A_{ij} und B_{ij}, die in meinen Simulationen verwendet wurden:

$$\begin{aligned} U_{vW}\left(r_{ij}\right) &= \frac{4\epsilon_{ij}\sigma_{ij}^{12}}{r_{ij}^{12}} - \frac{4\epsilon_{ij}\sigma_{ij}^{6}}{r_{ij}^{6}} \\ &= \frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} \end{aligned} \tag{2.13}$$

Die Mischungsregel von Lorentz-Berthelot (2.12) muss für die letzte Gleichung entsprechend angepasst werden:

$$\begin{aligned} A_{ij} &= 4\sqrt{\epsilon_{ii}\epsilon_{jj}}\left(\frac{\sigma_{ii} + \sigma_{jj}}{2}\right)^{12} \\ B_{ij} &= 4\sqrt{\epsilon_{ii}\epsilon_{jj}}\left(\frac{\sigma_{ii} + \sigma_{jj}}{2}\right)^{6} \end{aligned} \tag{2.14}$$

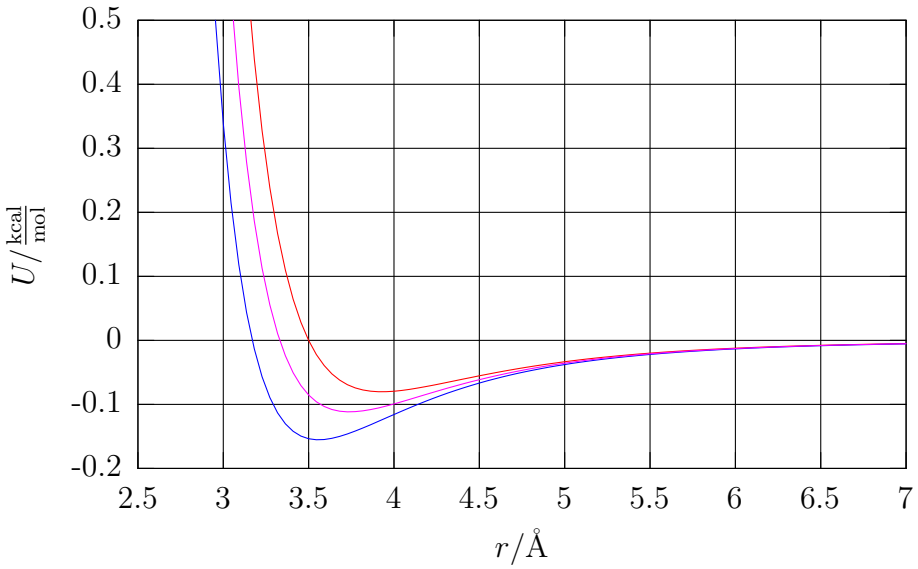


Abbildung 2.6: Lennard-Jones-Potential einer C-C Wechselwirkung (—), einer C-O Wechselwirkung (—) und einer O-O Wechselwirkung (—) am Beispiel des Ethanol Parametersatzes (Abbildung 3.1).

Die Mischungsregel von Jorgensen [28] baut auf die Schreibweise in Gleichung (2.13) auf, die Wechselwirkungsparameter A_{ij} und B_{ij} werden hier aus den Wechselwirkungsparametern A_{ii} und A_{jj} bzw. B_{ii} und B_{jj} der reinen Stoffe in einer geometrischen Mittelung bestimmt:

$$A_{ij} = \sqrt{A_{ii}A_{jj}} \qquad B_{ij} = \sqrt{B_{ii}B_{jj}} \tag{2.15}$$

Die im GROMOS [7] Parametersatz und auch von mir verwendete Mischungsregel von Jorgensen [28] führt zum Beispiel im Ethanol zu den in Abbildung 2.6 gezeigten Potentialverläufen für Kohlenstoff und Sauerstoff.

Eine alternative Mischungsregel wird von Kong [29] beschrieben, er verwendet zur Berechnung des Mischungsterms A_{ij} eine von Jorgensen abweichende Rechenvorschrift:

$$A_{ij} = \frac{A_{ii}}{2^{13}} \left(1 + \left(\frac{A_{jj}}{A_{ii}}\right)^{1/13}\right)^{13} \qquad B_{ij} = \sqrt{B_{ii}B_{jj}} \tag{2.16}$$

Ein Vergleich der Mischungsregel von Lorentz-Berthelot (2.12 / 2.14) mit der Regel von Kong [29] (2.16) und der Regel von Waldman-Hagler [30] wurde im Artikel von Delhommelle und Millie [31] durchgeführt. Angewendet wurden die Mischungsregeln in Simulationen der Edelgase Ne, Ar und Kr, wobei ein Vergleich mit experimentellen Werten ergab, dass das Verfahren von Kong [29] die besten Ergebnisse lieferte. Der Parametersatz von van Gunsteren u. a. [7] wurde für die Mischungsregel von Jorgensen [28] optimiert ist, weshalb letztere in dieser Arbeit verwendet wurde.

Setzt man für die Überlappung eine $e^{-r/a}$ Abhängigkeit an, so erhält man das Buckingham-Potential:

$$U_{vW}\left(r_{ij}\right) = A \exp\left(-\frac{r_{ij}}{\rho}\right) - \frac{C}{r_{ij}^6} \tag{2.17}$$

Die van der Waals-Wechselwirkung berechnet sich nicht aus der Wechselwirkung eines Atoms mit allen anderen Atomen des gesamten Simulationsraumes. Zum Einen werden zu ihrer Berechnung nur die Atome verwendet, die innerhalb einer Kugel vom Radius r_{max} um das betrachtete Atom liegen. Zum Anderen werden dabei die Atome anhand einer Ausschlussliste $\mathrm{Ex}(i)$ ausgeschlossen, die über die intramolekularen Wechselwirkungen eine direkte Bindung mit dem betrachteten Atom eingehen oder mit ihm einen festen Bindungswinkel besitzen. Nicht berücksichtigt werden außerdem die Atome, die über ein Torsionspotential miteinander verbunden sind, für sie existiert ein in Abhängigkeit von den Atomen abgeschwächtes Potential, das in einer separaten Rechnung bestimmt wird. Für die N_{vW} Atome einer Simulation mit positiven van der Waals-Parametern gilt somit:

$$U_{vW} = \sum_{i=1}^{N_{vW}} \sum_{\substack{j<i \\ j \notin \mathrm{Ex}(i)}} \begin{cases} U_{vW}(r_{ij}) & r_{ij} \leq r_{max} \\ 0 & \text{sonst} \end{cases} \tag{2.18}$$

2.2.2 cutoff-Radius

Wie im vorherigen Abschnitt bereits erwähnt, wird die Reichweite der langreichweitigen Wechselwirkungen jeweils durch eine Kugel vom Radius r_{max}, dem cutoff-Radius begrenzt. Für eine Begrenzung der Reichweite sprechen zwei Gründe:

- Die Reichweite wird durch den endlichen Simulationsraum begrenzt.
- Die Rechenzeit wird verkürzt, da die Anzahl der Nachbarn mit dem Abstand quadratisch zu nimmt.

Der cutoff-Radius r_{max} darf hierbei nicht zu klein gewählt werden, da das abgeschnittene Potential nur einen zu vernachlässigenden Anteil am Gesamtpotential besitzen darf. Der Fehler, der durch das Abschneiden des Potentials eingeführt wird, lässt sich aus dem außerhalb des cutoff-Radius r_{max} liegenden Potentials U^+ abschätzen:

$$U^+ = \frac{1}{2} \int_{r_{max}}^{\infty} \mathrm{d}r 4\pi\rho(r)U(r) \tag{2.19}$$

Für das Lennard-Jones-Potential ergibt diese Abschätzung:

$$U^+ = 8\pi\rho\left(\frac{1}{9r_{max}^9} - \frac{1}{3r_{max}^3}\right) \qquad \text{bei} \qquad r_{max} \geq 2.5\sigma \tag{2.20}$$

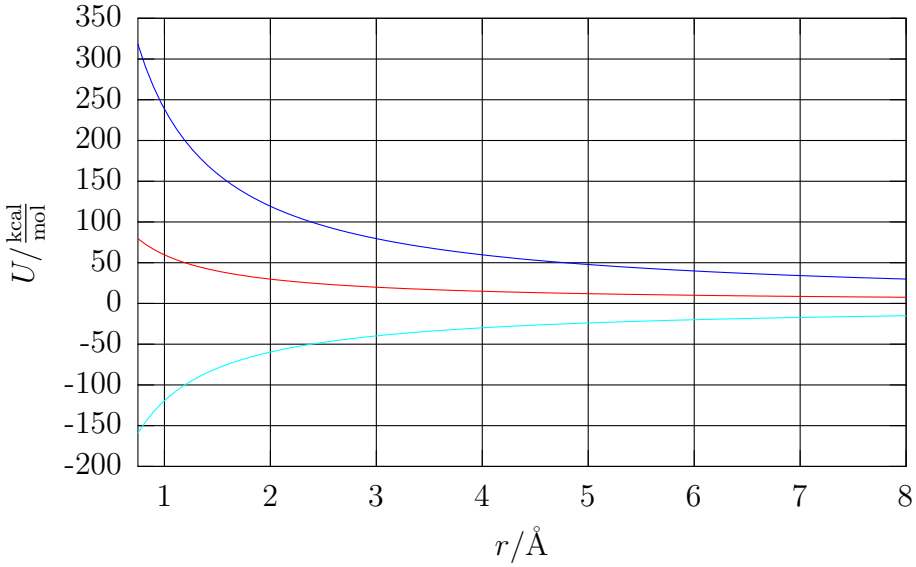


Abbildung 2.7: Coulomb-Potentiale der O-O Wechselwirkung (—), der H-H Wechselwirkung (—) und der O-H Wechselwirkung (—) von Wasser im SPC/E Parametersatz (Abbildung 5.2).

Die Fehlerabschätzung (2.19) zeigt aber auch, dass ein einfacher cutoff-Radius für Potentiale mit einem Abfall bis r^{-3}, wie zum Beispiel dem Coulomb-Potential, ungeeignet ist. Aus diesem Grund wird das Coulomb-Potential im Ewald-Verfahren aufgespalten in einen Realanteil, der durch cutoff-Radius begrenzt ist, und einen reziproken Anteil für die Wechselwirkung mit entfernten Atomen. Beide Bestandteile werden im folgenden Abschnitt ausführlich erläutert.

2.2.3 Elektrostatisches Potential

Die zweite intramolekulare oder auch langreichweitige Wechselwirkung ist das elektrostatische Potential U_c. Die Reichweite der elektrostatischen Wechselwirkung ist um ein vielfaches größer, als die Reichweite der van der Waals-Wechselwirkung. Die van der Waals-Energie hat für große r eine r^{-6} Abhängigkeit, wohingegen die elektrostatische Energie nur mit r^{-1} abnimmt:

$$U_c(r_{ij}) = \frac{1}{4\pi\epsilon_0\epsilon_r}\frac{q_i q_j}{r_{ij}} \tag{2.21}$$

Wie Abbildung 2.7 zeigt, ist das Potential selbst bei größeren Abständen noch relativ hoch, dies zeigt sich insbesondere beim Vergleich mit dem Lennard-Jones-Potential aus Abbildung 2.6. Um dem Rechnung zu tragen, werden bei der Berechnung der elektrostatischen Energie auch weiter entfernte Teilchen, die

eine Ladung tragen, berücksichtigt. Da der maximale Abstand r_{ij} zweier Punktladungen q_i und q_j durch die Länge L des Simulationsraumes ($r_{ij} \leq L/2$) begrenzt ist, erfordert eine Methode, wie die Direkte Summe in Gleichung (2.22), die nur die Teilchen innerhalb des Simulationsraumes berücksichtigt, einen sehr großen Simulationsraum. Trotzdem liefert sie nur eine grobe Näherung der elektrostatischen Wechselwirkungen, da wie oben bereits angedeutet, das Potential auch bei großen Abständen nicht zu vernachlässigen ist.

$$U_c = \frac{1}{4\pi\epsilon_0\epsilon_r} \sum_{i=1}^{N_c} \sum_{\substack{j<i \\ j \notin Ex(i)}} \begin{cases} \frac{q_i q_j}{r_{ij}} & r_{ij} \leq r_{max} \\ 0 & \text{sonst} \end{cases} \tag{2.22}$$

Berücksichtigt man den periodischen Aufbau einer Simulation, hervorgerufen durch die periodischen Randbedingungen, und somit auch die periodische Fortsetzung der Punktladungen außerhalb des zentralen Simulationsraumes, so bietet sich zur Bestimmung der elektrischen Wechselwirkung die aus der Festkörperphysik bekannte Ewald-Kornfeld Summationsmethode an. Diese ist z. B. in Allen und Tildesley [32] oder in Haberlandt u. a. [33] beschrieben. Die Periodizität des Simulationsraumes erzeugt ein unendliches Gitter von identischen Simulationsräumen. Ist $\vec{\mathbf{n}}_i$ der Basisvektor eines jeden Simulationsraumes i, dann lässt sich das elektrostatische Potential $U_c(\vec{\mathbf{r}})$ der N_c Ladungen in jedem Simulationsraum schreiben als:

$$U_c(\vec{\mathbf{r}}) = \sum_i \sum_{j=1}^{N_c} \frac{q_j}{|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j + \vec{\mathbf{n}}_i|} \tag{2.23}$$

$U_c(\vec{\mathbf{r}})$ besteht somit aus dem Potential des Zentralraumes $\vec{\mathbf{n}}_0 = \vec{\mathbf{0}}$ und aller periodischen Fortsetzungen. Die Ewald-Kornfeld Methode führt eine Abschirmfunktion $\gamma(r)$ ein, die für kleine Werte ihres Argumentes gleich Eins ist und für Werte größer als die halbe Länge L des Simulationsraumes auf Null abfällt:

$$\begin{aligned} U_c(\vec{\mathbf{r}}) &= \sum_i \sum_{j=1}^{N_c} \frac{q_j}{|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j + \vec{\mathbf{n}}_i|} \Big(1 - \gamma\big(|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j + \vec{\mathbf{n}}_i|\big) + \gamma\big(|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j + \vec{\mathbf{n}}_i|\big)\Big) \\ &= \sum_i \sum_{j=1}^{N_c} \frac{q_j}{|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j + \vec{\mathbf{n}}_i|} \gamma\big(|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j + \vec{\mathbf{n}}_i|\big) \\ &\quad + \sum_i \sum_{j=1}^{N_c} \frac{q_j}{|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j + \vec{\mathbf{n}}_i|} \Big(1 - \gamma\big(|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j + \vec{\mathbf{n}}_i|\big)\Big) \qquad (2.24) \\ &= \sum_{j=1}^{N_c} \frac{q_j}{|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j|} \gamma\big(|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j|\big) \\ &\quad + \sum_i \sum_{j=1}^{N_c} \frac{q_j}{|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j + \vec{\mathbf{n}}_i|} \Big(1 - \gamma\big(|\vec{\mathbf{r}} - \vec{\mathbf{r}}_j + \vec{\mathbf{n}}_i|\big)\Big) \qquad (2.25) \end{aligned}$$

Auf Grund der Abschirmfunktion $\gamma(r)$ trägt im ersten Term von Gleichung 2.24 nur der Zentralraum zur Summation bei. Der zweite Term der Gleichung erfordert theoretisch eine Summation über sämtliche Wechselwirkungen der periodischen Fortsetzungen des zentralen Simulationsraumes, die durch eine Darstellung als Fourier-Reihe umgangen werden kann. Die Abschirmfunktion selbst wird in der Regel durch die Fehlerfunktion $\mathrm{erf}(x)$ oder genauer durch die Konjugierte Fehlerfunktion $\mathrm{erfc}(x)$ beschrieben, wobei zur Variation der Reichweite ein zusätzlicher Faktor κ eingefügt wird:

$$\gamma(r) = \mathrm{erfc}(\kappa r) = 1 - \mathrm{erf}(\kappa r) = \frac{2}{\sqrt{\pi}} \int_{\kappa r}^{\infty} e^{-y^2} \mathrm{d}y \tag{2.26}$$

Aus dem Wechselwirkungspotential lässt sich nun einfach die Wechselwirkungsenergie bestimmen:

$$E_c = \frac{1}{2} \sum_{i=1}^{N_c} q_i U\left(\vec{\mathbf{r}}_i\right) + E_D \tag{2.27}$$

Der Oberflächenterm E_D wird bestimmt von der angenommenen Oberfläche des makroskopischen Systems. Durch die unendliche periodische Fortsetzung des Simulationsraumes ist dieser in einem unendlichen makroskopischen Körper eingebettet, dessen Oberfläche nur von untergeordneter Bedeutung ist. Geschickter weise darf somit die Oberfläche als Metallhülle angenommen werden, bei dem der Oberflächenterm verschwindet: $E_D = 0$. In der Ewald-Kornfeld-Methode wird die Energie in drei separaten Teilschritten bestimmt:

1. Die Energie E_{cr} bestimmt den reellen Anteil der Ewaldsumme, sie wird aus dem ersten Term der Gleichung 2.25 für die Wechselwirkung aller Atome untereinander berechnet. Von dieser Summation ausgenommen sind alle Wechselwirkungen, die in einer der Ausschlusslisten aufgeführt werden. Für jedes Teilchen i existiert eine solche Ausschlussliste $\mathrm{Ex}(i)$, deren Inhalt bereits am Ende von Abschnitt 2.2.1 beschriebenen wurde.

2. Der komplexe Anteil der Ewaldsumme wird durch Fourier-Transformation berechnet. Zur Bestimmung der Energie E_{ck} ist es nicht notwendig das Potential im ganzen reziproken Raum zu bestimmen. Die Abschirmfunktion $1 - \gamma(r)$ liefert nur für große Distanzen r einen Beitrag, somit müssen im reziproken Raum nur kleine Abstände k berücksichtigt werden. In der Praxis werden die Berechnungen auf einen Bereich innerhalb einer Kugel vom Radius $5 * {}^{2\pi}/_{L}$ begrenzt.

3. Der Selbstwechselwirkungsterm E_{cs} berücksichtigt die in E_{cr} ausgeschlossenen Wechselwirkungen geladener Teilchen im Zentralraum und deren periodische Fortsetzungen, die sonst zu einem Fehler in der Berechnung des reziproken Anteils geführt hätten.

Für die genannten Anteile des Coulomb-Potentials E_c ergeben sich somit folgende Gleichungen:

$$E_{cr} = \frac{1}{4\pi\epsilon_0\epsilon_r}\sum_{i=1}^{N_c}\sum_{\substack{j<i \\ j\notin Ex(i)}}\begin{cases}\frac{q_iq_j}{r_{ij}}\mathrm{erfc}\left(\kappa r_{ij}\right) & r_{ij}\le r_{max}\\ 0 & \text{sonst}\end{cases} \tag{2.28}$$

$$E_{ck} = \frac{1}{4\pi\epsilon_0\epsilon_r}\frac{2\pi}{L^3}\sum_{\vec{\mathbf{k}}\neq\vec{\mathbf{0}}}\frac{\exp\left(-\frac{k^2}{4\kappa^2}\right)}{k^2}\left|\sum_{j=1}^{N_c}q_j\exp\left(-i\vec{\mathbf{k}}\vec{\mathbf{r}}_j\right)\right|^2 \tag{2.29}$$

$$E_{cs} = \frac{1}{4\pi\epsilon_0\epsilon_r}\left(\sum_{i=1}^{N_c}\sum_{j\in Ex(i)}\frac{q_iq_j}{r_{ij}}\mathrm{erf}\left(\kappa r_{ij}\right)+\frac{\kappa}{\sqrt{\pi}}\sum_{i=1}^{N_c}q_i^2\right) \tag{2.30}$$

$$E_c = E_{cr}+E_{ck}-E_{cs} \tag{2.31}$$

Erwähnt sei außerdem, dass die hier verwendete Ewald-Kornfeld-Methode periodische Randbedingungen in allen drei Raumrichtungen voraussetzt. Für die Untersuchung von Oberflächen oder dünnen Schichten müssen spezielle Formen der Ewald-Methode, wie z. B. die Hautman-Klein-Ewald-Methode [34], verwendet werden. Diese setzten nur eine Zweidimensionale Periodizität des Simulationsraumes voraus.

2.3 Kraft-Datenbank

Die Kraft-Datenbank enthält alle Informationen und Parameter über die verwendeten Kräfte eines Simulationssystems. Um eine möglichst große Flexibilität und Übersichtlichkeit zu erhalten, ist die Datenbank in Abschnitte aufgeteilt, die in beliebiger Reihenfolge angeordnet werden können. Kommentarzeilen werden durch ein Semikolon am Zeilenanfang gekennzeichnet.

Die Beschreibung der Kraftfelder erfolgt in fünf Abschnitten, die der Parametrisierung des GROMOS [7] Kraftfelds entsprechen und deren Parameter teilweise in das verwendete Kraftfeld eingegangen sind. Im GROMOS Kraftfeld und somit auch im verwendeten Kraftfeld für Polymere gibt es neben den üblichen Atomen die am Anfang dieses Kapitels eingeführten atomaren Einheiten, bestehend aus Einheiten von CH,CH_2 und CH_3. Zur Verdeutlichung des prinzipiellen Aufbaus der Kraftfelder zeigt Abbildung 2.8 einen entsprechenden Auszug der Kraft-Datenbank. Die Kraftfeldparameter bestehen aus den Parametern, die die Atomsorten charakterisieren, den van der Waals-Parametern und den Parametern für die intramolekularen Wechselwirkungen der gebundenen Atome. Die Atome definierenden Parametern so wie die Parameter der Lennard-Jones-Wechselwirkung und die Parameter einer anderen Wechselwirkungen werden in jeweils eigenen

```
[Nature]
;At Masse   C6    C12[0] C12[1] C12[2] C6trd  C12trd
C   12.0110 23.65  898.0    0      0   23.65   898.0    0      0
N   14.0067 24.13  636.0    0      0   24.13   636.0    0      0
[Bond]
;At Atm     K BO
C   O    1200 1.230
C   N    1000 1.330
C   NT    900 1.330
[Angle]
;At At  At     K Wikel
CH2 N   C    120 122.0
H   N   C     70 123.0
[Dihedral]: X - A1 - A2 - Y
;At Atm  K    Wink Period
C   N    8.00 180  2
C   NT   8.00 180  2
[Improper] Dihedral
;At Atm  K Winkel
C   O   40  0.0
N   H   40  0.0
NT  H   40  0.0
CH1 CH2 80 35.26439
CH1 CH1 80 35.26439
CH1 CH3 80 35.26439
```

Abbildung 2.8: Auszug aus der Kraft-Datenbank der GROMOS Kraftfeldparameter.

Abschnitten der Kraft-Datenbank beschrieben, somit ergeben sich folgende Abschnitte:

Element legt die Eigenschaften eines Atoms fest. In jeder Zeile werden die Daten eines Atoms beschrieben und jeder Spalte wird eine feste Bedeutung zugeordnet. Die ersten beiden Spalten legen den Namen des Atoms und dessen Masse in atomaren Massen-Einheiten fest, die restlichen Spalten bestimmen die van der Waals-Parameter. Spalte drei und vier legen $\sqrt{B_{ii}}$ und $\sqrt{A_{ii}}$ für die Lennard-Jones-Wechselwirkung mit anderen Atomen mit Ausnahme des dritten Nachbarn fest. Die Werte für den dritten Nachbarn liegen in den Spalten sieben und acht. Zur Zeit ungenutzt sind die Spalten fünf, sechs, neun und zehn, sie sollen später Werte zur Differenzierung des A_{ii} Parameters enthalten. Zu beachten ist, dass die A_{ii} (in $^{\text{kcal}}/_{\text{mol Å}^6}$) und die B_{ii} (in $^{\text{kcal}}/_{\text{mol Å}^{12}}$) angegeben werden.

Bond bestimmt die Parameter der Streckschwingung, hierzu enthalten die ersten beiden Spalten die Namen der wechselwirkenden Atome. Spalte drei legt die Federkonstante K_b (in $^{\text{kcal}}/_{\text{mol Å}^2}$) fest, während die letzte Spalte den Gleichgewichtsabstand r_0 in Ångström enthält.

Angle listet die Parameter der Biegeschwingung auf. Die Biegeschwingung wird wie die Streckschwingung durch einen harmonischen Oszillator modelliert.

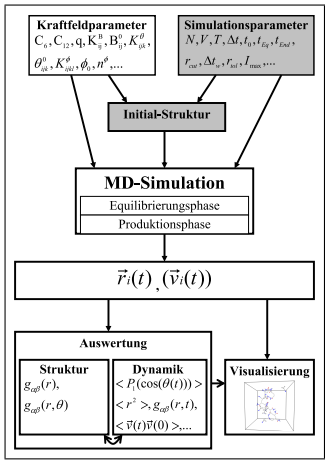
Die Namen der wechselwirkenden Atome ijk werden in den ersten drei Spalten eingetragen, die beiden letzten Spalten enthalten die Federkonstante K_θ in $^{\text{kcal}}/_{\text{mol rad}^2}$ und den Gleichgewichtswinkel θ_{ijk} in Grad.

Dihedral enthält die für den Torsionswinkel wichtigen Parameter. Zur Festlegung eines Torsionswinkels sind, wie Abbildung 2.3 zeigt, vier Atome $ijkl$ notwendig. Bestimmend für die Parameter sind dabei nur die mittleren beiden Atome j und k. Somit reicht für eine eindeutige Charakterisierung der Parameter die Angabe zweier Atome in den ersten beiden Spalten. Die Torsionsformel (2.8) verlangt drei bestimmende Parameter, einen das Potential skalierenden Vorfaktor K_ϕ, der in $^{\text{kcal}}/_{\text{mol}}$ angegeben wird, die Phasenverschiebung ϕ_0 in Grad und die Periodizität m der Torsion.

Improper bestimmt die Parameter für die Auslenkung eines Atoms i aus der Ebene der Atome j, k und l wie es Abbildung 2.5 zeigt. Zur Charakterisierung der Wechselwirkungsparameter sind, wie beim Torsionswinkel, nur zwei der vier wechselwirkenden Atome nötig, nämlich das zentrale Atom j der Ebene und eines der drei verbleibenden Atome. Die Wechselwirkung wird durch ein Federmodell beschrieben, d. h. die Wechselwirkungsparameter bestehen aus der Federkonstanten K_ϵ, deren Wert in der Einheit $^{\text{kcal}}/_{\text{mol rad}^2}$ angegeben wird, und dem Gleichgewichtswinkel ϵ_0 in Grad.

Kapitel 3

Die Startstruktur



Für eine Simulation wird ein Phasenraumpunkt als Startstruktur benötigt, mit dem die Simulation beginnt. Dieser Phasenraumpunkt kann aus vorhergehenden Simulationen stammen oder aus der Lage der Atome aus Röntgenstrukturanalysen gewonnen werden. Eine weitere Möglichkeit eine Startstruktur zu erzeugen, besteht darin, die Koordinaten der Atome zufällig zu wählen, wobei die aus den Wechselwirkungen rührenden Randbedingungen berücksichtigt werden.

In diesem Kapitel soll die letzte Variante betrachtet werden, da sie mit den in der Simulation ohnehin notwendigen Parametern auskommt. Zur Bestimmung der Startstruktur müssen als erstes die Eigenschaften des Simulationsraumes festgelegt werden, anschließend können die Moleküle im Simulationsraum verteilt werden.

Zur Erzeugung der Polymermoleküle stehen zwei Verfahren zur Verfügung, die sich in Flexibilität und in der Güte der erzeugten Bindungsabstände unterscheiden. Das erste Verfahren ist sehr starr und kann nur spezielle im Programmcode implementierte Polymere generieren. Dem gegenüber ist das zweite Verfahren flexibel und nicht auf spezielle Polymere beschränkt, benötigt aber zu ihrer Erzeugung zusätzliche die Struktur beschreibende Daten aus einer speziellen Kraft- / Struktur-Datenbank.

3.1 Der Simulationsraum

Alle Molekulardynamischen Simulationen sind auf ein begrenztes Volumen beschränkt, dieses bildet den Simulationsraum. Der Simulationsraum enthält alle zu simulierenden Atome und begrenzt gleichzeitig deren Aufenthaltsort. In meinen Simulationen habe ich einen kubischen Simulationsraum verwendet, dessen Kantenlänge L in der Struktur-Datenbank durch die Variable **BoxSize** vorgegeben wird. Die Wahl eines rechteckigen Simulationsraums, dessen Kanten auf den Achsen eines kartesischen Koordinatensystems liegen, erleichtert wesentlich die unten beschriebene Einhaltung und Bestimmung der periodischen Randbe-

dingungen.

Simuliert wird immer ein möglichst kleines Volumen $V = L^3$, da die Anzahl N der zu simulierenden Atome linear mit dem Volumen V steigt, die Anzahl der Wechselwirkungen aber mit N^2. Ein kleines Volumen V birgt aber das Problem der großen Oberfläche O und somit starker Oberflächeneffekte. Bei einem kubischen Simulationsraum ändert sich die Oberfläche O mit dem Volumen V wie folgt:

$$O = 6V^{2/3} \tag{3.1}$$

Um die starken Oberflächeneffekte zu verdeutlichen, betrachte man den Simulationsraum als kubisches Gitter mit 1000 Knoten auf denen die zu simulierenden Teilchen liegen. Von diesen Teilchen befinden sich dann 488 auf der Oberfläche, das sind fast 50%. Damit die Simulation nicht fast nur Oberflächeneffekte beschreibt, sollte das endliche zu simulierende System in einem unendlichem Medium eingebettet werden, dies ist aber nicht realisierbar. Ersatzweise führt man eine periodische Fortsetzungen des Simulationsraumes ein, hierbei werden vom Simulationsraum fiktive, identische Kopien erzeugt. Die Kopien werden um den Simulationsraum an dessen Kanten angelegt. An diesen Ring werden dann weitere Kopien angelegt, so dass schließlich ein fiktives unendliches System entsteht, das den Simulationsraum umschließt und aus periodischen Abbildungen eines kleinen, endlichen Volumens besteht.

Praktisch bedeutet dies für die Simulation, dass Atome, die durch eine Wand des Simulationsraum hinaus wandern in die benachbarte Kopie hinein wandern. Da aber Original und periodische Kopie immer identisch sein müssen, wandert das Atom gleichzeitig auf der gegenüber liegenden Wandseite wieder in den Simulationsraum hinein.

Ebenso wie die Atome unterliegen auch die Wechselwirkungen diesen periodischen Randbedingungen, da dessen Stärke durch den kürzesten Abstand der zwei wechselwirkenden Atome bestimmt wird. Solange der Abstand der Atome kleiner als $L/2$ ist, ist die Berechnung unproblematisch. In diesem Fall ist der Abstand im Simulationsraum der kürzeste Abstand, im anderen Fall ist der kürzeste Atomabstand der Abstand zur nächstgelegenen periodischen Abbildung des zweiten Atoms. Um Artefakte durch Selbstwechselwirkung zu vermeiden, muss die Kantenlänge L des Simulationsraumes mindestens doppelt so groß gewählt werden wie der cutoff-Radius der van der Walls Wechselwirkung.

Die sehr viel weiter reichenden Coulomb-Wechselwirkungen müssen anders behandelt werden. Neben den Atomen im Simulationsraum müssen dort auch die Atome in den periodischen Abbildungen berücksichtigt werden, wie dies zum Beispiel in der im Abschnitt 2.2.3 eingeführten Ewald-Kornfeld-Methode erfolgt. Diese teilt die Berechnung des Potentials und der Kräfte in einen inneren und einen äußeren Bereiche des Raums um das betrachtete Teilchen auf. Der bei der Ewald-Kornfeld-Methode verwendete cutoff-Radius des Realanteils sollte jedoch, wie im van der Waals Fall, kleiner als die halbe Kantenlänge L sein.

3.2 Strukturbildung im Gitter

Programme mit festen Konstruktionsplänen liefern spezifische Lösungen zur Bildung von Startstrukturen, diese können nur Strukturen für bestimmte Polymere bzw. Moleküle erstellen. Ein entsprechendes Programm für PAA-Gele wurde von Paulo Netz [35] geschrieben, dies bildete die Grundlage für mein erstes Programm zur Erzeugung von P-NIPAM-Gel Strukturen. Unterschiedliche Polymere lassen sich in meinen beiden Programmen durch eine Variation der Größe des Simulationsraums, dem Quervernetzeranteil, oder durch die, den Aufbau der Polymerketten bestimmenden, minimalen und maximalen Abstände der Quervernetzer, so wie dem Faktor p für die Wahrscheinlichkeit eines „random walk“ im Aufbau der Polymerketten erzeugen.

Für die Simulation muss ein Polymernetzwerk erzeugt werden, welches aus Polymerketten besteht, die durch Quervernetzermonomere miteinander verknüpft werden. Hierbei müssen die Ketten so angelegt werden, dass sie mit den Quervernetzern passgenau verknüpft werden können. Der erste Ansatz bestand darin, das Polymer auf einem künstlichen Gitter zu erzeugen. Die Atome der Monomere werden hierzu auf freien Gitterplätzen fixiert. Bei Netzwerken mit einem Polymerrücken, der im wesentlichen aus einfachen Kohlenstoffeinheiten aufgebaut ist, wie dies zum Beispiel beim NIPAM und beim PAA der Fall ist, eignete sich hierzu besonders das Diamantgitter. In einem Diamantgitter bleibt die Tetragonale Struktur der Kohlenstoffatome und damit die Kettenstruktur am besten erhalten. Auf Grund der festen Struktur des Diamantgitters werden jedoch sowohl beim NIPAM als auch beim PAA die Seitenketten der Monomere, so wie die Stickstoffbindungen des Bisacrylamid Quervernetzers, verzerrt erzeugt.

Der Aufbau der NIPAM Polymerstruktur beginnt immer mit der Verteilung des Bisacrylamid Quervernetzers auf dem Diamantgitter, wobei diese zuvor definierte Mindestabstände zueinander einhalten müssen. Die einzelnen Atome des Quervernetzers werden so auf die Gitterplätze verteilt, dass gebundene Atome auf benachbarten Gitterplätzen liegen und so die Bindungsstruktur erhalten bleibt. Eine Doppelbelegung der Gitterpunkte ist nicht zulässig, in einem solchen Fall muss eines der Monomere der kollidierenden Atome entfernt und an einer anderen Stelle neu aufgebaut werden.

Jedes Bisacrylamid besitzt vier offene Bindungen, die durch NIPAM-Monomerketten mit anderen Quervernetzern verknüpft werden müssen. Zwei zufällig ausgewählte offene Bindungen bilden jeweils ein Paar, zwischen dessen Enden mittels eines „directed random walk“ eine Monomerkette gespannt wird. Die Monomerkette wird an einem Ende begonnen, von dem aus dann durch die Platzierung weitere Monomere auf benachbarten Gitterplätzen eine Polymerkette gebildet wird. Diese Monomerkette wächst dabei mit einer vorgegebenen Wahrscheinlichkeit p in Richtung der anderen offenen Bindung. Dies hat zur Folge, dass die Länge der Kette außer vom Quervernetzerabstand nur noch von der Wahrscheinlichkeit p bestimmt wird. Schließt die Kette am Ende des Aufbaus

nicht genau mit der zweiten offenen Bindung ab, d. h. im Fall des PAA und des NIPAM, bei denen zwei Kohlenwasserstoffeinheiten den Polymerrücken einer Monomereinheit bilden, liegt noch ein Gitterpunkt zwischen offener Bindung und Kette, wird diese Lücke durch eine zusätzliche Kohlenwasserstoffeinheit geschlossen, ansonsten werden die Enden direkt verbunden.

Nachdem das Polymernetzwerk erzeugt worden ist, können weitere Moleküle in den Simulationsraum durch entsprechende Routinen im Programm eingefügt werden. Eine entsprechende Routine für Acetonitril wurde von Christian Oldiges entwickelt und verwendet. Abschließend muss zur Erzeugung eines Hydrogels der frei gebliebene Raum mit Wasser gefüllt werden. Die Wassermoleküle werden in diesem Programm nicht selbst erzeugt, sondern aus einer PDB-Datei mit Wasserkoordinaten entnommen, dies schränkt die Größe des Simulationsraumes auf Systeme bis zur Größe des von den Wassermolekülen benutzten Raumes ein. Die Wassermoleküle werden, falls an der jeweiligen Stelle im Simulationsraum kein anderes Atom liegt, an der in der PDB-Datei angegebenen Stelle im Simulationsraum platziert. Die so erzeugte Struktur bildet den Ausgangspunkt für die Equilibrierungssimulation.

3.3 Struktur-Datenbank

Die Struktur-Datenbank enthält alle Informationen und Parameter zur Erzeugung eines Simulationssystems mit dem Programm „StructC.“ Um eine möglichst große Flexibilität und Übersichtlichkeit zu erhalten ist die Datenbank, wie die bereits in Abschnitt 2.3 beschriebene Kraft-Datenbank, in Abschnitte aufgeteilt. Damit jedes Molekül eine abgeschlossene Einheit bildet, können alle Abschnitte aus der Kraft-Datenbank und die unten beschriebenen Bindungsstrukturabschnitte beliebig oft wiederholt und kombiniert werden. Lediglich die Abschnitte „Structure“ und „Create“ dürfen nur einmal in der Struktur-Datenbank vorhanden sein, da mehrere dieser Abschnitte unterschiedliche Strukturen generieren würden.

Die Struktur-Datenbank muss unter anderem die Bindungsstruktur der verwendeten Moleküle enthalten, welche für jedes Molekül in einem separaten Abschnitt („Molecule“, „Molecule1“, „Molecule2“, ...) beschrieben wird. Jedem Molekül muss hierzu, für die spätere Verwendung im eigentlichen Generierungsabschnitt, ein eindeutiger Name zugeordnet werden. Die Beschreibung der Struktur erfolgt dann in einem einer Z-Matrix ähnlichem Format. Die erste Spalte der Matrix ist optional und enthält die Atomnummer im Molekül. Die Atomnummer dient ausschließlich zur besseren Lesbarkeit der Strukturbeschreibung und hat keinen Einfluss auf die Generierung. Falls Nummern vergeben werden, müssen die Atome der Reihe nach durch nummeriert werden. Der weitere Aufbau der Matrix lässt sich am einfachsten an einem Beispiel verdeutlichen.

```
; from Molec. Sim. 18, 133 (1996)
[Molecule]
Name=Ethanol
;      H2 H2
; H1-O-C1-C2-H2
;      H2 H2

;[N] Nam  q  [R]    Atm [Ang] Atm Per/Ang Atm (vgl. Z-Matrix)
;negative AtomNr. Verknuepfung mit anderem Teil-Molek.
01 C2E  0.3
02 C1E  0.0 1.53    1
03 OE  -0.7 1.431   2  107.8  1
04 H1E  0.4 0.97    3  105.0  2    1      1
05 H2E  0.0 1.1     1  110.0  2    0      3
06 H2E  0.0 1.1     1  110.0  2    1      3
07 H2E  0.0 1.1     1  110.0  2    2      3
08 H2E  0.0 1.1     2  110.0  1    1      5
09 H2E  0.0 1.1     2  110.0  1    *      5

[Element]
;Atm Masse    C6    C12[0] C12[1] C12[2] C6trd  C12trd
H1E  1.00787  0       0     0      0      0       0     0      0
OE  15.99949 25.11  799.9   0      0      0       0     0      0
C1E 12       24.30 1041.9   0      0      0       0     0      0
C2E 12       24.30 1041.9   0      0      0       0     0      0
H2E  1.00787  7.61  129.1   0      0      0       0     0      0

[Bond]
;Atm Atm  K B0
H1E OE    0 0.97
C1E OE    0 1.431
C1E C2E   0 1.53
C1E H2E   0 1.1
C2E H2E   0 1.1
H2E H2E   1 1

[Angle]
;Atm Atm Atm   K Wikel
H1E OE  C1E  320 105
OE  C1E C2E  460 107.8
OE  C1E H2E  350 108
C1E C2E H2E  367 110
C2E C1E H2E  367 110
H2E C1E H2E  306 108
H2E C2E H2E  306 108

[Dihedral]; X - A1 - A2 - Y
;Atm2 Atm3   K    Wink Period
OE    C1E    6      0  3
C1E   C2E   11.5    0  3
```

Abbildung 3.1: Kraft- / Struktur-Datenbank von Ethanol.

Abbildung 3.1 zeigt eine Ethanol Kraft- / Struktur-Datenbank mit allen optionalen Parametern und den zugehörigen Kraftfeldern, wobei die von Müller-Plathe [36] bestimmten Kraftfeldparameter verwendet wurden. Ein aus dieser Struktur-Datenbank erzeugtes Molekül ist in Abbildung 3.2 zu sehen.

Der Aufbau des Moleküls erfolgt in der Reihenfolge, in der die Atome in der Liste angegeben werden. Die zweite und dritte Spalte der Matrix spezifizieren das neue Atom. Die zweite Spalte enthält die Bezeichnung des Atoms, wie sie im Ab-

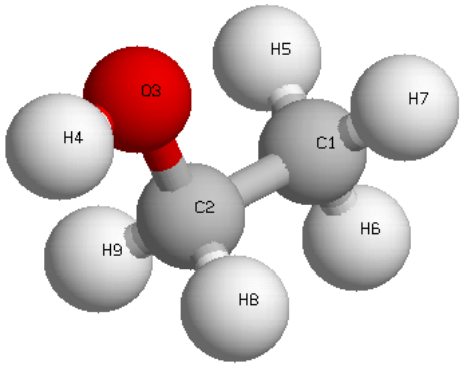


Abbildung 3.2: Darstellung der Ethanolstruktur.

schnitt „Element“ definiert wurde, gefolgt von der Teilchenladung als vielfaches der Elementarladung. Das erste Atom des Ethanols steht somit nach Zeile 10 in Abbildung 3.1 unter der Bezeichnung „C2E“ in der Kraftfeldliste und besitzt eine Ladung von 0.3 e. Es ist durch diese Daten ausreichend charakterisiert und kann im Simulationsraum erzeugt werden.

Jedes weitere Atom i benötigt zur Bestimmung der relativen Lage den Abstand r_{ij} zum Atom j an dem es bindet. Hierzu besitzt die Matrix ab dem zweiten Atom zwei weitere Spalten, von denen die erste optional ist und die Bindungslänge r_{ij} in Å enthält. Der Wert wird als Dezimalwert mit Dezimalpunkt angegeben, damit dieser nicht vom Programm mit dem Wert der nächsten Spalte verwechselt wird. Wird keine Bindungslänge definiert, so wird der in den Kraftfeldern definierte Wert verwendet. Für das Ethanol ist hier ein Wert von 1.53 Å angegeben, der in diesem Fall mit dem Kraftfeldparameter (Zeile 32) überein stimmt. Die zweite neue Spalte enthält die Nummer des Atoms j an den das neue Atom gebunden werden soll. Im Fall des zweiten Atoms ist dies immer die Nummer des ersten Atoms, also die Eins.

Alle weiteren Atome i (in dieser Ethanolstruktur ab dem Sauerstoff, „OE“ Zeile 12) bilden zusätzlich mit dem direkt bindenden j und dem übernächsten Atom k, das in der siebten Spalte angegeben werden muss, einen Bindungswinkel θ_{ijk}, der als weiterer optionaler Parameter mit Dezimalpunkt in Spalte sechs angegeben werden kann.

Sind mindestens drei Atome i, j, k vorhanden, so bilden diese in der Regel eine Ebene ijk. Die Atome j und k spannen zusammen mit dem neuen Atom l ebenfalls eine Ebene jkl auf, dessen Winkel zur Ebene ijk vom Torsionswinkel festgelegt wird. Die Atomnummern der Atome j, k, l werden in den Spalten fünf, sieben und neun eingetragen. Da die Kraftfeldparameter den Torsionswinkel nicht eindeutig definieren, muss in Spalte acht ein weiterer Parameter angegeben werden. Wird dort ein Dezimalwert (mit Dezimalpunkt) eingetragen, so wird dieser Wert als Torsionswinkel in Grad interpretiert. Wird statt dessen ein ganzzahliger Wert (ohne Dezimalpunkt) verwendet, wie beim Ethanol für die Atome 4

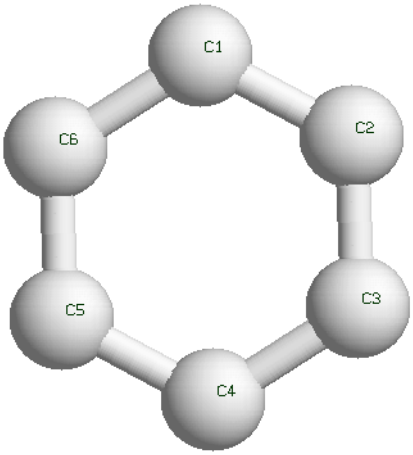


Abbildung 3.3: Darstellung der Benzolstruktur.

```
[Molecule]
Name=Benzol
;[No]  Nam    q  [R]    Atm [Ang] Atm Per/Ang Atm (vgl. Z-Matrix)
   1   CR6  0.0
   2   CR6  0.0 1.38     1
   3   CR6  0.0 1.38     2  120.0  1
   4   CR6  0.0          3  120.0  2    0.0    1
   5   CR6  0.0          4         3    0.0    2
       CR6  0.0 1.38     5  120.0  4    0.0    3     1
```

Abbildung 3.4: Struktur-Datenbank der Benzolringstruktur.

bis 8, gibt dieser das verwendete Minimum in der Torsionsenergie an (vgl. Abbildung 2.4). Soll das Energieminimum zufällig ausgewählt werden, wird wie bei Atom 9 in Zeile 18 ein Stern (*) in die Spalte eingetragen. Als weitere Alternative kann durch Angabe eines „M“ das Energieminimum ausgewählt werden, das die geringste van der Waals- und Coulomb-Wechselwirkung besitzt. Diese Variante wird unter anderem bei der Erzeugung des NIPAM-Polymers verwendet, dessen Struktur-Datenbank Abbildung 3.5 zeigt.

Mit dieser Matrixschreibweise lassen sich beliebige lineare Strukturen erzeugen. Es ist aber nicht möglich geschlossene Strukturen, wie zum Beispiel einen Benzolring zu erzeugen, es fehlt die Möglichkeit Bindungen mit mehreren Atomen anzugeben. Um dieses Manko zu beheben, besteht die Möglichkeit ab der zehnten Spalte die Atomnummern anzugeben, zu denen ein Atom weitere Bindungen besitzt. Die Konstruktion solcher geschlossener Systeme erfordert eine genaue Planung, so wie eine explizite Festlegung der Energieminima der Torsionsenergie. Es muss sichergestellt werden, dass die Abstände zu den ab Spalte zehn eingetragenen Atomen gerade den Bindungsabständen entspricht. Eine einfache geschlossene Struktur ist der in Abbildung 3.3 gezeigte Benzolring, der aus der Struktur-Datenbank in Abbildung 3.4 erzeugt wurde.

Polymere bestehen aus einer Verkettung von Monomeren. Die Strukturmatrix eines einzelnen Monomers ähnelt der Matrix eines einfachen Moleküls. Im Unterschied zu diesem muss ein Monomer aber mit anderen Monomeren Bindungen eingehen und sich in eine bestehende Struktur einfügen. Es kann somit nicht, wie ein Molekül, unabhängig von der Lage bereits erzeugter Strukturen erstellt werden.

Da die bestehende Teilstruktur des Polymers beim Einfügen eines weiteren Monomers berücksichtigt werden muss, sind bei den ersten drei Atomen des Monomers die Bindungs-, Biege- und Torsionswechselwirkungen mit Atomen des vorhergehenden Monomers zu berücksichtigen. In der Bindungsmatrix müssen somit auch für die ersten drei Atome alle Bindungswechselwirkungen, d. h. die Wechselwirkungen mit den Atomen des vorherigen Monomers, angeben werden. Abbildung 3.5 zeigt dies für die BIS und NIPAM Monomere. Auf der anderen Seite müssen auch die Bindungspunkte festgelegt werden, an denen andere Monomere angefügt werden können. Die Festlegung dieser Bindungspunkte erfolgt einfach über Platzhalter, die durch einen Stern (*) gekennzeichnet sind, zu sehen zum Beispiel in den Zeilen 26 bis 29 und 53 von Abbildung 3.5.

Aus den grundlegenden Kraftfeldparametern und den Strukturdaten der Moleküle ist es somit möglich die Startstruktur einer Simulation zu generieren. Der Bau einzelner Moleküle ist mit den oben angegebenen Beschreibungen der Strukturen möglich, erzeugen möchte man jedoch in der Regel ein Simulationssystem mit mehreren Molekülen und Polymerketten. Hierzu müssen Rahmendaten der Simulationen im Abschnitt „Structure“ festgelegt werden, dieser ist für eine NIPAM Simulation im ersten Teil der Struktur-Datenbank von Abbildung 3.6 zu finden. Dort wird die Kantenlänge des Simulationsraumes (**BoxSize**) und der Toleranzabstandsfaktor (**crwTol**) festgelegt, der zusammen mit der Bindungslänge den Abstand bestimmt, den zwei Teilketten unterschreiten müssen, um eine Bindung eingehen zu können. Die Reproduzierbarkeit der generierten Struktur ist gewährleistet bei Vorgabe eines Startwertes (**Seed**) für den Zufallszahlengenerator, mit dessen Hilfe eine pseudo zufällige Startstruktur erzeugt wird. In diesem Abschnitt der Datenbank werden außerdem die einzubindenden anderen Kraft-Datenbank und Struktur-Datenbank-Dateien (durch das Schlüsselwort **include**), so wie die unterschiedlichen Ausgabe-Dateien aufgeführt. Fünf verschiedene Ausgabeformate sind implementiert. Die beiden wichtigsten Formate sind das PDB-Format, das die Atomkoordinaten als Protein-Data-Base ausgibt, und das DYN-Format, welches die Startdaten für die Simulation liefert. Das PDB-Format ist in soweit interessant, als man mit dem Programm „RasMol“ von Sayle u.a. [37] bzw. der Windows Erweiterung „RasTop“ von Valadon die erzeugte Simulationsstruktur visualisieren kann. Die anderen Formate dienen lediglich der einfachen Überprüfung der erzeugten Struktur.

Die Anweisungen zur Generierung der Moleküle befinden sich im Abschnitt „Create“, dort wird auch festgelegt, wie die Quervernetzer- und Ketten-Moleküle die Polymere bilden. Die hierzu verfügbaren Kommandos werden in Abbildung 3.6

```
1  [Molecule0]
   Name=Bisacrylamid
   ;  (-1)
   ;   |
5  ;  (0*) 5   9   (*)
   ;   |   |   |    |
   ;   1   4-6-8   13
   ;   |  /     \   |
   ;   2-3      10-12
10 ;   |  \     /   |
   ;  (*)  7  11   (*)

   01 CH2  0     0 -1 m -2
   02 CH1  0     1  0 m -1
15 03 C    0.38  2  1 m  0
   04 N   -0.28  3  2 m  1
   05 H    0.28  4  3 m  2
   06 CH2  0     4  3 m  2
   07 O   -0.38  3  4 m  6
20 08 N   -0.28  6  4 m  3
   09 H    0.28  8  6 m  4
   10 C    0.38  8  6 m  4
   11 O   -0.38 10  8 m  6
   12 CH1  0    10  8 m  6
25 13 CH2  0    12 10 m  8
   *    0     1
   *    0    13
   *    0    12
   *    0     2
30
   [Molecule1]
   Name=NIPAM
   ; (*)-2-1-(0)-(-1)-(-2)
   ;       |
35 ;       3
   ;      / \
   ;     5   4
   ;    / \
   ;   6   9
40 ;  / \
   ; 7   8

   ;Atom Charge Connect
   1  CH1  0.00  0 -1 M -2
45 2  CH2  0.00  1  0 m -1
   3  C    0.38  1  0 m -1
   4  O   -0.38  3  1 m 2
   5  N   -0.28  3  1 m 2
   6  CH1  0.00  5  3 m 1
50 7  CH3  0.00  6  5 m 3
   8  CH3  0.00  6  5 m 3
   9  H    0.28  5  3 m 1
   10 *    0.00  2
```

Abbildung 3.5: Struktur-Datenbank des NIPAM-Polymers.

ab Zeile 18 für die Erzeugung einer P-NIPAM Simulation verwendet, im folgenden werden sie auch an diesem Beispiel erläutert.

Die Erstellung der Moleküle sollte nach Molekülgröße erfolgen, d. h. das Polymer wird immer als erstes erzeugt. Begonnen wird in diesem Fall mit einem Quervernetzermonomer, dem Bisacrylamid (BIS), zu sehen in Abbildung 3.7a.

```
1  [Structure]
   BoxSize=25.0
   Constraint 2e-5 3500
   crwTol 0.25
5  debug=5
   Seed=23904611
   ;include acetonitril.txt
   include gromos.txt
   include water.txt
10 include NIPAMmol.txt
   out pdb NIPAM.pdb
   ;out cfg NIPAM.cfg
   ;out exl PAA.exl
   ;out ctf NIPAM.ctf
15 out dyn NIPAM.dyn

   [Create]
   Bisacrylamid(1,2,3,4)
   add 1 NIPAM=4
20 add 1 Bisacrylamid(6,7,8)
   crw 0.8 2 7 NIPAM
   crw 0.8 3 8 NIPAM
   add 4 NIPAM=3
   crw 0.6 4 6 NIPAM
25 ;Acetonitril=2
   Water=10
   density 1.0 Water
```

Abbildung 3.6: Struktur-Datenbank zur Erzeugung einer P-NIPAM Startstruktur.

Das BIS besitzt, wie in den Zeilen 26 bis 29 in Abbildung 3.5 an den Platzhaltern zu erkennen ist, vier offene Bindungen, denen durch die Anweisung **Bisacrylamid(1,2,3,4)** die Identifikationsnummern eins bis vier zugeordnet werden. An die offene Bindung mit der Nummer eins werden in Zeile 19 vier NIPAM-Monomere zufällig angefügt. Die Anzahl der hier verwendeten Monomere bestimmt entscheidend das Verhältnis zwischen Quervernetzer- und Ketten-Monomer. An diese dann an einem NIPAM Monomer hängende offene Bindung wird durch das Kommando in Zeile 20 ein Quervernetzer gebunden, dies ergibt das in Abbildung 3.7b gezeigte Teilpolymer. Eine der vier offenen Bindungen des Quervernetzers wird durch die Bindung mit der offenen Bindung des Ketten-Monomers geschlossen. Es verbleiben drei weitere Bindungen, denen auf Grund des Kommandos in Zeile 20 die Identifikationsnummern 6 bis 8 zugeordnet werden. An jede der offenen Bindungen können zur Verringerung des Quervernetzeranteils weitere zufällige NIPAM-Ketten angehängt werden, so zum Beispiel für die Bindung Nummer 4 in der Zeile 23, in der drei NIPAM-Monomere in zufälliger Ausrichtung angefügt werden.

Zur Verknüpfung der offenen Bindungen wird die in Abschnitt 3.4 beschriebene Methode des „directed random walk“ verwendet, das entsprechende Kommando hierfür lautet: **crw** („connect random walk“). Der erste Parameter des Kommandos legt die Wahrscheinlichkeit eines gerichteten Aufbaus fest, je gerich-

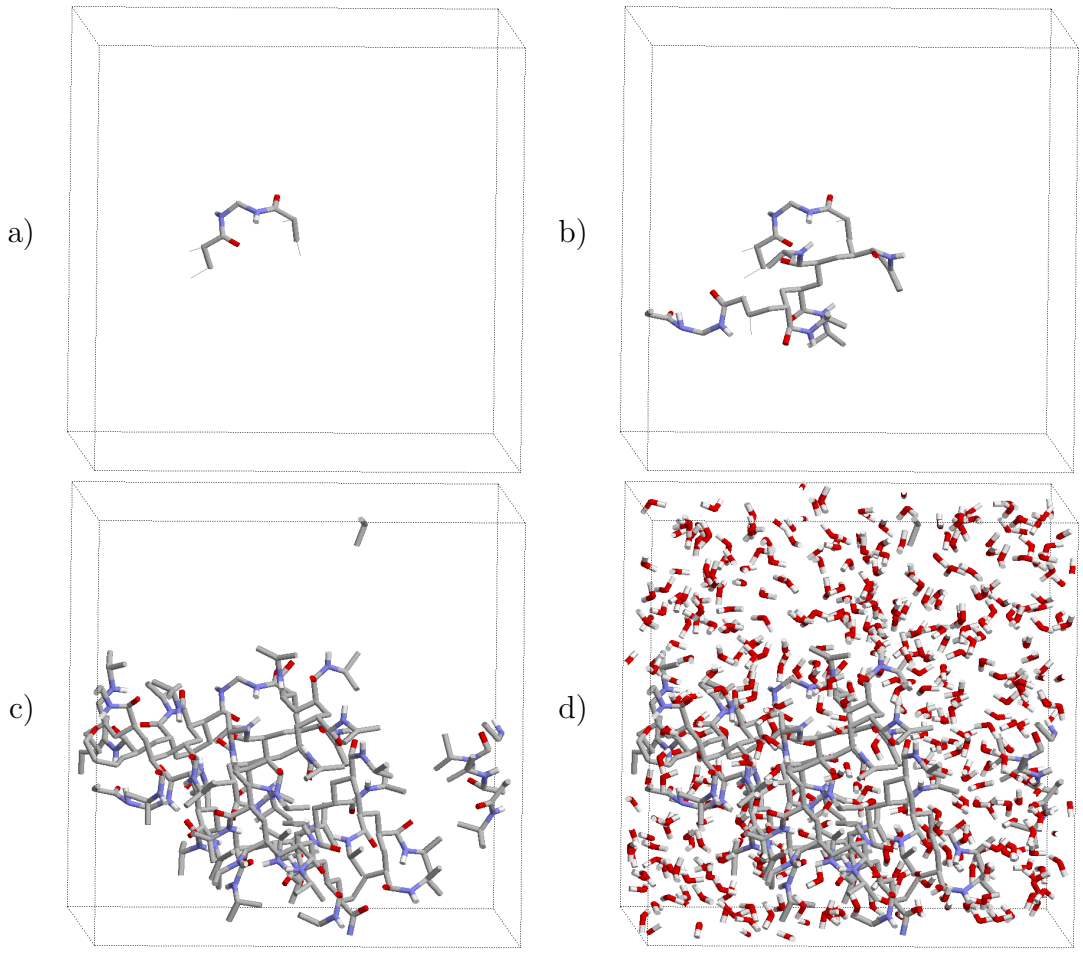


Abbildung 3.7: Konstruktion von P-NIPAM mit zwei BIS Quervernetzern.
a) Phase 1: Platzierung des BIS.
b) Phase 2: Anbringen einer Kette aus NIPAM-Monomeren und des zweiten BIS.
c) Phase 3: Verbinden der offenen Liganden durch Monomerketten.
d) Phase 4: Auffüllen des Systems mit Wasser.

teter der Aufbau, desto kürzer die Polymerketten. Zu beachten ist, dass kurze Ketten zu einem hohen Quervernetzeranteil führen. Die nächsten beiden Parameter enthalten die Identifikationsnummern der zu verbindenden offenen Bindungen des Polymers. Das Monomer, mit dem die Bindungskette aufgebaut wird, wird im letzten Parameter angegeben. Bei der Konstruktion der Kette wird an jeder Bindung eine Teilkette erzeugt, hierbei wachsen beide Teilketten durch abwechselndes anfügen von Monomeren aufeinander zu. Ist der Abstand der offenen Bindungen kleiner als der im Abschnitt „Structure“ definierte minimale Abstand **crwTol**, werden beide Enden miteinander verbunden. Ist eine Verbindung beider Teile aus geometrischen Gründen jedoch nicht möglich, so werden die Teilketten gelöscht und erneut zufällig aufgebaut. Konnten alle offenen Bindungen durch

Ketten geschlossen werden, ist das Polymer konstruiert. Abbildung 3.7c zeigt hierzu ein entsprechend erzeugtes NIPAM-Polymer.

Moleküle ohne offene Bindungen werden, wie die Zeilen 25 und 26 der Abbildung 3.6 zeigen, durch Angabe ihres Namens und ihrer Häufigkeit in das Simulationssystem integriert. Soll am Ende der Konstruktionsphase das Gesamtsystem bis zu einer bestimmten Dichte mit einer Sorte Molekülen aufgefüllt werden, so besteht, wie in Zeile 27 der Abbildung 3.6, der letzte Befehl aus dem Kommando „**density**“. Dieser füllt solange den Simulationsraum mit der im zweiten Parameter angegebenen Sorte von Lösungsmittelmolekülen, bis die im ersten Parameter angegebene Dichte erreicht ist. Die sich daraus ergebende Startstruktur ist damit komplett und für das verwendete Beispiel eines NIPAM-Gels ergibt sich das in Abbildung 3.7d gezeigte System. Dieses kann nun in den unterschiedlichen Formaten abgespeichert und mit dem Simulationsprogramm „dynC“ equilibriert werden.

3.4 Strukturbildung im freien Raum

Weitaus flexibler als das Gitterstrukturbildungsprogramm (Abschnitt 3.2) ist das von mir entwickelte Strukturgenerierungsprogramm „StructC“, das die Strukturdaten der zu simulierenden Moleküle aus einer Struktur-Datenbank (Abschnitt 3.3) erzeugt. Bei dieser Strukturerzeugung wird der Simulationsraum, wie auch später in der Simulation, als kontinuierlich periodisch fortgesetzter Raum betrachtet. Die Erstellung der Moleküle erfolgt geordnet nach Molekülgröße, d. h. als erstes werden die größten und damit unhandlichsten Moleküle erzeugt. Denn bei auftretenden Raumproblemen ist es wesentlich einfacher ein kleines Molekül an einer anderen Stelle neu zu erzeugen als ein großes Molekül.

Das erste Atom eines Moleküls wird in dem Bereich des Raumes zufällig platziert, der zu keinem Konflikt bezüglich des Abstands zu bereits erzeugten Molekülen führt. Das zweite Atom wird dann zufällig, ebenfalls unter Vermeidung von Konflikten mit anderen Atomen, mit Hilfe eines zufällig erzeugten Richtungsvektors auf einer Kugelschale um das erste Atom platziert. Der Abstand zwischen dem ersten und dem zweiten Atom wird hierbei entweder durch die Angaben der Z-Matrix festgelegt, oder durch die Bindungslänge r_0 der Streckschwingung zwischen beiden Atomen bestimmt.

Für die Platzierung der weiteren Atome eines Moleküls ist eine spezielle Methode der Klasse „CCreateMol“ zuständig. Diese benötigt drei Referenzatome, so wie die Bindungslänge r_0, den Bindungswinkel θ_{ijk} und den Torsionswinkel ϕ_{ijkl} zur Bestimmung des Ortsvektors des neuen Atoms. Bei der Bestimmung der Koordinaten des dritten Atoms existiert jedoch noch kein drittes Referenzatom. Um aber trotzdem diese Methode nutzen zu können, wird ein virtuelles Atom l eingeführt, das an der selben Position wie das zweite Atom j liegt. Aus der Lage der Atome jkl ergibt sich dann ein lineares Teilmolekül, um dessen Achse das

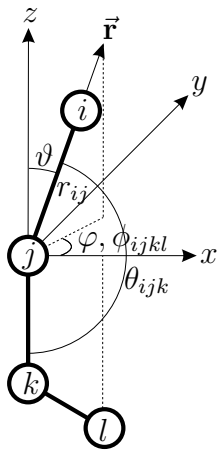


Abbildung 3.8: Strukturgenerierung aus der Torsionswechselwirkung.

Atom i frei rotieren kann. Die Bindungslänge r_0 und der Bindungswinkel θ_{ijk} werden durch die Z-Matrix festgelegt oder ergeben sich aus den Kraftfeldern. Der Torsionswinkel ϕ_{ijkl} wird zufällig gewählt, um eine zufällige Orientierung der ijk-Ebene zu erreichen.

Die Bestimmung der Ortsvektoren des vierten und aller weiteren Atome eines Moleküls erfolgt dann ohne ein virtuelles Atom. In der Regel lässt sich, wie Abbildung 2.3 zeigt, aus dem neuen Atom i, dem direkt verbundenen Atom j und einem mit j verbundenen Atom k ein Dreieck ijk bilden. Zusammen mit einem weiteren Atom l lässt sich im Gegensatz zum letzten Fall ein weiteres Dreieck jkl bilden. Zwischen beiden Dreiecken liegt der Torsionswinkels ϕ_{ijkl}. Der Ortsvektor des neuen Atoms i ist somit eindeutig bis auf einen Faktor $n \in \{0, 1, .., m-1\}$ festgelegt, dieser bestimmt das vom System festgelegte zu wählende Energieminimum der Torsionswechselwirkung. Zur Festlegung dieses Faktors bzw. des Torsionswinkels gibt es im Programm vier Möglichkeiten:

1. Die explizite Angabe des Torsionswinkels ϕ_{ijkl} in der Z-Matrix.

2. Der Faktor n wird in der Z-Matrix angegeben.

3. Der Wert n wird zufällig aus den möglichen Werten ausgewählt.

4. Es wird ein optimaler Faktor n aus der Berechnung des absoluten Minimums der van der Waals- und Coulomb-Wechselwirkungsenergie bestimmt. Gilt das absolute Minimum für mehrere Faktoren n, so wird einer von ihnen zufällig ausgewählt.

Welche der Varianten letztendlich verwendet wird, hängt von der beabsichtigten Variabilität der Molekülstruktur ab.

Zur Bestimmung der Koordinaten des neuen Atoms i wird aus den bereits definierten Atomen jkl ein lokales, kartesisches Koordinatensystem erzeugt. Hierbei

bilden die Vektoren $\vec{\mathbf{L}}_x$,$\vec{\mathbf{L}}_y$ und $\vec{\mathbf{L}}_z$ die Achsen des in Abbildung 3.8 gezeigten lokalen Koordinatensystems. Der Vektor $\vec{\mathbf{L}}_z$ zeigt immer von Atom k auf Atom j. Spannen die Koordinaten der Atome i, j und k eine Ebene ijk auf, so wird der Vektor $\vec{\mathbf{L}}_y$ aus dem Vektorprodukt des Vektors von Atom k nach Atom l und dem Vektor $\vec{\mathbf{L}}_z$ errechnet, dadurch ist er senkrecht zu $\vec{\mathbf{L}}_z$ und senkrecht zur Ebene jkl. Falls die Atome j, k und l keine Ebene aufspannen, d. h. ein lineares Teilmolekül bilden, muss der Vektor $\vec{\mathbf{L}}_y$ nur senkrecht auf dem Vektor $\vec{\mathbf{L}}_z$ stehen. Aus der folgenden Definition für einen senkrechten Vektor:

$$\vec{\mathbf{L}}_y \; \vec{\mathbf{L}}_z = 0$$

lässt sich einfach eine Berechnungsvorschrift für einen zufälligen, zu $\vec{\mathbf{L}}_z$ senkrechten Vektor $\vec{\mathbf{L}}_y$ ableiten:

$$\begin{aligned}
(\vec{\mathbf{L}}_y)_x &= (\vec{\mathbf{L}}_z)_y\big((\vec{\mathbf{L}}_z)_z - 1\big) - (\vec{\mathbf{L}}_z)_z \\
(\vec{\mathbf{L}}_y)_y &= (\vec{\mathbf{L}}_z)_x\big((\vec{\mathbf{L}}_z)_z + 1\big) \\
(\vec{\mathbf{L}}_y)_z &= (\vec{\mathbf{L}}_z)_x\big(1 - 2(\vec{\mathbf{L}}_z)_y\big)
\end{aligned} \tag{3.2}$$

Der letzte Vektor $\vec{\mathbf{L}}_x$ auf der X-Achse wird mit Hilfe des Vektorprodukts von $\vec{\mathbf{L}}_z$ und $\vec{\mathbf{L}}_y$ bestimmt. Durch Normierung werden die Vektoren $\vec{\mathbf{L}}_x$,$\vec{\mathbf{L}}_y$ und $\vec{\mathbf{L}}_z$ in die Basisvektoren des von den Atome j, k und l definierten kartesischen Koordinatensystems umgewandelt.

Um mit Hilfe der Basisvektoren $\vec{\mathbf{L}}_x$,$\vec{\mathbf{L}}_y$ und $\vec{\mathbf{L}}_z$ die Position des neuen Atoms i zu bestimmen, wird zuvor ein Richtungsvektor $\vec{\mathbf{r}}_K = (1, \vartheta, \varphi)^T$ aus den Winkeln ϑ und φ gebildet. Die Winkel ϑ und φ des Richtungsvektors werden aus den oben erwähnten Randparametern von Bindungswinkel θ_0 und Torsionsparameter ϕ_0, n und m wie folgt bestimmt:

$$\vartheta = \pi - \theta_{ijk} = \pi - \theta_0 \qquad \varphi = \phi_{ijkl} = \frac{\pi + 2n\pi + \phi_0}{m} \tag{3.3}$$

Die Transformation des Richtungsvektors $\vec{\mathbf{r}}_K$ aus dem Kugelkoordinatensystem in das lokale kartesische Koordinatensystem $(\vec{\mathbf{L}}_x, \vec{\mathbf{L}}_y, \vec{\mathbf{L}}_z)$ liefert einen neuen Richtungsvektor $\vec{\mathbf{r}}_L$. Durch eine weitere Koordinatentransformation wird dieser Richtungsvektor $\vec{\mathbf{r}}_L$ in den Richtungsvektor $\vec{\mathbf{r}}_G$ des globalen kartesischen Koordinatensystems überführt. Zusammen mit der Bindungslänge r_0 aus der Z-Matrix bzw. aus den Parametern des Kraftfelds legt der Richtungsvektor $\vec{\mathbf{r}}_G$ die relative Lage des neuen Atoms i im globalen Koordinatensystem bezüglich des Atoms j fest.

Bei der Erzeugung von Polymeren, die aus einer wiederkehrenden Abfolge von Monomeren bestehen, als auch bei der Erzeugung von Polymernetzwerken, in denen die Polymere zusätzlich durch Vernetzungsmonomere mit einander verbunden sind, wäre eine fest vorgegebene Struktur wie es sie bei den Molekülen gibt ungeeignet. Zum Einen wäre die Erstellung einer entsprechenden Struktur-Datenbank

sehr aufwendig und würde zudem redundante Informationen enthalten, da das Polymer aus kleinen wiederkehrenden Einheiten (Monomeren) besteht. Zum Anderen führte jede Änderung der Quervernetzeranteile, der Kettenlängen und jede Variation im Polymergerüst zu einer neuen Z-Matrix, was einen hohen Aufwand zur Folge hätte. Es ist deshalb sinnvoller, zur Erzeugung von Polymeren / Polymernetzwerken nur die Struktur der einzelnen Monomereinheiten vorzugeben und diese dann zu großen Molekülen in einem extra Schritt zu verbinden.

Das Strukturgenerierungsprogramm „StructC“ folgt diesem Ansatz, und es konstruiert das Polymergerüst im freien Raum ohne Einschränkung durch ein die Form bestimmendes starres Gitter. Die Atome werden dabei so platziert, dass die in den Kraftfeldparametern definierten Gleichgewichtsabstände und Winkel genauso eingehalten werden wie bei den einfachen Molekülen. Da aufgrund der hohen Librationsgeschwindigkeiten der Wasserstoffatome der Abstand zu ihren Bindungspartnern durch Zwangskräfte fixiert werden muss, ist besonders hier die Einhaltung des Gleichgewichtsabstandes wichtig.

Wie aus dem letzten Abschnitt über die Struktur-Datenbank bekannt ist, beginnt die Generierung des Polymergerüstes mit der zufälligen Platzierung eines Quervernetzermonomers. An einer der offenen Bindungen des Quervernetzers wird eine Polymerkette angebaut, die durch einen weiteren Quervernetzer abgeschlossen wird. An diesen können weitere Polymerketten angebracht werden, die entweder wiederum durch Quervernetzer abgeschlossen werden oder durch einen „directed random walk“ eine Verbindung mit einer anderen offenen Bindung eines Quervernetzers oder einer Teilkette eingehen.

Die hier verwendete Variante des „directed random walk“ unterscheidet sich grundlegend von der weniger realistischen im Gittermodell verwendeten Variante, bei der eine Polymerkette von einem Quervernetzer zu einem anderen unter Zuhilfenahme der Knotenpunkte eines Gitters aufgebaut wird. Durch den von einem Gitter unabhängigen Aufbau der Polymerketten ist es nun aber unwahrscheinlich, dass Bindungslänge und Bindungswinkel am Verknüpfungspunkt der Enden den Gleichgewichtswerten entsprechen. Damit sich dieses Ungleichgewicht schnell ausgleichen kann, sollte der Verknüpfungspunkt an eine möglichst flexiblen Stelle der Polymerkette liegen. Am flexibelsten ist die Mitte der Ketten, da die Quervernetzer durch die Verknüpfung mit weiteren Polymerketten in ihrer Flexibilität stark gehindert sind. Aus diesem Grund werden bei der Konstruktion Monomere abwechselnd an beiden Enden mit offener Bindung angebracht, diese wachsen durch den „directed random walk“ aufeinander zu.

Die Wachstumsrichtung einer Polymerkette ist durch die internen Wechselwirkungen der Moleküle stark eingeschränkt. Ein Atom des Polymerrückens kann unter Einhaltung der Gleichgewichtsparameter der intramolekularen Wechselwirkungen nur auf Grund der Periodizität m des Torsionswinkels die Wachstumsrichtung des Polymers ändern. Die Polymerkette kann deshalb durch Anfügen eines Atoms nur in m verschiedene Raumrichtungen wachsen, wobei im Normalfall der Abstand der Enden beider Ketten in Abhängigkeit von der Wachstumsrichtung

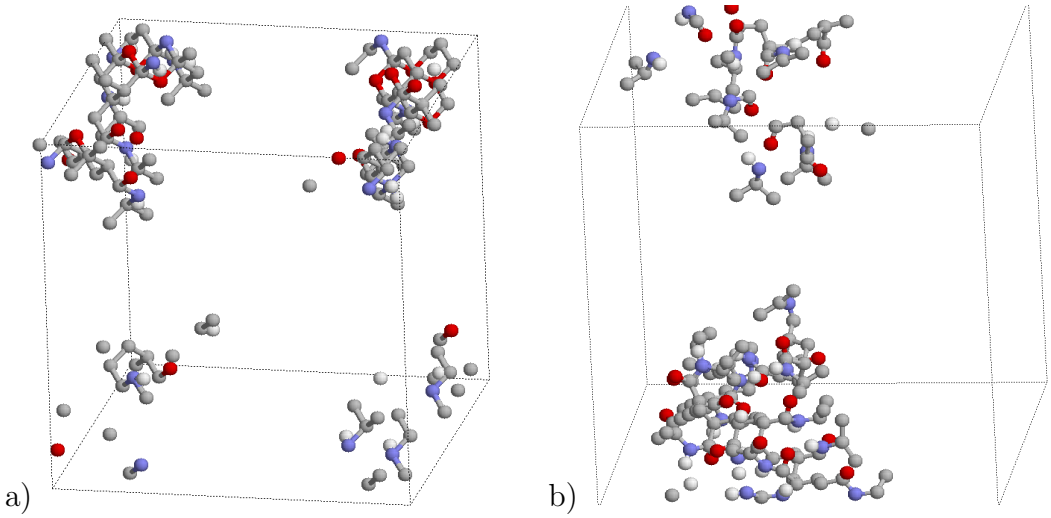


Abbildung 3.9: Vergleich der Startstruktur des P-NIPAM Polymers aus System A in Tabelle 5.1 erzeugt in einem Gitter (a) und des P-NIPAM Polymers aus System D erzeugt im freien Raum (b). Das Polymer besteht jeweils aus zwei Bisacrylamid Quervernetzern und aus 15 bzw. 19 NIPAM Kettenmonomeren, wobei die Periodischen Randbedingungen die Polymere in mehrere Segmente aufspalten.

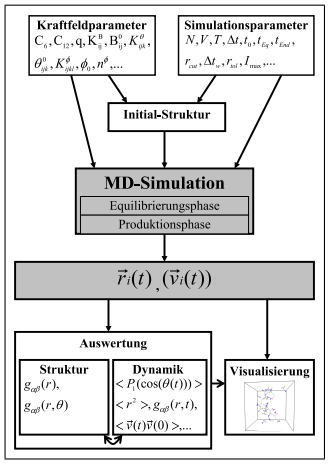
unterschiedlich stark variiert.

Der „directed random walk“ wählt mit einer vorgegebenen Wahrscheinlichkeit einen auf das andere Ende des Polymers gerichteten Aufbauschritt aus, dabei wird die Kette in einer der m möglichen Richtungen erweitert, die den Abstand zum anderen Ende der Kette minimieren. D. h. bei der Platzierung des neuen Atoms am Polymerrücken wird eines der Energieminima des Torsionswinkels gewählt, das den Abstand zum anderen Ende der Kette minimiert. Mit Hilfe des „directed random walk“ verringert sich der Abstand der Enden mit zunehmender Kettenlänge. Weicht der End- zu Endabstand um weniger als um den in **crwTol** festgelegten Faktor von der Bindungslänge ab, ist der Aufbau dieser Polymerkette beendet und die beiden Endstücke werden zu einem Polymerstrang verbunden.

Vergleicht man beide Verfahren, so liefert letzteres, wie in Abbildung 3.9 zu sehen, eine wesentlich natürlichere Ausgangssituation für die anschließende Equilibrierung des Systems. Das hier beschriebene Verfahren entspricht zudem mehr der im Abschnitt 5.2 beschriebenen realen Polymerisation von P-NIPAM. Außerdem lassen sich mit dem Programm „StructC“ die unterschiedlichsten Moleküle und Polymere erzeugen, da dieses nicht auf die Konstruktion von Polymeren mit fester Gitterstruktur festgelegt ist.

Kapitel 4

Simulation



Die MD-Simulation dient dazu, den vom gewählten System aufgebauten Phasenraum zu durchmustern. Sie löst hierfür numerisch die Newtonsche Bewegungsgleichung (4.1) der klassischen Mechanik. Im Phasenraum wird ein Startpunkt gewählt, der dem Simulationsprogramm in Form einer Startstruktur vorgegeben wird. Durch die Integration der Bewegungsgleichungen wandert die Simulation vom Startpunkt auf einem Pfad im Phasenraum, der für lange Simulationen zu einer Durchmusterung eines Abschnittes des Phasenraums führt. Befindet sich das System im thermischen Gleichgewicht, erzeugt man wegen der Ergodenhypothese (Kapitel 7) eine repräsentative Darstellung des gesamten Ensembles. Zur späteren Auswertung kann der hierbei entstehende Phasenraumpfad als Trajektorie abgespeichert werden.

Während der Simulation erfolgt die Integration der Bewegungsgleichung (4.1) in festen Zeitschritten δt, dessen Größe im wesentlichen durch die Trägheit der simulierten Atome bestimmt wird. Um eine hohe Schrittweite und somit eine schnelle Durchmusterung des Phasenraums zu erreichen, werden die einzeln simulierten Wasserstoffatome, die leichtesten und somit beweglichsten Atome, durch Zwangskräfte (4.2) an den Bindungspartnern fixiert.

Werden während einer Simulation nur die Newtonschen Bewegungsgleichungen gelöst, wird das abgeschlossene System bei konstanter Anzahl Atome N, festem Volumen V und konstanter Gesamtenergie E im mikrokanonischen Ensemble (NVE-Ensemble) simuliert. Für viele praktische Anwendungen ist das mikrokanonische Ensemble allerdings ungeeignet, da entsprechende Experimente im Allgemeinen bei fester Temperatur T und / oder festem Druck p ausgeführt werden. Häufig ist es sinnvoller im kanonischen Ensemble, dem NVT-Ensemble oder im großkanonischen Ensemble (NPT-Ensemble) zu simulieren. Insbesondere ist es während der Equilibrierungsphase auf dem Weg zum thermischen Gleichgewicht erforderlich, die Temperatur T zu korrigieren, anderen Falls würde die kinetische Energie durch die Umwandlung von potentieller Energie sehr stark steigen. Die Fixierung bzw. Korrektur der Temperatur T und des Druckes p erfolgt mit Hilfe der in Abschnitt 4.3 exemplarisch beschriebenen Thermostaten und Barostaten.

Mit dem von mir geschriebenen Programm „dynC“ können insbesondere Molekulardynamische Simulationen von Polymeren und Gelen durchgeführt werden. Als Startstruktur kann sowohl eine von „StructC“ erzeugte Struktur-Datenbank (DYN), als auch eine von „dynC“ während einer Simulation selbst generierte Struktur-Datenbank (DYN) dienen.

4.1 Integrations Algorithmen

Bei den in MD-Simulationen verwendeten kontinuierlichen Potentialen hat das Simulationsprogramm die Newtonschen Bewegungsgleichungen 4.1 zu lösen, die durch das erste Newtonsche Axiom bestimmt werden:

$$\vec{\mathbf{F}} = m\vec{\mathbf{a}} \tag{4.1}$$

Die zeitliche Integration der Bewegungsgleichung erfolgt in der Regel in festen Zeitinkrementen δt, so dass die Trajektorie in festen Intervallen abgespeichert werden kann. Mit Hilfe der Integration der Bewegungsgleichungen werden aus den bestehenden Ortsvektoren $\vec{\mathbf{r}}_i(t)$ und Geschwindigkeitsvektoren $\vec{\mathbf{v}}_i(t)$ der Atome i, die Ortsvektoren $\vec{\mathbf{r}}_i(t+\delta t)$ und Geschwindigkeitsvektoren $\vec{\mathbf{v}}_i(t+\delta t)$ zum nächsten Zeitpunkt $t+\delta t$ bestimmt. Besonders einfach ist der Integrations-Algorithmus von Verlet [38]. Man erzeugt zur Bestimmung von $\vec{\mathbf{r}}_i(t+\delta t)$ die Taylor-Reihe um $t+\delta t$:

$$\begin{aligned}\vec{\mathbf{r}}_i(t+\delta t) &= \vec{\mathbf{r}}_i(t) + \delta t\dot{\vec{\mathbf{r}}}_i(t) + \frac{\delta t^2}{2}\ddot{\vec{\mathbf{r}}}_i(t) + \frac{\delta t^3}{3!}\dddot{\vec{\mathbf{r}}}_i + O\left(\delta t^4\right) \\ &= \vec{\mathbf{r}}_i(t) + \vec{\mathbf{v}}_i(t)\delta t + \frac{\vec{\mathbf{F}}_i(t)}{2m}\delta t^2 + \frac{\delta t^3}{3!}\dddot{\vec{\mathbf{r}}}_i + O\left(\delta t^4\right)\end{aligned} \tag{4.2}$$

so wie die Taylor-Reihe um $t-\delta t$:

$$\vec{\mathbf{r}}_i(t-\delta t) = \vec{\mathbf{r}}_i(t) - \vec{\mathbf{v}}_i(t)\delta t + \frac{\vec{\mathbf{F}}_i(t)}{2m}\delta t^2 - \frac{\delta t^3}{3!}\dddot{\vec{\mathbf{r}}}_i + O\left(\delta t^4\right) \tag{4.3}$$

Die Addition von Gleichung (4.2) und (4.3) liefert den Verlet-Algorithmus:

$$\vec{\mathbf{r}}_i(t+\delta t) = 2\vec{\mathbf{r}}_i(t) - \vec{\mathbf{r}}_i(t-\delta t) + \frac{\vec{\mathbf{F}}_i(t)}{m}\delta t^2 + O\left(\delta t^4\right) \tag{4.4}$$

Der Geschwindigkeitsvektor eines Atoms wird in diesem Verfahren nur indirekt durch Differenzierung der Ortsvektoren (4.4) bestimmt:

$$\vec{\mathbf{v}}_i(t) \approx \frac{\vec{\mathbf{r}}_i(t+\delta t) - \vec{\mathbf{r}}_i(t-\delta t)}{2\delta t} \tag{4.5}$$

Die Differenzierung reduziert jedoch die Genauigkeit des Geschwindigkeitsvektors, diese liegt in der Größenordnung von $O\left(\delta t^2\right)$.

Beim Leapfrog-Algorithmus [39] werden dem gegenüber sowohl Ortsvektor als auch Geschwindigkeitsvektor berechnet, die Berechnungen erfolgen jedoch zeitlich versetzt. Der Leapfrog-Algorithmus oder auch Bock-Sprung-Algorithmus ist technisch äquivalent zum Verlet-Algorithmus, da er aus ihm hergeleitet werden kann. Der Name Bock-Sprung-Algorithmus stammt von der Tatsache, dass der Ortsvektor und der Geschwindigkeitsvektor immer abwechselnd zu unterschiedlichen Zeitpunkten bestimmt wird. Um den Ortsvektor vom Startzeitpunkt aus zu jedem anderen Zeitschritt zu erhalten, wird der Geschwindigkeitsvektor immer zum halben Zeitschritt $\delta t/2$ berechnet:

$$\vec{\mathbf{v}}\left(t - \frac{\delta t}{2}\right) := \frac{\vec{\mathbf{r}}(t) - \vec{\mathbf{r}}(t - \delta t)}{\delta t} \tag{4.6}$$

$$\vec{\mathbf{v}}\left(t + \frac{\delta t}{2}\right) := \frac{\vec{\mathbf{r}}(t + \delta t) - \vec{\mathbf{r}}(t)}{\delta t} \tag{4.7}$$

Durch Umformung liefert die Definition der Geschwindigkeiten (4.7) den Ausdruck für die Ortsbestimmung:

$$\vec{\mathbf{r}}_i(t + \delta t) = \vec{\mathbf{r}}_i(t) + \delta t \vec{\mathbf{v}}_i\left(t + \frac{\delta t}{2}\right) \tag{4.8}$$

Mit Hilfe des Verlet-Algorithmus lässt sich dann auch aus alter Geschwindigkeit $\vec{\mathbf{v}}_i\left(t - \delta t/2\right)$ und Kraft $\vec{\mathbf{F}}_i$ auf die neue Geschwindigkeit $\vec{\mathbf{v}}_i\left(t + \delta t/2\right)$ schließen:

$$\vec{\mathbf{v}}_i\left(t + \frac{\delta t}{2}\right) = \vec{\mathbf{v}}_i\left(t - \frac{\delta t}{2}\right) + \delta t \frac{\vec{\mathbf{F}}_i\big(r(t)\big)}{m_i} \tag{4.9}$$

In der Simulation erzeugt der Leapfrog-Algorithmus somit folgendes Ablaufschema:

$$\begin{array}{rcl} \vec{\mathbf{r}}(t) & = & \vec{\mathbf{r}}(t - \delta t) + \delta t \vec{\mathbf{v}}\left(t - \frac{\delta t}{2}\right) \\ \vec{\mathbf{F}}\big(\vec{\mathbf{r}}(t)\big) & & \text{zum nächsten Zeitschritt} \\ \vec{\mathbf{v}}\left(t + \frac{\delta t}{2}\right) & = & \vec{\mathbf{v}}\left(t - \frac{\delta t}{2}\right) + \delta t \frac{\vec{\mathbf{F}}(\vec{\mathbf{r}}(t))}{m} \\ \vec{\mathbf{r}}(t + \delta t) & = & \vec{\mathbf{r}}(t) + \delta t \vec{\mathbf{v}}\left(t + \frac{\delta t}{2}\right) \end{array}$$

Werden Orte und Geschwindigkeiten zum selben Zeitpunkt benötigt, um zum Beispiel die Gesamtenergie aus potentieller und kinetischer Energie zu berechnen, bestimmt man einen mittleren Geschwindigkeitsvektor $\vec{\mathbf{v}}_i\left(t\right)$ aus den Geschwindigkeitsvektoren der halben Zeitschritte:

$$\vec{\mathbf{v}}_i\left(t\right) = \frac{\vec{\mathbf{v}}_i\left(t + \frac{\delta t}{2}\right) + \vec{\mathbf{v}}_i\left(t - \frac{\delta t}{2}\right)}{2} \tag{4.10}$$

Dies bedeutet natürlich einen zusätzlichen Rechenaufwand, so wie zusätzlichen Speicher für jedes Atom, da zu jedem Atom neben dem Ortsvektor $\vec{\mathbf{r}}_i(t)$ und dem Geschwindigkeitsvektor $\vec{\mathbf{v}}_i\left(t + {}^{\delta t}/_2\right)$ auch zusätzlich der mittlere Geschwindigkeitsvektor $\vec{\mathbf{v}}_i\left(t\right)$ oder der Geschwindigkeitsvektor $\vec{\mathbf{v}}_i\left(t - {}^{\delta t}/_2\right)$ zum vorhergehenden Zeitschritt gespeichert werden muss.

Der velocity Verlet Algorithmus [40] verfolgt einen anderen Ansatz, hier werden Ortsvektor und Geschwindigkeitsvektor zum gleichen Zeitpunkt bestimmt:

$$\vec{\mathbf{r}}_i(t+\delta t) = \vec{\mathbf{r}}_i(t) + \delta t \vec{\mathbf{v}}_i(t) + \frac{\delta t^2}{2m_i}\vec{\mathbf{F}}_i(t) \tag{4.11}$$

$$\vec{\mathbf{v}}_i(t+\delta t) = \vec{\mathbf{v}}_i(t) + \frac{\delta t}{2m_i}\left(\vec{\mathbf{F}}_i(t) + \vec{\mathbf{F}}_i(t+\delta t)\right) \tag{4.12}$$

Auf den ersten Blick erscheint es so, also ob der Preis für die Gleichzeitigkeit von Ortsvektor und Geschwindigkeitsvektor, die Vorhaltung der Kraft zu zwei Zeitpunkten wäre. Doch durch eine geschickte Wahl des Integrationsalgorithmusses ist es möglich, mit nur einem Zeitpunkt auszukommen:

Startwerte:	$\vec{\mathbf{r}}_i(0),\ \vec{\mathbf{v}}_i(0),\ \vec{\mathbf{F}}_i(0) = \vec{\mathbf{f}}\left(\vec{\mathbf{r}}_0(0)\ldots\vec{\mathbf{r}}_N(0)\right)$
Schleife	$n = 0\ldots n_{max}$ für alle Atome i
(Zeitschritt):	$\tilde{\vec{\mathbf{v}}}_i(t) = \vec{\mathbf{v}}_i(t) + \frac{\delta t}{2m_i}\vec{\mathbf{F}}_i(t)$
	$\vec{\mathbf{r}}_i(t+\delta t) = \vec{\mathbf{r}}_i(t) + \delta t\tilde{\vec{\mathbf{v}}}_i(t)$
	$\vec{\mathbf{F}}_i(t+\delta t) = \vec{\mathbf{f}}\left(\vec{\mathbf{r}}_0(t+\delta t)\ldots\vec{\mathbf{r}}_N(t+\delta t)\right) = -\sum_{i\neq j}\frac{\vec{\mathbf{r}}_{ij}(t+\delta t)}{r_{ij}(t+\delta t)}\nabla_j U$
	$\vec{\mathbf{v}}_i(t+\delta t) = \tilde{\vec{\mathbf{v}}}_i(t) + \frac{\delta t}{2m_i}\vec{\mathbf{F}}_i(t+\delta t)$

Die drei angegebenen Algorithmen sind im Prinzip äquivalent und erzeugen einen globalen Fehler der Ordnung $O(\delta t^2)$. Der velocity Verlet Algorithmus ist jedoch den anderen vorzuziehen, da Ortsvektoren und Geschwindigkeitsvektoren für den selben Zeitpunkt berechnet werden. Die Speicherung der Trajektorie, die den Ortsvektor und den Geschwindigkeitsvektor enthält, ist deshalb ohne zusätzlichen Rechenaufwand möglich. Außerdem ist eine Änderung der Simulationsschrittweite problemlos, somit ist die Thermalisierung im Vergleich mit dem Verlet Algorithmus wesentlich einfacher.

4.2 Zwangskräfte

Die Schrittweite, mit der Simulationen betrieben werden können, ist wesentlich von der Bewegung der beweglichsten Atome in einer Simulationen abhängig. Wasserstoffatome haben mit Abstand die geringste Masse, somit sind sie auch die beweglichsten Atome mit den höchsten Schwingungsfrequenzen. Folglich bestimmen

sie die Schrittweite des Integrationsalgorithmusses. Werden die Bindungsabstände der Wasserstoffatome zu ihren Bindungspartnern mit speziellen Routinen in ihrem Gleichgewichtsabstand fixiert, sind wesentlich größere Schrittweiten möglich.

Die Verfahren zur Fixierung des Gleichgewichtsabstandes beruhen auf der Theorie der holonomen Zwangsbedingungen der Lagrange Darstellung. Sie werden in die Standard Integrationsalgorithmen integriert und „korrigieren“ die Ortsvektoren und Geschwindigkeitsvektoren der Atome, die den Zwangsbedingungen unterliegen. Zu den verschiedenen Integrationsverfahren existieren unterschiedliche Methoden um die Zwangsbedingungen zu erfüllen, so ist für das Verlet Verfahren der SHAKE Algorithmus [41] verbreitet. Das bessere velocity Verlet Verfahren verwendet im allgemeinen den RATTLE Algorithmus von Andersen [42].

Im Prinzip teilen alle Algorithmen den Integrationsschritt in zwei Teilschritte auf. Im ersten Teilschritt wird die Bewegungsgleichung ohne Zwangsbedingungen (unconstrained) gelöst, um dann im zweiten Teilschritt mit Hilfe von Zwangskräften die Zwangsbedingungen zu erfüllen. Die Zwangskräfte $\vec{\mathbf{g}}$ werden als virtuelle Kräfte eingeführt, um die Bindungslänge zwischen Atom i und j nach dem ersten Teilschritt wieder auf die vorgegebene Länge d_{ij} zu korrigieren. Die zwischen beiden Atomen in entgegengesetzte Richtungen wirkenden Kräfte verlaufen entlang der Zwangsbedingung, d. h. entlang der Atombindung. Sie korrigieren die Atombewegung in der Art, dass der Abstand der Atome nach jedem Simulationsschritt dem Gleichgewichtsabstand der Bindung entspricht. Zu jeder Zwangskraft $\vec{\mathbf{g}}_{ij}$ auf ein Atom i gibt es eine entsprechende Gegenkraft $\vec{\mathbf{g}}_{ji}$ auf das zweite Atom j entlang der Zwangsbedingung, so dass sich die Zwangskräfte innerhalb eines Moleküls gegenseitig aufheben. Dies ist insbesondere für die Energieerhaltung eine notwendige Bedingung. Zwangskräfte erzeugen nur virtuelle Kräfte, d. h. sie haben kein nach außen wirkendes Potential und dürfen folglich keine Kräfte auf das Molekül als ganzes ausüben. Für die Newtonsche Bewegungsgleichung (4.1) führen die Zwangskräfte zu einer Erweiterung der Gleichung, so dass für ein Atom i gilt:

$$m_i\vec{\mathbf{a}}_i = \vec{\mathbf{F}}_i\big(\vec{\mathbf{r}}_i(t)\big) + \vec{\mathbf{g}}_i\big(\vec{\mathbf{r}}_i(t), \vec{\mathbf{v}}_i(t)\big) \tag{4.13}$$

Die Zwangskräfte $\vec{\mathbf{g}}$ hängen sowohl von den Zwangsbedingungen als auch von den auf die Atome wirkenden Wechselwirkungen ab, zu ihrer Beschreibung enthalten sie zeitabhängige Lagrangemultiplikatoren.

Im Unterschied zum SHAKE Algorithmus (basiert auf dem Verfahren von Verlet) enthält das RATTLE-Verfahren eine Approximation für die Ortsgleichung und eine für die Geschwindigkeitsgleichung, denn im velocity Verlet Verfahren erfolgt zwischen der Bestimmung des Ortsvektors $\vec{\mathbf{r}}_i(t+\delta t)$ und des Geschwindigkeitsvektors $\vec{\mathbf{v}}_i(t+\delta t)$ die Berechnung der Kraft $\vec{\mathbf{F}}_i(t+\delta t)$ für den neuen Zeitpunkt. Im RATTLE-Verfahren gilt für die Ortsgleichung des Atoms i:

$$\vec{\mathbf{r}}_i(t+\delta t) = \vec{\mathbf{r}}_i(t) + \delta t\vec{\mathbf{v}}_i(t) + \frac{(\delta t)^2}{2m_i}\left(\vec{\mathbf{F}}_i\big(\vec{\mathbf{r}}_i(t)\big) + \vec{\mathbf{g}}_{ri}(t)\right) \tag{4.14}$$

Aus dem neuen Ortsvektor $\vec{\mathbf{r}}_i(t+\delta t)$ lässt sich die auf das Atom i wirkende Kraft $\vec{\mathbf{F}}_i\left(\vec{\mathbf{r}}_i(t+\delta t)\right)$ und damit auch die neue Geschwindigkeit $\vec{\mathbf{v}}_i(t+\delta t)$ des Atoms bestimmen:

$$\begin{aligned}\vec{\mathbf{v}}_i(t+\delta t) &= \vec{\mathbf{v}}_i(t) + \frac{\delta t}{2m_i}\Big(\vec{\mathbf{F}}_i\big(\vec{\mathbf{r}}_i(t)\big) + \vec{\mathbf{g}}_{ri}\big(t\big) \\ &\quad + \vec{\mathbf{F}}_i\big(\vec{\mathbf{r}}_i(t+\delta t)\big) + \vec{\mathbf{g}}_{vi}\big(t+\delta t\big)\Big) \end{aligned} \tag{4.15}$$

Daraus ergeben sich für das RATTLE-Verfahren einige wichtige Eigenschaften:

- Zur Berechnung von $\vec{\mathbf{r}}(t+\delta t)$ und $\vec{\mathbf{v}}(t+\delta t)$ sind nur $\vec{\mathbf{r}}(t)$ und $\vec{\mathbf{v}}(t)$ zu einem Zeitpunkt t erforderlich, keine Daten aus vorhergehenden Zeitschritten.
- Durch die Aufspaltung der Zwangskräfte in einen Orts- und einen Geschwindigkeitsanteil erfüllen die Ortsvektoren $\vec{\mathbf{r}}_i(t)$ zu jedem Zeitpunkt t die Zwangsbedingungen.
- Zu jedem Zeitpunkt t erfüllen die Geschwindigkeitsvektoren $\vec{\mathbf{v}}_i(t)$ die Zwangsbedingungen.
- Die Genauigkeit ist vergleichbar mit der des velocity Verlet (Fehler: $O(\delta t^2)$).

Zur Bestimmung der Zwangskräfte $\vec{\mathbf{g}}_{ri}(t)$ und $\vec{\mathbf{g}}_{vi}(t)$, die die Atome i und j im Gleichgewichtsabstand d_{ij} halten sollen, wird eine Funktion σ_{ij} der Abweichung vom Sollwert d_{ij} definiert:

$$\sigma_{ij} = \left|\vec{\mathbf{r}}_i(t) - \vec{\mathbf{r}}_j(t)\right|^2 - d_{ij}^2 \tag{4.16}$$

Erfüllen die Atome i und j die Zwangsbedingung, so gilt die Zwangsbedingungsgleichung:

$$\left|\vec{\mathbf{r}}_i(t) - \vec{\mathbf{r}}_j(t)\right|^2 - d_{ij}^2 = 0 \tag{4.17}$$

Die Ableitung der Zwangsbedingungsgleichung liefert die Gleichung, die von den Geschwindigkeiten eingehalten werden müssen um die Zwangsbedingung zu erfüllen:

$$\left|\vec{\mathbf{v}}_i(t) - \vec{\mathbf{v}}_j(t)\right|\left|\vec{\mathbf{r}}_i(t) - \vec{\mathbf{r}}_j(t)\right| = 0 \tag{4.18}$$

Für den bei Ryckaert u. a. [41] beschriebenen Zusammenhang zwischen der Zwangskraft $\vec{\mathbf{g}}_i(t)$ auf Atom i und der Gewichtung der Zwangskräfte mit dem Lagrangemultiplikatoren $\lambda_{ij}(t)$ für die mit i verbundenen Atome j gilt:

$$\vec{\mathbf{g}}_i(t) = -\sum_j \lambda_{ij}(t)\nabla_j\sigma_{ij} \tag{4.19}$$

Aus Gleichung (4.14) ergibt sich eine neue Gleichung zur Ortsbestimmung, die durch die Nebenbedingung $\sigma_{ij} = \sigma_{ji}$ und der daraus für die Lagrangemultiplikatoren der Ortsgleichung λ_{rij} folgenden Bedingung $\lambda_{rij} = \lambda_{rji}$ vereinfacht wird

zu:

$$\vec{\mathbf{r}}_i(t+\delta t) = \vec{\mathbf{r}}_i(t) + \delta t\vec{\mathbf{v}}_i(t) + \frac{\delta t^2}{2m_i}\left(\vec{\mathbf{F}}_i(t) - 2\sum_j \lambda_{rij}(t)\vec{\mathbf{r}}_{ij}(t)\right) \tag{4.20}$$

Die Lagrangemultiplikatoren $\lambda_{rij}(t)$ sind so zu wählen, dass Gleichung (4.16) erfüllt wird. Die Geschwindigkeitsgleichung (4.15) ändert sich durch Gleichung (4.19) entsprechend, so dass sich für die Zwangskraft der Geschwindigkeit $\vec{\mathbf{g}}_{vi}(t+\delta t)$ ergibt:

$$\vec{\mathbf{g}}_{vi}(t+\delta t) = -2\sum_i \lambda_{vij}(t+\delta t)\vec{\mathbf{r}}_{ij}(t+\delta t) \tag{4.21}$$

Zur Bestimmung der $\lambda_{rij}(t)$ und $\lambda_{vij}(t+\delta t)$ wird das von Ryckaert u. a. [41] und von Andersen [42] beschriebene Iterationsverfahren verwendet. Für das Iterationsverfahren werden definiert:

$$g_{ij} = \delta t\lambda_{rij}(t) \tag{4.22}$$

$$k_{ij} = \delta t\lambda_{vij}(t) \tag{4.23}$$

$$\vec{\mathbf{q}}_i(t) = \vec{\mathbf{v}}_i(t) + \frac{\delta t}{2m_i}\vec{\mathbf{F}}_i(t) - \frac{1}{m_i}\sum_j g_{ij}\vec{\mathbf{r}}_{ij}(t) \tag{4.24}$$

Die Ortsgleichungen (4.14 / 4.20) und die Geschwindigkeitsgleichungen (4.15) zusammen mit Gleichung (4.21) können dann geschrieben werden als:

$$\vec{\mathbf{r}}_i(t+\delta t) = \vec{\mathbf{r}}_i(t) + \delta t\vec{\mathbf{q}}_i(t) \tag{4.25}$$

$$\vec{\mathbf{v}}_i(t+\delta t) = \vec{\mathbf{q}}_i(t) + \frac{\delta t}{2m_i}\vec{\mathbf{F}}_i(t+\delta t) - \frac{1}{m_i}\sum_j k_{ij}\vec{\mathbf{r}}_{ij}(t+\delta t) \tag{4.26}$$

Die $\vec{\mathbf{q}}_i$ werden durch Iteration wie folgt bestimmt:
Das Iterationsverfahren startet mit dem Startwert:

$$\vec{\mathbf{q}}_i(t) = \vec{\mathbf{v}}_i(t) + \frac{\delta t}{2m_i}\vec{\mathbf{F}}_i(t) \tag{4.27}$$

Bei einer Zwangsbedingung zwischen einem Atom i und einem Atom j existiert folgender Abstandsvektor $\vec{\mathbf{s}}$:

$$\vec{\mathbf{s}} = \vec{\mathbf{r}}_i(t) + \delta t\vec{\mathbf{q}}_i(t) - \vec{\mathbf{r}}_j(t) - \delta t\vec{\mathbf{q}}_j(t) \tag{4.28}$$

Weicht die Differenz des Abstandsquadrats $|\vec{\mathbf{s}}|^2$ vom vorgegebenen Abstandsquadrat d_{ij}^2 um mehr als ein vorgegebenes $\epsilon < |\vec{\mathbf{s}}|^2 - d_{ij}^2$ ab, so müssen $\vec{\mathbf{q}}_i(t)$ und $\vec{\mathbf{q}}_j(t)$ in einem Iterationsschritt korrigiert werden. Für die temporären Ortsvektoren $\vec{\mathbf{r}}_i^*$ und $\vec{\mathbf{r}}_j^*$ der Atome gilt:

$$\vec{\mathbf{r}}_i^* = \vec{\mathbf{r}}_i(t) + \delta t\left(\vec{\mathbf{q}}_i(t) - \frac{g}{m_i}\vec{\mathbf{r}}_{ij}(t)\right) \tag{4.29}$$

$$\vec{\mathbf{r}}_j^* = \vec{\mathbf{r}}_j(t) + \delta t\left(\vec{\mathbf{q}}_j(t) + \frac{g}{m_j}\vec{\mathbf{r}}_{ij}(t)\right) \tag{4.30}$$

Hierbei muss g so gewählt werden, dass $\left|\vec{\mathbf{r}}_i^* - \vec{\mathbf{r}}_j^*\right|^2 = d_{ij}^2$, d. h. für g gilt näherungsweise:

$$g = \frac{|\vec{\mathbf{s}}|^2 - d_{ij}^2}{2\delta t\big(\vec{\mathbf{s}}\ \vec{\mathbf{r}}_{ij}(t)\big)\left(m_i^{-1} + m_j^{-1}\right)} \tag{4.31}$$

Daraus werden die neuen Werte für $\vec{\mathbf{q}}$ bestimmt:

$$\vec{\mathbf{q}}_i = \vec{\mathbf{q}}_i - \frac{g}{m_i}\vec{\mathbf{r}}_{ij}(t) \qquad \vec{\mathbf{q}}_j = \vec{\mathbf{q}}_j + \frac{g}{m_j}\vec{\mathbf{r}}_{ij}(t) \tag{4.32}$$

und die temporären Ortsvektoren werden als neue Orte festgelegt:

$$\vec{\mathbf{r}}_i(t+\delta t) = \vec{\mathbf{r}}_i^* \qquad \vec{\mathbf{r}}_j(t+\delta t) = \vec{\mathbf{r}}_j^* \tag{4.33}$$

Weicht das Abstandsquadrat weiterhin um mehr als das vorgegebene ϵ ($|\vec{\mathbf{s}}|^2 - d_{ij}^2 > \epsilon$) vom vorgegebenen Abstandsquadrat d_{ij}^2 ab, so wird ein neuer Iterationsschritt gestartet.

Für die Geschwindigkeiten wird ebenfalls ein Iterationsverfahren verwendet. Der Startwert wird hier folgendermaßen festgelegt:

$$\vec{\mathbf{v}}_i(t+\delta t) = \vec{\mathbf{q}}_i(t) + \frac{\delta t}{2m_i}\vec{\mathbf{F}}_i(t+\delta t) \tag{4.34}$$

Bei einer Zwangsbedingung zwischen Atom i und j müssen die Geschwindigkeiten ebenfalls korrigiert werden, falls $\vec{\mathbf{r}}_{ij}(t+\delta t)\ \vec{\mathbf{v}}(t+\delta t) > \epsilon'$. Wie bei der Ortsbestimmung wird auch hier ein temporärer Wert mit Hilfe eines Gleichgewichtsfaktors k bestimmt:

$$\vec{\mathbf{v}}_i^* = \vec{\mathbf{v}}_i(t+\delta t) - \frac{k}{m_i}\vec{\mathbf{r}}_{ij}(t+\delta t) \tag{4.35}$$

$$\vec{\mathbf{v}}_j^* = \vec{\mathbf{v}}_j(t+\delta t) + \frac{k}{m_j}\vec{\mathbf{r}}_{ij}(t+\delta t) \tag{4.36}$$

In jedem Fall muss $\vec{\mathbf{v}}_i^* - \vec{\mathbf{v}}_j^*$ senkrecht auf $\vec{\mathbf{r}}_{ij}(t+\delta t)$ stehen, um Gleichung (4.18) zu erfüllen, für k gilt somit:

$$k = \frac{\vec{\mathbf{r}}_{ij}(t+\delta t)\Big(\vec{\mathbf{v}}_i(t+\delta t) - \vec{\mathbf{v}}_j(t+\delta t)\Big)}{d_{ij}^2\left(m_i^{-1} + m_j^{-1}\right)} \tag{4.37}$$

Abschließend werden die Geschwindigkeiten der temporären Werte übernommen, die dann durch neue Iteration weiter verbessert werden können:

$$\vec{\mathbf{v}}_i(t+\delta t) = \vec{\mathbf{v}}_i^* \qquad \vec{\mathbf{v}}_j(t+\delta t) = \vec{\mathbf{v}}_j^* \tag{4.38}$$

4.3 Thermostate und Barostate

Einfache MD-Simulationen laufen im mikrokanonischen oder kurz NVE-Ensemble, d. h. die Anzahl der Atome N, das Volumen V und die Gesamtenergie E des simulierten Systems sind während der Simulation konstant. Da während einer Simulation weder Atome hinzugefügt noch entfernt werden und der Simulationsraum nicht geändert wird, ist die Konstanz der ersten beiden Parameter N und V offensichtlich. Vorausgesetzt, es gibt keine Wechselwirkungen mit externen Feldern und keine Ungenauigkeiten während der Simulation, führt die Abgeschlossenheit des Systems zur Konstanz der Energie E.

Während einer Simulation wird zwischen potentieller Energie und kinetischer Energie ständig Energie ausgetauscht, die Gesamtenergie bleibt jedoch Aufgrund der Energieerhaltung konstant. Dieses ideale Bild lässt sich bei einer Computersimulation nicht aufrecht erhalten. Durch die begrenzte Genauigkeit der Variablen und durch Fehler in der Integration der Bewegungsgleichung gibt es in der Regel eine leicht schwankende Gesamtenergie.

Die hohe potentielle Energie des von „structC“ erzeugten Startsystems, würde in einem NVE-Ensemble, wegen starker Beschleunigungen, die Atome nach kurzer Zeit auf sehr hohen kinetischen Energien bringen. Das System würde damit auf eine sehr hohe Temperatur aufgeheizt, aus diesem Grund muss dem System in der Anfangsphase Energie entzogen werden. Die Simulation erfolgt in der Equilibrierungsphase deshalb nicht im NVE-Ensemble, sondern im NVT-Ensemble, worin, anstatt der Gesamtenergie E, die kinetische Energie und damit die Temperatur T konstant gehalten wird. Um die Temperatur T bzw. kinetische Energie E_{kin} konstant zu halten, werden die Geschwindigkeiten in jedem Simulationsschritt mit einem von E_{kin} abhängigen Faktor S skaliert. Das bedeutet, die Geschwindigkeiten v_i aller Atome i werden um den selben Faktor S geändert, dieser korrigiert dadurch die Temperatur T des Systems in Richtung der vorgegebenen Referenztemperatur T_{ref}.

In der Equilibrierungsphase, in der ein starker Energieumsatz von potentieller in kinetischer Energie erfolgt, hat sich das sehr radikale Verfahren von Woodcock [4] bewährt. Es skaliert die Geschwindigkeiten v_i in jedem Zeitschritt so, dass die Referenztemperatur T_{ref} exakt erreicht wird. Für die Bestimmung des Faktors S ist die momentane Temperatur T des Systems erforderlich, sie lässt sich mit Hilfe seines Freiheitsgrades f und der Boltzmannkonstanten k_B aus der kinetischen Energie E_{kin} berechnen, es gilt:

$$\begin{aligned} E_{kin} &= \frac{1}{2}\sum_{i=1}^{N} m_i v_i^2 \\ &= f k_b T \end{aligned} \tag{4.39}$$

Jedes Atom i besitzt eine feste Masse m_i und eine momentane Geschwindigkeit v_i. Ferner hat es in jede Raumrichtung einen Freiheitsgrad, diese drei

Freiheitsgrade können durch Zwangskräfte teilweise oder vollständig wieder aufgehoben werden. Durch die periodischen Randbedingungen ist die absolute Lage eines Atoms irrelevant, nur die relative Lage der Atome untereinander ist entscheidend. In einem System mit periodischen Randbedingungen kann deshalb der Massenschwerpunkt des Gesamtsystems immer in den Koordinatenursprung verlegt werden, die Anzahl der Freiheitsgrade f im Gesamtsystem verringern somit um die drei Raumrichtungen des Massenschwerpunkts. Für ein Systems aus N Atomen mit periodischen Randbedingungen und N_{cf} Zwangskräften ergibt sich somit eine Gesamtanzahl an Freiheitsgraden f von:

$$f = 3N - N_{cf} - 3 \tag{4.40}$$

Aus Gleichung 4.39 lässt sich eine Gleichung ableiten, die ein System mit Temperatur T in ein System mit Temperatur T_N transferiert. Hierbei hängt T_N über einen Skalierungsfaktor S mit T zusammen, es gilt:

$$\begin{aligned} T_N &= S^2 T && (4.41)\\ &= S^2 \frac{1}{2fk_B} \sum_i m_i v_i^2 \\ &= \frac{1}{2fk_B} \sum_i m_i \left(S v_i\right)^2 && (4.42) \end{aligned}$$

Um die Temperatur T eines Systems auf eine neue Temperatur T_N zu ändern, werden die Geschwindigkeiten v_i aller Atome i durch den Skalierungsfaktor S geändert:

$$\vec{\mathbf{v}}_i \leftarrow S\vec{\mathbf{v}}_i \qquad \forall \qquad i \in 1, .., N \tag{4.43}$$

Der oben beschriebene Thermostat von Woodcock [4] ändert die Temperatur T des Systems in nur einem Simulationsschritt auf die vorgegebene Referenztemperatur T_{ref}. Dies lässt sich durch eine einfache Umformung von Gleichung (4.41) erreichen, die zu einem Skalierungsfaktor S führt von:

$$S = \sqrt{\frac{T_{ref}}{T}} \tag{4.44}$$

Eine wesentlich realistischere Modellierung eines Wärmebades bietet der Berendsen Thermostat [43], dieser ändert die Geschwindigkeiten im allgemeinen nicht so abrupt wie der Woodcock Thermostat. Die Geschwindigkeit des Energieflusses in bzw. aus dem System wird hier zusätzlich durch eine Relaxationszeit τ_B beeinflusst. Der Berendsen Thermostat verwendet zur Skalierung der Geschwindigkeiten folgenden Skalierungsfaktor S:

$$S = \sqrt{1 + \frac{\delta t}{\tau_B} \left(\frac{T_{ref}}{T} - 1 \right)} \tag{4.45}$$

Mit Hilfe der Relaxationszeit τ_B lässt sich der Einfluss des Thermostaten auf das System steuern. Er reicht von einem dem Woodcock Thermostaten äquivalenten Verhalten bei einer Relaxationszeit $\tau_B = \delta t$, die der Schrittweite δt entspricht, bis hin zu keinem Einfluss bei $\tau_B \to \infty$. Für genauere Betrachtungen passt man Gleichung (4.41) an den Berendsen Thermostaten an. Die Anwendung des Berendsen Thermostaten liefert nicht mehr die Referenztemperatur T_{ref}, sondern eine neue Temperatur T_N, die zwischen der Temperatur T des Systems nach dem Integrationsschritt und der zu erreichenden Referenztemperatur T_{ref} liegt. Die Temperatur T_N erhält man durch Einsetzen der Gleichung (4.45) in Gleichung (4.41) und einer anschließenden Umformung, die zu folgender Gleichung führt:

$$\begin{aligned} T_N &= \left(1 + \frac{\delta t}{\tau_B}\left(\frac{T_{ref}}{T} - 1\right)\right) T \\ &= T + \frac{\delta t}{\tau_B} T_{ref} - \frac{\delta t}{\tau_B} T \\ &= \frac{\delta t}{\tau_B} T_{ref} + \left(1 - \frac{\delta t}{\tau_B}\right) T \end{aligned} \tag{4.46}$$

Diese Gleichung zeigt, dass der Faktor $\delta t/\tau_B$ eine Gewichtung zwischen der Temperatur T nach dem Integrationsschritt und der Referenztemperatur T_{ref} vornimmt um den neuen Temperaturwert T_N zu bestimmen.

Verwendung findet der Berendsen Thermostat in erster Linie mit langen Relaxationszeiten τ_B, dort hält er die Auswirkungen der Thermalisierung auf die statischen, und vor allem, die diesbezüglich sehr empfindlichen dynamischen Größen klein. Ist die Relaxationszeit größer als 0.1 ps sind Auswirkungen auf das System nicht mehr nachweisbar [43]. In einem NVT oder NPT Ensemble ist deshalb außerhalb der Equilibrierungsphase der Berendsen Thermostat zu verwenden. Neben den beiden hier vorgestellten Thermostaten gibt es eine Reihe weiterer Thermostate [44, 45, 46, 47], die hier nicht weiter betrachtet werden sollen.

Mit den selben Ansätzen wie bei den Thermostaten ist es ebenfalls möglich den Druck konstant zu halten, um damit zum Beispiel die Simulation eines NPT Ensembles zu ermöglichen. Die hierfür verwendeten Barostate unterscheiden sich von den Thermostaten nur durch ein Wechsel der zu korrigierenden Größen. Anstatt durch Änderung der Geschwindigkeiten bzw. der kinetischen Energie die Temperatur T auf einer Referenztemperatur T_{ref} zu halten, wird bei den Barostaten der Druck p durch Skalierung des Volumens V konstant gehalten. Gleichzeitig mit der Skalierung des Volumens V müssen auch die Schwerpunkte der Moleküle mit einem entsprechenden Skalenfaktor verschoben werden, keines falls dürfen dadurch aber die Moleküldurchmesser oder die Abstände der Atome innerhalb der Moleküle verändert werden. Die mit der Änderung des Volumens verbundene Verschiebung der Moleküle führt zu einer Verringerung bzw. Vergrößerung der Abstände und somit zu einer Druckerhöhung bzw. Druckverminderung im System.

Bei den hier verwendeten Simulation von NIPAM und PAA wird das Polymer an den Raumgrenzen periodisch fortgesetzt, bei einer Änderung des Volumens V kann das Polymer der Volumenänderung wegen seiner Größe nicht folgen, wie dies kleine Moleküle durch die Verlagerung des Schwerpunktes mit Hilfe des Skalenfaktors können. Eine Volumenänderung im Polymer / Gel Systemen führt unweigerlich zur Verzerrung von Atomabständen innerhalb des Polymers, da die einzige Möglichkeit des Polymers sich einer Volumenänderung anzupassen darin besteht, den Abstand der Atombindungen zu ändern. Die Simulation von Polymeren bei festem Druck p ist somit nur für Polymere aus wenigen Monomeren sinnvoll.

4.4 Simulation mit „dynC“

Die in den letzten Abschnitten beschriebenen Methoden wurden im Simulationsprogramm „dynC“ implementiert. Zur Simulation wird zum Einen eine Startstruktur benötigt, die den Startpunkt im Phasenraum bestimmt, und zum Anderen eine Skript-Datei, die sämtliche Randparameter der Simulation enthält und mit denen der Verlauf der Simulation festgelegt wird. Die Startstruktur kann entweder aus einem bereits simulierten System bestehen, oder wird wie in den Abschnitten 3.2 und 3.4 beschrieben, aus den Kraftfeldern und den Daten der Molekülstruktur zufällig erzeugt.

Analog zur Aufteilung der Struktur- und Kraft-Datenbank in Abschnitte besteht auch die Steuerungsdatei aus einem Abschnitt. Im Abschnitt „[Simulation]“ erfolgt die Steuerung des Ablaufs einer Simulation durch ein Ablaufskript. Der grundsätzliche Aufbau dieses Abschnittes soll am Beispiel des in Abbildung 4.1 gezeigten Simulationsabschnittes einer Equilibrierungssimulation vorgestellt werden.

Die im Skript verwendeten Kommandos lassen sich in vier Kategorien einteilen. In der ersten Kategorie werden die Parameter und Randbedingungen für die Potentiale festgelegt. Der cutoff-Radius wird in der Datenbank durch **LJCutOff** (Zeile 3) bestimmt. Ist der Abstand zwischen zwei Atomen größer als der cutoff-Radius, so wird deren van der Waals-Wechselwirkung nicht mehr berücksichtigt. Für den Realteil der elektrostatischen Wechselwirkung existiert ein analoger cutoff-Radius, dieser wird, wie in Zeile 4 zu sehen, mit dem Kommando **CCutOff** festgelegt. Ergänzt wird er durch einen Abschirmfaktor κ (**Kappa**) und einen, den maximalen Radius des reziproken Teils festlegenden, Parameter **Kmax** für das Ewaldsummenverfahren in den Zeilen 6 und 7. Die Stärke der elektrostatischen Bindungen wird durch den Dielektrikumsfaktor **Diel** bestimmt, der mit $(4\pi\epsilon_0\epsilon_r)^{-1}$ im Vakuum ($\epsilon_r = 1$) bei $332{,}063\,{}^{\text{kcal}\,\text{Å}}/_{\text{mol}\,e^2}$ liegt. Außerdem kann mit den Anweisungen **Coulomb** und **Ewald** zwischen einer klassischen Bestimmung der elektrostatischen Wechselwirkungen und einer Berechnung nach dem Ewaldsummenverfahren gewählt werden. Die in der Kraft-Datenbank vorgege-

```
1  [Simulation]
   debug=30
   LJCutOff 10.0
   CCutOff 10.0
5  Diel 332.0636
   Kappa 6.0
   Kmax 5
   in dyn NIPAM.dyn
   ;Coulomb
10 Temp 300
   ;TempBer 300 0.5
   Step 3e-6
   constraint 1e-5 3500
   ;reset timer
15 ;report new
   out dyn 500 NIPAM1.dyn
   report new
   out report 1 NIPAMrep.txt
   out Trajold 10 NIPAM
20 simu 10
   equil 2e-3 500 600
```

Abbildung 4.1: Equilibrierungssimulation der P-NIPAM-Struktur.

benen Zwangskräfte werden über zwei Parameter geregelt. Die Toleranzschwelle legt fest, wie groß die Abweichung der Bindungslänge vom Sollwert sein darf, bevor diese durch ein Constraint-Verfahren (RATTLE oder SHAKE) korrigiert werden muss. Der zweite Parameter bestimmt die Obergrenze der Iterationen, durch den gegebenenfalls mit einer Fehlermeldung abgebrochen wird. Festgelegt werden beide Parameter, dessen Bedeutung bereits im Abschnitt 4.2 beschrieben wurde, mit dem in Zeile 13 stehenden Kommando **constraint**.

Die zweite Gruppe bilden die äußeren Randbedingungen, sie bestehen aus der Größe des Simulationsraums, definiert in **BoxSize**, und der Temperatur, letztere wird in **Temp** bzw. **TempBer** festgelegt. Die Temperatur wird entweder, wie in Zeile 10 durch Woodcocks [4] Algorithmus auf eine feste Temperatur fixiert oder flexibler durch den Berendsen Thermostat [43], der in Zeile 11 neben der Ziel-Temperatur, die Relaxationszeit als weiteren Parameter benötigt. Beide Algorithmen wurden bereits im Abschnitt 4.3 beschrieben.

Die letzte Gruppe von Parametern legt die Dauer der Simulationen fest. Sie bestimmen die zu Beginn gewählte Schrittweite des Integrationsverfahrens (**Step**) in ps und die Anzahl der Simulationsschritte (**simu**). Muss das System durch eine Equilibrierungsphase in ein Gleichgewichtssystem überführt werden, wird dies ebenfalls hier festgelegt. Der Befehl **equil** erfordert als ersten Parameter die Schrittweite in ps, die am Ende der Equilibrierungsphase erreicht werden muss. Die beiden weiteren Parameter legen die obere und untere Grenze (in Kelvin) des Temperaturkorridors fest, in dem sich das System während der Simulation befinden muss, siehe hierzu Zeile 21 in Abbildung 4.1. Der Algorithmus zur Schrittweitenkontrolle verwendet diesen Temperaturkorridor zur Bestimmung der maximalen Schrittweite an jedem Zeitpunkt.

Ergänzt werden diese Kommandos durch Anweisungen, die die Simulation selbst nicht beeinflussen, sondern der Protokollierung und Speicherung der gewonnen Daten dienen. In der Regel werden in der Produktionsphase drei Dateien erzeugt. Die Report-Datei (Abbildung 4.2), erzeugt mit **out report** (Zeile 18), speichert neben den Randparametern in festen Abständen die Temperatur und die Energieanteile der einzelnen Wechselwirkungspotentiale. Diese ist insbesondere für die Überwachung der Equilibrierungsphase wichtig. Da eine Simulation in der Regel aus mehreren Programmdurchläufen besteht, wird eine bestehende Report-Datei fortgesetzt, es sei denn man erzwingt einen Neuanfang durch die Anweisung **report new**.

In bestimmten Abständen sollte die aktuelle Konfiguration des Systems mit dem aktuellen Phasenraumpunkt in eine Datei gespeichert werden. Nach einem Ausfall des Rechners, kann dann ab der zuletzt gespeicherten Konfiguration fortgesetzt werden. Zudem ist es möglich mit den gespeicherten Konfigurationen bestimmte Abschnitte in kleinerer Schrittweite neu zu simulieren und diese dann mit einer höheren Auflösung zu untersuchen. Die aktuelle Konfiguration wird wie in Zeile 16 mit der Anweisung **out dyn** gesteuert, hierbei legt der erste Parameter das Speicherintervall fest indem die Konfiguration in die im zweiten Parameter angegebene Datei gespeichert wird.

Die dritte Datei wird von der Anweisung **out traj** erzeugt, sie ist für die anschließende Auswertung die wichtigste Datei und speichert in definierten Abständen die Phasenraumpunkte des simulierten Systems. D. h. sie enthält die Orte und Geschwindigkeiten eines jeden Atoms zu jedem gespeicherten Zeitpunkt. Aus diesen Daten werden dann in der Auswertungsphase zum Beispiel Korrelationen und Häufigkeitsverteilungen bestimmt.

Bevor mit der Simulation begonnen werden kann, muss die Strukturdatei (DYN) geladen werden, diese enthält unter anderem eine Atomliste, die wiederum den Namen des Atoms, seine Ladung, seinen Ortsvektor, so wie seinen Geschwindigkeitsvektor enthält. Die Struktur der Moleküle wird durch eine ebenfalls in der Strukturdatei enthaltene Konnektivitätsliste beschrieben, die für jedes Atom die bindenden Nachbarn erhält. Zudem enthält die Strukturdatei die Kraftfeldparameter eines jeden Atoms. Aus der Konnektivitätsliste und den Kraftfeldparametern werden zur Geschwindigkeitsoptimierung vor der Simulation Berechnungslisten erzeugt.

Zur Geschwindigkeitsoptimierung wird mit Hilfe der Konnektivitätsliste die Liste der gebundenen Wechselwirkungen bestimmt, in der die Paarwechselwirkungen, die Biegeschwingung, die Torsionswechselwirkungen und die uneigentlichen Torsionswechselwirkungen eingehen. Weiter werden Listen zur Optimierung der Berechnung der weitreichenden Wechselwirkungen erzeugt. Diese sind besonders wichtig, da für die Berechnung der weitreichenden Kräfte über 90% der Rechenzeit benötigt wird. In die allgemeine Liste der van der Waals-Wechselwirkungen werden die Atompaare nicht aufgenommen, bei denen mindestens eins der beiden Atome keine van der Walls-Wechselwirkung besitzt oder beide Atome Teil der

```
#Simulation DynC Ver. 2.0

#Molecules = 526
#Atoms = 1768
#Box size = 27.000000 A
#Density = 1.000446 g/cm^3
#Temperature  = 300.000000 K
#TimeStep = 0.000003 ps
#Lenard-Jones cut off = 10.000000 A
#Ewald sum parameters:
#Coulomb cut off = 10.000000
#Dielectricum = 332.063600
#Kappa = 6.000000
#Kmax = 5

#Constraint parameters:
#Tolerance = 0.000010
#Max iterations = 3500

#Output:
#DYN every 500 steps in NIPAM1.dyn
#Report every 1 steps in NIPAMrep.txt
#Trajectory every 0 steps in (null)
#Old Trajectory every 10 steps in NIPAM
# Total Mass:11860.357300
# POLY = 20.254637 %
# Water = 79.745363 %

#   Step           Time    Temp          Ekin            VW        Coulomb          Bond          Epot          Eges
       1  3.000000e-006     1.0  5.958912e-015  3.902242e-010 -8.919605e-010  2.783242e-012 -4.989531e-010 -4.989471e-010
       2  6.000000e-006   351.2  2.145964e-012  3.898058e-010 -8.919609e-010  2.783197e-012 -4.993719e-010 -4.972260e-010
       3  9.000000e-006   369.0  2.254218e-012  3.893805e-010 -8.919613e-010  2.783151e-012 -4.997976e-010 -4.975434e-010
       4  1.200000e-005   370.3  2.262270e-012  3.889561e-010 -8.919608e-010  2.783105e-012 -5.002215e-010 -4.979593e-010
       5  1.500000e-005   369.8  2.259154e-012  3.885325e-010 -8.919601e-010  2.783059e-012 -5.006445e-010 -4.983853e-010
       6  1.800000e-005   369.4  2.256983e-012  3.881102e-010 -8.919603e-010  2.783013e-012 -5.010671e-010 -4.988101e-010
       7  2.100000e-005   369.4  2.256707e-012  3.876889e-010 -8.919610e-010  2.782968e-012 -5.014891e-010 -4.992324e-010
       8  2.400000e-005   369.1  2.254818e-012  3.872683e-010 -8.919608e-010  2.782922e-012 -5.019096e-010 -4.996548e-010
       9  2.700000e-005   368.8  2.253477e-012  3.868485e-010 -8.919603e-010  2.782876e-012 -5.023289e-010 -5.000755e-010
      10  3.000000e-005   368.8  2.253292e-012  3.864293e-010 -8.919601e-010  2.782830e-012 -5.027480e-010 -5.004947e-010
      11  3.300000e-005   369.1  2.255230e-012  3.860108e-010 -8.919607e-010  2.782785e-012 -5.031671e-010 -5.009118e-010
      12  3.600000e-005   368.5  2.251665e-012  3.855938e-010 -8.919610e-010  2.782739e-012 -5.035844e-010 -5.013328e-010
      13  3.930000e-005   368.4  2.250614e-012  3.851381e-010 -8.919605e-010  2.782689e-012 -5.040397e-010 -5.017891e-010
```

Abbildung 4.2: Anfang der Reportdatei einer Equilibrierungssimulation.

selben, in Abschnitt 2.1 beschriebenen, intramolekularen Wechselwirkung sind. Ergänzt wird die allgemeine Liste der van der Waals-Wechselwirkungen durch eine Liste der dritten Nachbarn, das sind die Atompaare deren intramolekulare Wechselwirkungen sich auf eine Torsionswechselwirkung beschränken. Für diese Atompaare gibt es im van der Waals-Potential einen speziellen Parametersatz, sie werden deshalb getrennt von den anderen van der Waals-Wechselwirkungen bestimmt. Für die Coulomb-Wechselwirkung gelten ähnliche Einschränkungen wie für die van der Waals-Wechselwirkung, da aber nicht zwischen den dritten Nachbarn und den anderen Atomen unterschieden wird, reicht eine Wechselwirkungsliste zur Optimierung der Kraftberechnung. Der Rechenaufwand für die Berechnung des Wassers verringert sich durch diese Listenbildung zum Beispiel auf etwa 1/9 des ursprünglichen Aufwands. Vorab werden zudem die Vektoren des reziproken Raumes der Ewaldsumme bestimmt. Diese Vektoren werden durch die Größe des Simulationsraumes festgelegt, der im NVT-Ensemble unverändert bleibt. Unter Berücksichtigung der periodischen Randbedingungen, der Anzahl der Atome und der Zwangsbedingungen lässt sich ebenfalls der Freiheitsgrad f des Systems durch Gleichung (4.40) vorab bestimmen.

Nach diesen vorbereitenden Optimierungsschritten wird mit der eigentlichen Simulation begonnen. Die Simulation wird in Simulationsschritte mit definierter Länge eingeteilt. Jeder Simulationsschritt besteht aus einem Integrationsschritt, der wiederum aus Kraft- und Potentialberechnungen und gegebenenfalls aus einer Thermalisierung des Systems besteht. Im hier verwendeten Fall werden die Zwangskräfte, die im allgemeinen nur auf die leichten Wasserstoffatome wirken, durch den in Abschnitt 4.2 beschriebenen RATTLE Algorithmus in das System integriert. Dieser verwendet das bereits in 4.1 behandelte velocity Verlet Verfahren zur Integration der Bewegungsgleichungen. Aufgrund des velocity Verlet Anteils muss der Integrationsschritt in einen Anteil vor und einen Anteil nach der Kraftberechnung separiert werden, wie es schon das Ablaufschema auf Seite 38 gezeigt wurde.

Ein Simulationsschritt wird durch die Bestimmung der neuen Ortsvektoren der Atome im ersten Teil der Integrationsroutine eingeleitet. Mit den neuen Koordinaten werden die jetzt auf die Atome wirkenden Kräfte und die dazu gehörigen Potentiale bestimmt. Die Bestimmung der Kräfte und Potentiale erfolgt in drei Teilschritten. Zuerst werden die in Abschnitt 2.1 eingeführten kurzreichweitigen Wechselwirkungen berechnet, dann die van der Waals-Kräfte und Potentiale aus Abschnitt 2.2.1, die durch die Coulomb-Wechselwirkungen aus Abschnitt 2.2.3 ergänzt werden. Die Coulomb-Wechselwirkungen werden je nach Einstellung entweder klassisch, mit einem begrenzenden cutoff-Radius, bestimmt oder durch das Ewaldsummenverfahren, das die elektrostatischen Kräfte wesentlich genauer berechnet. Zusammen führen alle Wechselwirkungen unter Berücksichtigung von N_b Streckschwingungen, N_θ Biegeschwingung, N_ϕ Torsionswinkeln und N_ϵ uneigent-

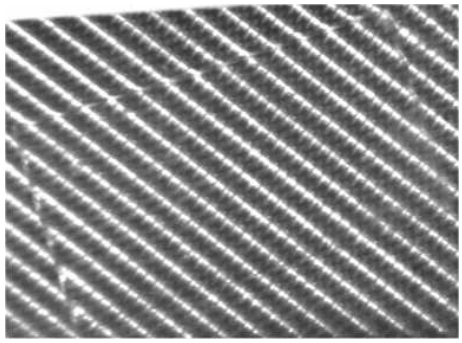
lichen Torsionswinkeln zu folgendem Gesamtpotential U:

$$\begin{aligned} U &= \sum_i \sum_j \left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} \right) + \sum_i \sum_j \frac{q_i q_j}{4\pi\epsilon_0\epsilon_r r_{ij}} \\ &+ \frac{1}{2} \sum_{N_b} K_b (r - r_0)^2 + \frac{1}{2} \sum_{N_\theta} K_\theta (\theta - \theta_0)^2 \\ &+ \frac{1}{2} \sum_{N_\phi} K_\phi \big(1 + \cos\left(m\phi - \phi_0\right)\big) + \frac{1}{2} \sum_{N_\epsilon} K_\epsilon (\epsilon - \epsilon_0)^2 \end{aligned} \tag{4.47}$$

Nach Berechnung des Potentials und der daraus resultierenden Kräfte erfolgt der zweite Teil des Integrationsschrittes. In ihm werden die Kräfte auf die Geschwindigkeitsvektoren angewendet. Im NVE-Ensemble schließt er auch den Simulationsschritt ab, d. h. das System hat in der Simulationsphase einen neuen Phasenraumpunkt erreicht. Nun können Systemdaten, falls ein mit den Speicherintervallen festgelegter abzuspeichernder Simulationsschritt erreicht worden ist, in Report-, Trajektorien-, Struktur- oder PDB-Datei geschrieben werden. Befindet sich das System in der Equilibrierungsphase, so wird zusätzlich die Schrittweite der Simulation an die Stärke der Temperaturveränderung angepasst. Eine starke Abweichung der Ist-Temperatur von der Soll-Temperatur nach oben verringert die Schrittweite, eine entsprechende Abweichung nach unten vergrößert sie. In der Equilibrierungsphase oder in einem Ensemble mit fester Temperatur T folgt dem Integrationsschritt ein Thermalisierungsschritt, dabei wird die Ist-Temperatur des Systems durch einen der beiden im Abschnitt 4.3 beschriebenen Thermostaten in Richtung Soll-Temperatur korrigiert. Nach jedem Simulationsschritt beginnt wieder ein neuer, bis der letzte durch das Skript festgelegte Simulationsschritt ausgeführt wurde und das Programm beendet wird.

Kapitel 5

Gele



Im ersten Abschnitt werden die physikalischen Eigenschaften von Gelen und deren Anwendungsgebiete in Natur und Technik, insbesondere das Verhalten von Poly-N-Isopropylacrylamid (NIPAM) beschrieben. Der zweite Abschnitt beschäftigt sich mit der radikalischen Polymerisation von Polymeren und der Herstellung von Gelen und deren Polymernetzwerken mit Hilfe von Quervernetzern. Die Beschreibung der Strukturbildung von Gelen erfolgt im dritten Abschnitt, worin auch erläutert wird, wie ein Gel durch eine künstliche „Synthese“ erzeugt wird, die sich am natürlichen Syntheseprozess orientiert. Die generierte Struktur bildet eine Startstruktur für den im letzten Abschnitt beschriebenen Ablauf einer Simulation, die sich in eine Equilibrierungsphase und eine Produktionsphase aufspaltet.

5.1 Die Natur der Gele

Gele sind Stoffe, die aus zwei Komponenten bestehen, aus einem festen Polymergerüst und aus einer Flüssigkeit, dem Lösungsmittel. Beide Bestandteile bestimmen das Verhalten des Gels. Das Gerüst selbst besteht aus einem Netzwerk von Polymerketten, dessen Steifigkeit vom Vernetzungsgrad der Polymere abhängt. Es sorgt für die Stabilität des Gels und hindert das Lösungsmittel am Zerfließen. Das Lösungsmittel, das sich zwischen dem Polymergerüst befindet, verhindert im Gegenzug das Kollabieren des Polymergerüstes und sorgt für die Viskoelastizität des Polymers. Eine Unterklasse der Gele bilden die Hydrogele, bei ihnen wird Wasser als Lösungsmittel verwendet. Die Wechselwirkungen zwischen dem Polymernetzwerk und dem Lösungsmittel verleihen einem Gel seine charakteristischen Eigenschaften, die zum Beispiel dafür sorgen, dass das Gel sich in dem ihm typischen Grenzzustand zwischen Festkörper und Flüssigkeit befindet. Obwohl die Flüssigkeit der Hauptbestandteil eines Gels ist, ergibt sich für ein Gel ein mehr oder weniger fester makroskopischer Zustand, der sein elastisches Verhalten auszeichnet.

Auf mikroskopischer Ebene herrschen jedoch viskoelastische Eigenschaften vor. Nicht nur der Aufbau der Gele bestimmt deren Festigkeit und Größe, wesentlich sind auch äußere Parameter, die die inneren Kräfte des Gels beeinflussen. Bei Störung des Kräftegleichgewichts zwischen Polymergerüst und Lösungsmittel durch Variation der Randparameter, versucht das Gel dieser Störung durch Schrumpfen oder Quellen entgegenzuwirken, und so das Gleichgewicht wieder herzustellen. Durch den Schrumpfungs- bzw. Quellprozess wird Lösungsmittel aus dem Polymerverband gedrängt bzw. aufgenommen. Die Änderungen im Kräftegleichgewicht sind bei einigen Polymeren so drastisch, dass zur Wiedererlangung des Gleichgewichts ein Phasenübergang induziert wird, durch den sich das Volumen eines Gels um mehr als das Hundertfache [48] ändern kann. Hierbei reagieren die Gele insbesondere auf Temperaturänderungen [49], auf eine Änderung der Zusammensetzung des Lösungsmittels [50], auf einen geänderten pH-Wert oder auf elektrische Felder und indirekt auch auf Licht [51]. Unter Berücksichtigung der Hysterese am Volumen-Phasenübergang [49] ist diese Volumenänderung sogar reversibel. Für ein genaueres Verständnis der mikroskopischen Wechselwirkungen in den Gelen eignen sich Computersimulationen, mit denen sowohl Polymere als auch Lösungsmittelmoleküle verfolgt werden können und deren mesoskopisches Verhalten untersucht werden kann.

In der Natur kommen Gele sehr häufig vor. Das Zytoskelett einer Zelle, eine dreidimensionalen Proteinmatrix, bestimmt zusammen mit dem wässrigen Milieu nicht nur die Form und mechanische Stabilität, sondern dem Zytoskelett kommt zusätzlich die Aufgabe zu, die Fließeigenschaften im Innern der Zelle zu modifizieren und so den Stofftransport und den Metabolismus zu kontrollieren. Ein wesentlicher Vorteil biologischer Gele liegt in ihrer ausgesprochen hohen Transparenz, so beruht die Sehfähigkeit auf der gelartigen Konsistenz des Glaskörpers im Auge. Hier hätte ein Phasenübergang im Glaskörper schwerwiegende Folgen, es kämme zur Ablösung der Netzhaut. Die Augenhornhaut wird von einem weiteren Gel gebildet, dessen Trübung (Grauer Star) sich durch eine Phasenaufspaltung der Hornhaut und der damit verbundenen Opaleszenz erklären lässt. Zwischen Gelenken befinden sich ebenfalls Gele, deren besondere viskoelastische Eigenschaften, die auf der Wechselwirkung zwischen Polymer und Lösungsmittel beruhen, für eine gewisse Stabilität des „Schmiermittels“ sorgen, aber gleichzeitig durchlässig für die Nährstoffe zur Versorgung der lebenden Zellen sind.

In der Technik finden Gele zum Beispiel Anwendung bei der Elektrophorese, um Moleküle nach ihrer Größe zu trennen. Ein weiteres Einsatzgebiet sind Druckerfarben, die mit Gelen versetzt schneller antrocknen. In der Mikromechanik können Gele wie „Muskeln“ eingesetzt werden. Die drastische Volumenänderung am Phasenübergang kann auch dazu genutzt werden, Medikamente in Gele einzuschließen und so sicher an ihren Wirkort zu bringen. Dabei können die Medikamente in den Poren der gequollenen Gele eingelagert werden. Die Gele werden anschließend geschrumpft, um am Wirkort zum Beispiel durch Enzyme, die Säuren erzeugen, wieder zu quellen, wodurch die Medikamente dann freigesetzt

würden.

Bei den hier untersuchen Gelen handelt es sich um Gele aus NIPAM. In reinem Wasser beginnen sie bei Erwärmung zu schrumpfen. An der kritischen Temperatur, die zwischen 32 °C und 38 °C liegt, kommt es zum Volumen-Phasenübergang. Das Gel ändert seinen Zustand radikal, es geht vom Zustand der gequollenen Phase in den Zustand der geschrumpften Phase über, wobei es zu einer Phasenaufspaltung kommen kann, in der beide Phasen gleichzeitig im Gel vorkommen. Gele mit einem niedrigen Quervernetzeranteil verringern ihr Volumen am Phasenübergang sprunghaft und haben einen diskontinuierlichen Phasenübergang. Ob Gele mit höheren Quervernetzeranteilen am Phasenübergang eine kontinuierliche oder eine diskontinuierliche Volumenänderung vollführen, lässt sich durch Salze und Tenside beeinflussen. Der beobachtete Phasensprung bezieht sich hier auf eine Temperaturänderung, d. h. bei geringer Temperaturänderung am Phasenübergang kommt es zu einer großen Volumenänderung. Dieser Vorgang kann jedoch sehr langsam ablaufen; ein Gel benötigt für einen Phasenübergang unter Umständen Stunden. Dieses langsame zeitliche Verhalten wird auf Oberflächeneffekte zurückgeführt.

5.2 Die Synthese von Gelen

Eine MD-Simulation bildet das Verhalten realer Moleküle im Rahmen des verwendeten Atommodells und der klassischen Mechanik im Computer ab. Die meisten kleinen Moleküle besitzen in dieser Modellvorstellung naturgemäß eine relativ geringe Variabilität ihrer Struktur, nicht jedoch Makromoleküle wie Polymere oder Gele. Um die Struktur eines Polymers hinreichend realistisch zu modellieren ist es entweder notwendig die Struktur aus experimentellen NMR spektroskopischen Untersuchungen zu gewinnen oder den Bildungsprozess der Natur bei einer künstlichen Strukturbildung nachzuahmen. Im Folgenden wird der zweite Fall betrachtet, nämlich die Synthese von Polymeren und Gelen, hier insbesondere die von Poly(N-Isopropylacrylamid) Hydrogelen.

Zur Herstellung von Polymeren müssen die Monomere miteinander verbunden werden, hierzu gibt es mehrere Polymerisationsverfahren [52]. Sehr häufig werden Polymere durch radikalische Polymerisation gebildet. Die Polymerisation wird mit Hilfe eines Radikals gestartet. An das Radikal kann sich ein Monomer anlagern und mit ihm eine Verbindung eingehen, wobei die Verbindung wiederum ein Radikal bildet. Die Verbindung kann dann ein weiteres Monomer aufnehmen und durch weitere Polymerisation schließlich eine Polymerkette bilden. Die Polymerisation wird dabei so lange fortgesetzt, bis die radikale Polymerkette durch ein zweites Radikal neutralisiert wird oder keine weiteren Monomere mehr vorhanden sind.

Für die Gelbildung ist zwingend erforderlich, dass sich die Polymerketten untereinander zu einem Netzwerk verbinden. Die Verknüpfung der Polymerketten

kann hierbei entweder rein physikalisch, durch ein Verhaken der Polymerketten unter einander, erfolgen oder chemisch, mit Hilfe eines Quervernetzers. Der Quervernetzer verfügt über mehrere Gruppen, die in die Polymerketten eingebunden werden können. Diese erlauben es, den Quervernetzer in mehrere (meist zwei) Polymerketten einzubauen. Die so verknüpften Polymerketten können durch weitere Verknüpfungen mit anderen Polymerketten ein zusammenhängendes Netzwerk erzeugen, das dann als Gerüst für ein Gel dient.

Die drei in Abbildung 5.1a gezeigten Substanzen bilden zusammen mit dem Lösungsmittel Wasser das Poly(N-Isopropylacrylamid) Hydrogel. Als Monomere, die später das Gerüst des Polymernetzwerkes bilden sollen, werden die Kettenmonomere N-Isopropylacrylamid (NIPAM) und die Quervernetzer N,N'-Methylenbisacrylamid (BIS) verwendet. Das N-Isopropylacrylamid besteht aus Acrylamid, bei dem eines der Wasserstoffatome am Stickstoff durch eine Propyl-Gruppe ersetzt ist. Der Quervernetzer, das Bisacrylamid besteht aus zwei Acrylamiden, die am Stickstoff miteinander verknüpft sind. Die Acrylamide können jeweils eine Bindung mit einer Polymerkette eingehen, d. h. das Bisacrylamid ist ein tetrafunktionales Monomer, das in zwei Ketten eingebaut werden kann und diese mit einander verbindet.

Zur Herstellung des Gels werden beide Monomere in Wasser gelöst, zusätzlich wird Tetramethyl-ethylendiamin (TEMED) hinzu gegeben. Zur Initiierung der Synthese kommt eine frisch angesetzte Lösung Ammoniumpersulfat hinzu. Das Ammoniumpersulfat reagiert mit dem TEMED, indem es ein Wasserstoffatom abspaltet (Abbildung 5.1b). Dadurch wird das TEMED zu einem Radikal, an dem sich jetzt ein Monomer anlagern kann. Bei der Anlagerung des Monomers wird dessen Doppelbindung unter Bildung eines bifunktionalen Radikals geöffnet, dessen eine Hälfte sich dann mit dem Radikal des TEMED verbindet, wodurch sich ein dimeres Radikal bildet (Abbildung 5.1c). An das dimere Radikal kann sich dann ein weiteres Monomer anlagern und eine Bindung eingehen, wodurch aus dem dimeren Radikal ein trimeres Radikal wird. Durch eine sukzessive Anlagerung von Monomeren wird so eine Polymerkette gebildet.

Wie Abbildung 5.1d zeigt, wird die Kette zu einem Netzwerk erweitert, wenn sich anstatt eines NIPAM-Moleküls ein BIS-Molekül an die Kette anlagert. Das BIS-Molekül kann, da es wie erwähnt zwei Acrylamidreste besitzt, in zwei Ketten gleichzeitig eingebaut werden. Das BIS bildet damit einen Knotenpunkt zwischen zwei Polymerketten. Verknüpfen sich die Ketten mehrfach mit anderen Ketten, bildet sich ein Polymernetzwerk aus. Ob sich ein NIPAM-Molekül oder ein BIS-Molekül an eine Kette anlagert, ist immer eine Frage der Wahrscheinlichkeit. Das Verhältnis von NIPAM und BIS im Polymernetzwerk wird dem Verhältnis der Monomere im Lösungsmittel entsprechen. Die Polymerisation eines radikalischen Polymers wird durch Bindung mit einem zweiten Radikal abgebrochen.

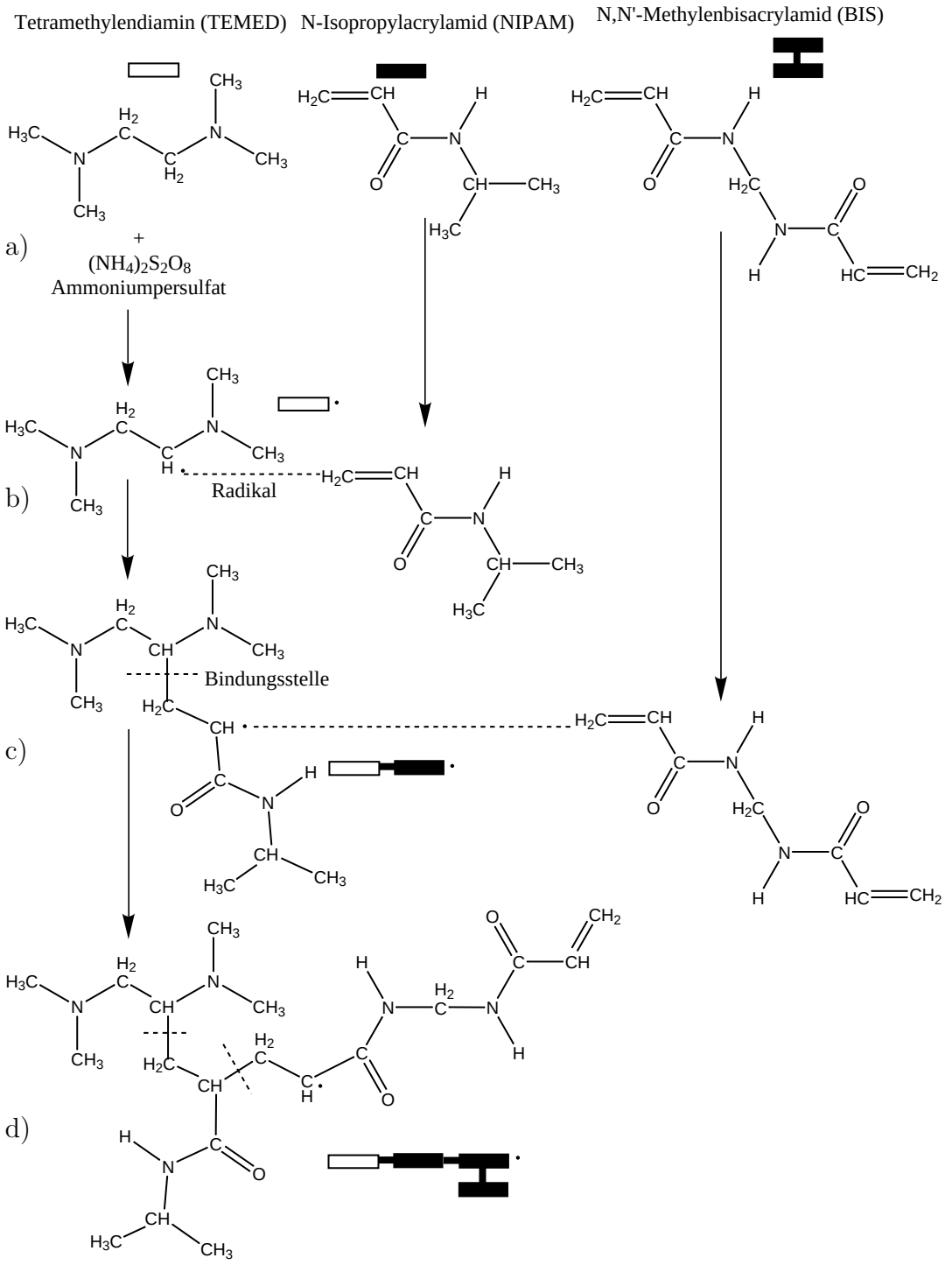


Abbildung 5.1: Synthese eines P-NIPAM-Hydrogels.

5.3 Strukturbildung

Zur Bildung einer Startstruktur mit dem Programm „StructC“ wird die in Abschnitt 2.3 beschriebene Kraft-Datenbank und die in Abschnitt 3.3 beschriebe Struktur-Datenbank bzw. eine Kombination beider verwendet. Grundlage für die Strukturbildung und für die spätere Simulation bilden die Kraftfelder, die auch die Relationen zwischen den wechselwirkenden Atomen festlegen. Für die Kraftfelder gibt es Standard-Kraftfeldparameter, die zwar für die Simulation von Proteinen optimiert wurden, aber auch technische Polymere recht gut charakterisieren. Eine Beschreibung der Kraftfeldmodelle und eine Auflistung beispielhafter Parametersätze für Kraftfelder ist am Anfang von Kapitel 2 zu finden.

Zur Simulation des P-NIPAM Gels werden keine naturgetreuen Abbilder der realen P-NIPAM Netzwerke erzeugt, diese wären für eine Simulation viel zu groß. Implementiert wird ein angenähertes Modell, das die wesentlichen Eigenschaften eines P-NIPAM Gels besitzt. Das P-NIPAM Gel besteht im Wesentlichen aus Wasser, dem Kettenmonomer NIPAM, dem Quervernetzer BIS, so wie dem Startmonomer TEMED. Das TEMED kommt in der Lösung nur in sehr geringer Konzentration vor und ist in einem Teilvolumen, der Größenordnung eines Simulationsraumes praktisch nicht vorhanden. In den Simulationen wird das TEMED deshalb vernachlässigt. Eine geschlossene Struktur ohne Startmonomer erreicht man, in dem man die Polymerketten nur zwischen Quervernetzern aufbaut. Das zufällige Wachstum einer realen Polymerkette wird durch die Erzeugung der Ketten auf einem Pfad imitiert, der mit der Methode des „directed random walk“ erzeugt wird.

Die Bindungsverhältnisse innerhalb eines Moleküls werden innerhalb der Struktur-Datenbank, in einer der Z-Matrix ähnelnden Struktur beschrieben, hierbei müssen zusätzlich die Bindungsverhältnisse zu den Nachbarmonomeren in der Monomerstruktur aufgeführt werden. Die genaue Kodierung der Strukturdaten wurde ebenfalls in Abschnitt 3.3 beschrieben, dort ist auch in Abbildung 3.5 die Struktur-Datenbank des Bisacrylamid Quervernetzers und der NIPAM Kettensegmente dargestellt.

Die Erstellung der zu simulierenden Moleküle im periodischen Simulationsraum erfolgt in der, im Abschnitt „Create“ der Struktur-Datenbank (z. B. in Abbildung 3.6) vorgegebenen Reihenfolge. Ebenfalls dort wird der Ablauf der „Polymerisation“ der verwendeten Polymere festgelegt. Festgelegt wird aber nur der grobe Ablauf, d. h. welches Bisacrylamid mit welchem anderen über eine NIPAM-Kette verbunden ist, der genaue Verlauf der Ketten wird durch den in Abschnitt 3.4 beschriebenen „random walk“ definiert. Zu beachten ist, dass der zufällig erzeugte Verlauf der Polymerkette im freien Raum die Realität besser widerspiegelt, als die vom alten Programm erzeugte Kette auf dem Diamantgitter.

Die Konstruktion eines NIPAM-Gels mit zwei Quervernetzermonomeren ist in Abbildung 3.6 beschrieben. Für Polymere mit höheren Quervernetzeranteil und damit zwangsläufig einem dichteren Polymergerüst werden weitere Quervernetzer

```
[Molecule]
Name=Water
OW -0.8476
HW  0.4238 1
HW  0.4238 1 2

[Nature]
OW     15.9994 25.01  793.0    0.0    0   25.01   421.0    0.0    0.0
HW      1.0080  0       0      0      0    0        0      0      0

[Bond]
OW    HW    0 1.000

[Angle]
HW   OW   HW    0 109.5

;[Dihedral]
;[Improper]
```

Abbildung 5.2: Kraft- / Struktur-Datenbank von SPC/E Wasser.

durch zusätzliche Ketten in das Gel eingebunden. Nach Erzeugung des Polymergerüstes, wird der Simulationsraum solange mit Wasser aufgefüllt, bis eine Dichte von $1\,{}^{g}/_{cm^3}$ erreicht ist. Das verwendete Wasser entspricht dem extended simple point charged (SPC/E) Wassermodell von Berendsen u. a. [53]. Die Struktur-Datenbank des Wassers ist in Abbildung 5.2 abgebildet. Nach dem Auffüllen des Simulationsraumes mit Wasser ist die Erzeugung der Startstruktur beendet, die Strukturdaten werden abschließend im Skript aus Abbildung 3.6 in eine PDB- und eine Strukturdatei (DYN) gespeichert.

Die oben beschriebene Generierung einer Struktur mit dem Programm „StructC“ ist sehr flexibel, dem gegenüber ist das alte Strukturgenerierungsprogramm aus Abschnitt 3.2 nur auf die Erzeugung von P-NIPAM-Gelen spezialisiert. Unterschiedliche Polymere lassen sich wie bereits erwähnt durch eine Variation in der Größe des Simulationsraumes, dem Quervernetzeranteil, oder durch die, den Aufbau der Polymerketten bestimmenden, minimalen und maximalen Abstände der Quervernetzer, so wie einem Faktor p für den „random walk“ erzeugen. Für die Simulation und Auswertung wurden nach dem hier beschriebenen Verfahren die in Tabelle 5.1 aufgeführten vier Polymersysteme A bis D, mit unterschiedlichem Polymeranteil erzeugt, ergänzt durch eine Vergleichssimulation bestehend aus reinem Wasser.

5.4 Simulation

Mit Hilfe der im vorherigen Abschnitt erzeugten Startstruktur lässt sich ein, wie in Abschnitt 4.4 beschriebenes Simulationssystem eines P-NIPAM Hydrogels simulieren. Für die Simulation wird eine Steuerungsdatei benötigt, die sämtliche für die Simulation notwendigen Parameter enthält. Handelt es sich bei der Startstruk-

Tabelle 5.1: Simulationssysteme

System	BIS	NIPAM	Wasser	Volumen/ nm^3	Dichte g/cm^3	Polymer-anteil	Querver-netzeranteil
A	2	15	317	12,204	1,05	25 %	13 %
B	3	23	274	12,654	1,05	38 %	13 %
C	4	33	227	13,348	1,05	50 %	12 %
D	2	19	525	19,683	1,00	20 %	10 %
Wasser	0	0	390	11,667	1,00	0 %	0 %

tur um eine der im vorherigen Abschnitt mit dem Programm „StructC“ erzeugten Strukturen, so ist diese nicht im thermischen Gleichgewicht. Sie muss durch Simulation einer Equilibrierungsphase in dieses Gleichgewicht gebracht werden. Die Steuerungsdatei einer solchen Equilibrierungsphase ist in Abbildung 4.1 dargestellt.

Für die Bestimmung der Wechselwirkungen zwischen den Atomen des Polymers werden, die bereits bei der Konstruktion verwendeten GROMOS [7] Kraftfeld-Parameter verwendet. Ebenso werden die SPC/E [53] Kraftfeld-Parameter wieder benutzt. Für die van der Waals- und die Coulomb-Wechselwirkungen wird ein einheitlicher cutoff-Radius von 10 Å eingestellt. Zur Bestimmung der elektrostatischen Wechselwirkungen wird, das gegenüber dem einfachen Coulomb-Verfahren wesentlich genauere Ewald-Kornfeld-Verfahren [32], verwendet. Hierbei wird ein κ von 6.0 verwendet, so wie im reziproken Raum eine Schale der Wechselwirkung vom Radius $5 * {}^{2\pi}/_{L}$ festgelegt. Weiter wird angenommen, dass sich das elektrostatische Feld im Vakuum befindet, somit ist die Permittivitätszahl $\epsilon_r = 1$. Für das elektrostatische Feld ergibt sich somit ein Vorfaktor von:

$$\text{Diel} = \frac{1}{4\pi\epsilon_0\epsilon_r} = 332.0636 \frac{\text{kcal Å}}{\text{mol } e^2} \tag{5.1}$$

Die in der Simulationsstruktur vorkommenden einzelnen Wasserstoffatome werden, wie in 4.4 beschrieben, von Zwangskräften auf feste Bindungsabstände gehalten, die bereits in den Kraftfeldparameter festgelegt wurden. Zur Erhaltung dieser Zwangskräfte kommt hier der RATTLE-Algorithmus [42] zusammen mit dem velocity Verlet [40] Integrationsalgorithmus zur Anwendung.

Ein Simulationssystem wie das in Abbildung 4.1 besitzt nach der Strukturgenerierung keine kinetische Energie, dafür aber eine sehr hohe potentielle Energie. Ein Großteil der potentiellen Energie wird in der Equilibrierungsphase in kinetische Energie umgewandelt, die wiederum größten Teils durch einen Thermostaten an ein „Wärmebad“ abgeführt wird. Da vor allem zu Beginn der Equilibrierung

sehr viel Energie umgewandelt wird, die Simulationstemperatur jedoch nicht zu groß werden darf, muss das System mit sehr kleiner Schrittweite simuliert werden.

Vor der ersten Equilibrierung werden bei jedem neuen Simulationssystem zehn Integrationsschritte mit einer Schrittweite von $3\ 10^{-3}$ fs ausgeführt, damit haben diese vor der eigentlichen Equilibrierung bereits einen kleinen Teil ihrer potentiellen Energie in kinetische Energie umgewandelt, ihre Temperatur ist dann nicht mehr Null Kelvin. Dies ist für die Festlegung der Schrittweiten am Anfang der Equilibrierung wichtig, da im Algorithmus zur Schrittweitenkontrolle die aktuelle Temperatur als Faktor eingeht. Anschließend beginnt die Equilibrierungsphase, in der das System nach jedem Simulationsschritt auf 300 K thermalisiert wird. Die Schrittweite der Simulation wird dabei vom Simulationsprogramm „dynC“ dynamisch von $3\ 10^{-3}$ fs bis auf 1 fs bzw. 2 fs, der späteren Schrittweite in der Produktionsphase erhöht.

Die Schrittweitenregelung vergrößert die Schrittweite um 10%, falls die Temperatur des Systems nach einem Simulationsschritt ohne Thermalisierung unterhalb einer Temperatur von 500 K bleibt. Bei einer Temperatur von mehr als 600 K wird die Schrittweite automatisch auf 80% der ursprünglichen Schrittweite verringert, so dass die Temperatur immer unterhalb des im Skript angegebenen oberen Grenzwertes von 600 K bleibt. Nach Festlegung der Schrittweite wird das System mit Hilfe des Thermostaten von Woodcock [4] auf 300 K thermalisiert. Der erste Teil der Equilibrierungsphase ist beendet, wenn der Integrationsalgorithmus seine endgültige Schrittweite erreicht hat. An diesen schließt sich eine zweite Equilibrierungsphase von 200 ps an, in der die Temperatur nun durch einen Berendsen Thermostaten mit einer Kopplungskonstanten von 0.5 ps geregelt wird. Diese zweite Phase soll das System endgültig in das thermische Gleichgewicht bei einer Temperatur von 300 K führen.

Die Umwandlung von potentieller Energie in kinetischer Energie und deren Abgabe an ein externes Wärmebad während der Equilibrierung lässt sich in Tabelle 5.2 für das System *D* gut beobachten. Zu Anfang wird mit kleinen Schrittweiten viel potentielle Energie in kinetische Energie umgewandelt, die Temperatur des Systems liegt immer über 400 K, da das System nach jedem Simulationsschritt auf 300 K „abgekühlt“ wird, wird viel Energie an das externe Wärmebad abgegeben. Die Energieabgabe an das externe Wärmebad führt zu einer entsprechenden Verringerung der Gesamtenergie des kanonischen Systems von $-497.23\ 10^{-12}$ cal am Beginn der Equilibrierung auf $-881.21\ 10^{-12}$ cal am Ende der ersten Equilibrierungsphase, die mit dem Erreichen der endgültigen Schrittweite abgeschlossen ist (leere Zeile in Tabelle 5.2).

In der zweiten Equilibrierungsphase vermindert sich die Energieabgabe stetig, da jetzt die Schrittweite konstant bleibt und nicht durch weitere Erhöhung immer mehr kinetische Energie bis zur Abgabe an das Wärmebad akkumuliert wird. Durch die abnehmende Energieumwandlung verringert sich die Temperatur des Systems und somit der Transfer von kinetischer Energie in das externe Wärmebad. Nach dem erreichen der Ziel-Temperatur von 300 K findet nur noch

ein geringer Energieaustausch zwischen System und Wärmebad statt. Insgesamt wird in der zweiten Equilibrierungsphase nur noch etwa 0.2% der anfänglichen Energie abgegeben.

Hat das System das thermische Gleichgewicht erreicht, beginnt die Produktionsphase, in der die Phasenraumpunkte der laufenden Simulation in einer Datei (zur späteren Auswertung) abgespeichert werden. Nach Tabelle 5.2 beginnt die Produktionsphase im System D bei 200 ps. Im Idealfall sollte in der Produktionsphase die Gesamtenergie konstant bleiben, doch wegen Ungenauigkeiten bei der Kraftberechnung und im Integrationsalgorithmus bzw. wegen des endlichen Simulationsraumes schwankt die Gesamtenergie um einen Mittelwert, der hier bei $-895.2\ 10^{-12}$ cal liegt.

Vor der Produktionsphase wurden aus dem thermalisierten System zwei Simulationen abgespalten, die anschließend neu equilibriert wurden und sich bei einer Temperatur von 285 K bzw. 325 K im thermischen Gleichgewicht befinden. Hierdurch lässt sich das selbe System einfach bei unterschiedlichen Temperaturen untersuchen.

Die Phasenraumpunkte der Simulation werden in der Produktionsphase in einer Schrittweite von 1 fs bzw. 2 fs aufgesucht. Mit Hilfe des Berendsen Thermostaten [43] wird die Temperatur nach jedem Simulationsschritt, über eine Kopplungskonstante von 0.5 ps, auf dem vorgegebenen Wert gehalten, so dass die Simulation im kanonischen Ensemble (NVT) abläuft. Alle 10 bis 50 fs werden die Phasenraumpunkte in die Trajektoriendatei gespeichert. Die Aufzeichnungsdauer variiert hier, in Abhängigkeit von der Abspeicherfrequenz, zwischen 300 ps und 1 ns. Die Auswertung kurzzeitiger, eher lokaler Effekte, wie der Reorientierungsdiffusion erfolgt mit hoher Abspeicherfrequenz. Im Gegensatz dazu wird die Translationsdiffusion, die durch ein Fortschreiten der Atome bestimmt wird, bei niedriger Abspeicherfrequenz, dafür aber über eine lange Zeitspanne ausgewertet.

Tabelle 5.2: Energieentwicklung während der Equilibrierung / Simulation von System D.

$t/$ ps	$\delta t/$ fs	$T/$ K	$E_{kin}/$ 10^{-12} cal	v.d. Waals/ 10^{-12} cal	Coulomb/ 10^{-12} cal	Bindung/ 10^{-12} cal	$E_{pot}/$ 10^{-12} cal	$E_{ges}/$ 10^{-12} cal
$6\ 10^{-6}$	0.003	351.2	2.14	389.81	-891.96	2.783	-499.37	-497.23
$15\ 10^{-6}$	0.003	369.8	2.25	388.53	-891.96	2.783	-500.64	-498.39
$368\ 10^{-6}$	0.011	529.4	3.23	340.85	-891.96	2.777	-548.33	-545.10
$1037\ 10^{-6}$	0.017	527.6	3.22	273.83	-891.97	2.766	-615.38	-612.15
$2\ 10^{-3}$	0.024	504.0	3.07	207.36	-891.98	2.752	-681.87	-678.79
$3.9\ 10^{-3}$	0.052	529.8	3.23	139.67	-892.02	2.743	-749.61	-746.37
0.008	0.136	518.8	3.17	73.95	-892.18	2.796	-815.44	-812.27
0.02	0.686	493.8	3.02	22.59	-893.43	2.908	-867.94	-864.92
0.029	1.337	491.3	3.00	12.97	-894.66	2.524	-879.17	-876.17
0.036	2.000	478.2	2.92	9.17	-895.45	2.159	-884.13	-881.21
0.04	2	461.3	2.82	0.01	-896.75	0.742	-896.00	-893.18
0.35	2	439.4	2.68	-0.26	-896.62	0.480	-896.39	-893.71
0.83	2	376.5	2.30	-0.47	-896.53	0.360	-896.64	-894.34
1.33	2	347.8	2.12	-0.62	-896.56	0.341	-896.83	-894.71
3.33	2	310.7	1.90	-0.83	-896.57	0.354	-897.05	-895.15
5.33	2	298.8	1.83	-0.80	-896.57	0.362	-897.00	-895.18
20.13	2	300.3	1.83	-0.85	-896.60	0.434	-897.01	-895.18
100.13	2	302.2	1.85	-0.88	-896.59	0.400	-897.07	-895.23
200.13	2	300.5	1.84	-0.87	-896.57	0.372	-897.06	-895.22
300.13	2	300.1	1.83	-0.82	-896.60	0.355	-897.07	-895.23
400.13	2	306.3	1.87	-0.85	-896.60	0.362	-897.09	-895.21
500.13	2	295.9	1.81	-0.86	-896.55	0.359	-897.05	-895.24

Kapitel 6

Simulation von Flüssigkristallen

Flüssigkristalle liegen in einer Phase vor, deren Ordnung zwischen der einer Flüssigkeit und der eines Kristalls liegt. Sie vereinigen typische Eigenschaften von Flüssigkeiten, wie dem Fließverhalten, mit den Eigenschaften von Festkörpern, wie zum Beispiel der Anisotropie. Gebildet werden die Flüssigkristalle von Molekülen mit ausgeprägter Stab- oder Scheibenform. In Abhängigkeit von der Temperatur sind die Moleküle dabei in verschiedenen Phasen mehr oder weniger geordnet.

Die einfachste Flüssigkristallphase bildet die nematische Phase. Sie zeichnet sich durch eine Orientierungsfernordnung der Moleküle aus, das heißt die Moleküle sind wie in Abbildung 6.1a gezeigt im Mittel in gleicher Richtung ausgerichtet. Optisch verhält sich eine nematische Phase wie ein einachsiges Kristall. Im Gegensatz zu einem Kristall lässt sich die Orientierung der Moleküle und somit die Orientierung des Flüssigkristalls jedoch durch elektrische Felder beeinflussen. Durch die Orientierung der Moleküle wird Licht, das den Flüssigkristall durchdringt, in einer Ebene polarisiert. Der Flüssigkristall wirkt somit als Polarisationsfilter, dessen Polarisationsebene aber mit Hilfe einer angelegten elektrischen Spannung gesteuert werden kann. Anwendung findet diese elektrisch steuerbare Drehung der Polarisationsebene zum Beispiel in jeder Flüssigkristallanzeige.

Neben der nematischen Phase können Flüssigkristalle auch in sogenannten smektischen Phasen vorliegen; diese treten in der Regel bei niedrigeren Temperaturen als die nematischen Phasen auf. Die smektischen Phasen bilden Schichtstrukturen, innerhalb derer die Moleküle frei beweglich sind. Abbildung 6.1b zeigt Moleküle in der smektischen A-Phase, sie sind senkrecht zur Schichtebene orientiert. Bilden die Moleküle einen von 90^o verschiedenen Winkel, so befinden sich die Moleküle in der C-Phase. Neben diesen smektischen Phasen gibt es weitere, die sich durch eine Fernordnung der Moleküle auszeichnen.

Simuliert werden Flüssigkristalle in der Regel durch ein sehr einfaches Stäbchen-Molekülmodell. Hierbei wird jedes Molekül als starres Ellipsoid betrachtet, das über ein Lennard-Jones-Potential mit anderen Molekülen wechselwirkt. Dieses Modell wurde zuerst von Gay und Berne [54] aufgestellt und ist Grundla-

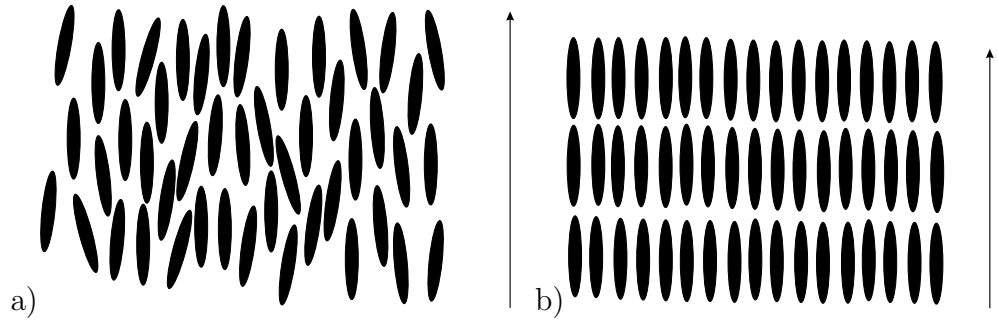
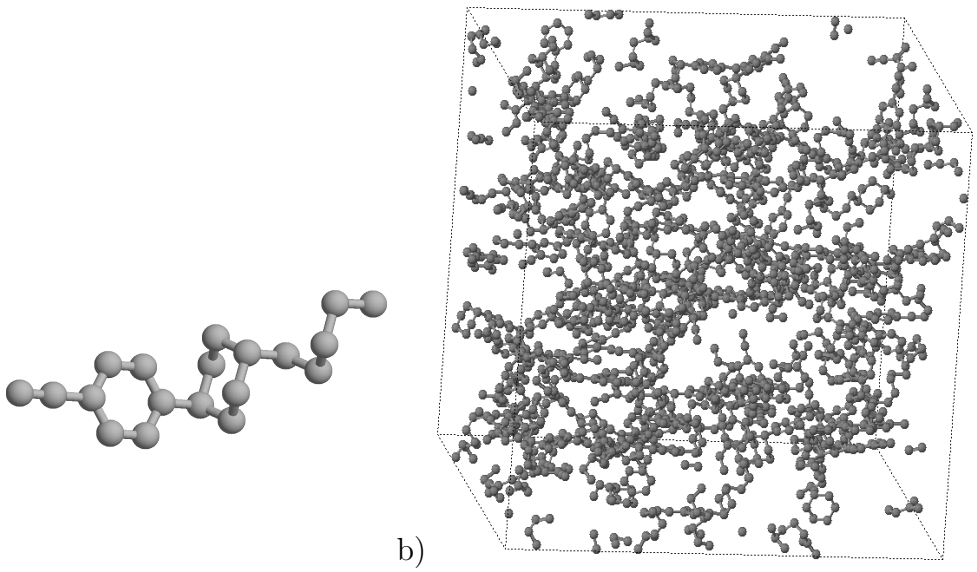


Abbildung 6.1: a) Nematische Phase b) Smektische A-Phase. Orientierung der Moleküle wird durch ein äußeres Kraftfeld hervorgerufen, das durch die Vektoren angedeutet werden soll.

ge der meisten Simulationen von Flüssigkristallen, so zum Beispiel der von Allen u.a. [55] oder der von Wagner und Bennett [56].

Wie oben bereits erwähnt wird in Flüssigkristallanzeigen die Polarisationsebene bzw. die Orientierung der Moleküle durch äußere elektrische Felder variiert. Zur Simulation von Flüssigkristallanzeigen gehören deshalb auch immer äußere elektrische Felder, da aber die Gay-Berne-Moleküle als neutrale Teilchen konzipiert sind, sind sie für die Simulation von Flüssigkristallanzeigen ungeeignet.

Einen besseren Ansatz bieten hier die realistischeren Atom-Atom Wechselwirkungen einer MD-Simulation, die zum Beispiel die Gruppe von Geiger [57,58] und Sandström u.a. [59] verwendet haben. Mit dem Programm „StructC“ lassen sich diese in zufälligen Startstrukturen ohne Vororientierung der Kristalle generieren. Abbildung 6.2b zeigt hierzu die Startstruktur eines Flüssigkristalls aus 100 Molekülen, erzeugt aus 4-(Trans-4-Pentylcyclohexyl)Benzonitril Molekülen. Diese in Abbildung 6.2a gezeigte Struktur wurde aus der in Abbildung 6.3 gezeigten Struktur-Datenbank mit dem Programm „StructC“ erzeugt. Die in der Datenbank verwendeten Kraftfelder entsprechen den von Krömer u. a. [57] aus den GROMOS Kraftfeldern [7] abgeleiteten Werten. Das mit dieser Startstruktur erzeugte System kann mit dem Simulationsprogramm „dynC“ in ein thermisches Gleichgewicht gebracht und anschließend über ein längeres Zeitintervall simuliert werden. Allerdings ist die Simulation von Flüssigkristallanzeigen mit „dynC“ zur Zeit nicht möglich, da ein hierfür notwendiges äußeres (elektrisches) Feld nicht implementiert ist. Eine Implementierung eines solchen Feldes dürfte aber problemlos möglich sein.



a) b)

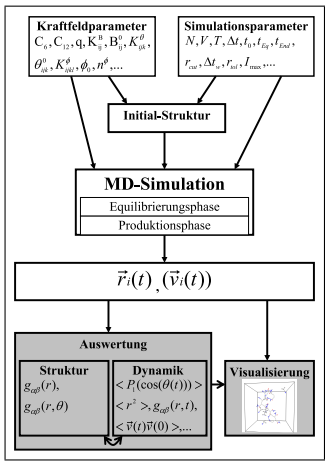
Abbildung 6.2: a) 4-(Trans-4-Pentylcyclohexyl)Benzonitril; Aufbau von links nach rechts entsprechend der Struktur-Datenbank in Abbildung 6.3.
b) Startstruktur mit 100 Molekülen.

```
[Structure]
seed=779
BoxSize=45
include gromos.txt
out pdb pch5.pdb
[Create]
PCH5=100

[Molecule]
Name=PCH5

;[No] Nam    q  [R]    Atm [Ang] Atm Per/Ang Atm (vgl. Z-Matrix)
   1  N   -0.5
   2  C    0.5          1
   3  CB   0.0          2         1
   4  CR6  0.0          3         2      0    1
   5  CR6  0.0          4         3      0    2
   6  CR6  0.0          3         2      1    1
   7  CR6  0.0          6         3      0    2
   8  CB   0.0          7         6      1    3     5
   9  CS1  0.0          8         7      0    6
  10  CS2  0.0          9         8      1    7
  11  CS2  0.0         10         9      2    8
  12  CS2  0.0          9         8      0    7
  13  CS2  0.0         12         9      0    8
  14  CS1  0.0         13        12      0    9    11
  15  CH2  0.0         14        13      0   12
  16  CH2  0.0         15        14      0   13
  17  CH2  0.0         16        15      0   14
  18  CH2  0.0         17        16      0   15
  19  CH3  0.0         18        17      0   16
```

Abbildung 6.3: Struktur-Datenbank von 4-(Trans-4-Pentylcyclohexyl)Benzonitril.

Kapitel 7

Auswertung und Visualisierung



Die Auswertung der gespeicherten Phasenraumpunkte bildet den eigentlichen Zweck einer Simulation. Mit Hilfe der unterschiedlichen Methoden zur Auswertung werden die chemisch-physikalischen Eigenschaften der simulierten Systeme bestimmt, wobei sich die Methoden grob in zwei Klassen einteilen lassen. In der einen Klasse wird die Struktur des Systems analysiert, d. h. es werden Abstände zwischen den Atomen oder deren relative Lage zueinander statistisch ausgewertet. Die andere Klasse beschäftigt sich mit der Dynamik der einzelnen Atome bzw. Moleküle, hierbei wird sowohl die absolute Dynamik einzelner Atome berücksichtigt, als auch die relative Dynamik zu anderen Atomen oder Molekülen.

Es gibt zwei Arten relevanter Mittelwerte. Der Erste ist der *Ensemble*-Mittelwert (7.1), er ist der statistische Mittelwert einer Observablen $f^{(k)}(t)$ zu *einer* vorgegebenen Zeit t über *alle* N Systeme eines Ensembles. Die zweite Art Mittelwert ist der *Zeit*-Mittelwert (7.2), der den Mittelwert der Observablen $f(t) = f^{(k)}(t)$ in *einem* System k des Ensembles über ein langes *Zeitintervall* τ bestimmt. Die Ergodenhypothese sagt für stationäre Zustände voraus, dass die Observable f in jedem System jeden ihr zugänglichen Wert in einer hinreichend langen Zeit annimmt. In einem stationären Ensemble ist der Ensemblemittelwert zeitunabhängig, außerdem muss wegen der Ergodenhypothese auch der Zeitmittelwert für fast alle Systeme des Ensembles gleich ein. Zur Bestimmung des Mittelwertes einer Observablen f reicht es somit aus, den Zeitmittelwert (7.2) zu bestimmen. Da in einem stationären ergodischen Ensemble Zeitmittelwert (7.2) und Ensemblemittelwert (7.1) äquivalent sind, gilt:

$$\langle f(t) \rangle = \frac{1}{N} \sum_{k=1}^{N} f^{(k)}(t) \tag{7.1}$$

$$= \lim_{\tau \to \infty} \frac{1}{\tau} \int_0^{\tau} f(t) \mathrm{d}t \tag{7.2}$$

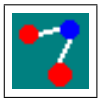
Die Analyseverfahren zur Auswertung einer Simulation lassen sich einteilen in Methoden zur Analyse der statischen Struktureigenschaften und in Methoden zur Bestimmung der Dynamik. Die statischen Eigenschaften werden im wesentlichen durch die Anwendung von Korrelationsfunktionen (Abschnitt 7.2.1) auf die Trajektorien bestimmt. Insbesondere die Paarkorrelation (Abschnitt 7.2.2), die die Verteilung einer Sorte von Atomen um eine Andere beschreibt, wird in den unterschiedlichsten Varianten verwendet.

Für die Dynamik der Atome und Moleküle spielen Diffusionen und deren zugehörige Diffusionskoeffizienten eine wichtige Rolle. Unterschieden wird zwischen Translationsdiffusion (Abschnitt 7.2.3), berechnet mit Hilfe der mittleren quadratischen Verschiebung, und Rotationsdiffusion (Abschnitt 7.2.4), die die Drehbarkeit der Moleküle beschreibt. Ein anderes Verfahren bestimmt die Geschwindigkeitsautokorrelation eines Moleküls, aus der dann durch Fouriertransformation die Spektraldichte (Abschnitt 7.2.5) bestimmt werden kann. In der Spektraldichte werden die Schwingungsfrequenzen der Moleküle dargestellt. Einen anderen Ansatz verfolgt die Analyse der Wasserstoffbrückenbindungen (Abschnitt 7.2.6), die sich auf die Bestimmung der Dauer einer Wasserstoffbrückenbindungen konzentriert.

Implementiert wurden die oben aufgeführten Methoden im „*Si*mulatuions *D*aten *An*alyse“-Programm „SiDAn“. Eine genaue Beschreibung der Implementierungen und Modifikationen ist in Abschnitt 7.3 zu finden. Eines der Korrelationsfunktionen erzeugt eine drei dimensionale Verteilung einer Sorte von Atomen um eine Gruppe aus drei Atomen. Zur Visualisierung dieser und anderer Verteilungen wurde das Programm „TTPlot“ (Abschnitt 7.4) von mir entwickelt.

Dennoch lässt sich häufig aus den Eigenschaften der Systeme nur schwer eine Vorstellung von den Verhältnissen im Simulationsraum gewinnen. Das im nächsten Abschnitt beschriebene Programm „TrajPlay“ erzeugt aus einer Trajektorie eine Animation der Simulation, wobei einzelne Atome und deren intramolekulare Bindungen im zeitlichen Verlauf visualisiert werden.

7.1 Visualisierung mit TrajPlay



Eine einfache, aber zugleich sehr anschauliche Methode der Auswertung, ist die Visualisierung einer Simulation. In ihr kann der Verlauf einer Simulation als Animation betrachtet werden. Mit dem Programm „TrajPlay“ (Abbildung 7.1) ist es möglich aus einer Konfigurations-Datei und der zugehörigen Trajektorien-Datei eine solche Animation zu erzeugen.

Zur Visualisierung der Trajektorie wird entweder ein Visualisierungsrahmen erzeugt oder eine TrajPlay-Datei (*.tpl) mit Visualisierungsrahmen eingelesen. Ein Visualisierungsrahmen speichert die zur Visualisierung notwendigen Daten, dies sind zum Beispiel Daten über den Blickwinkel, die Vergrößerung, die verwendeten Farben oder die benutzte Konfigurations- und Trajektorien-Datei.

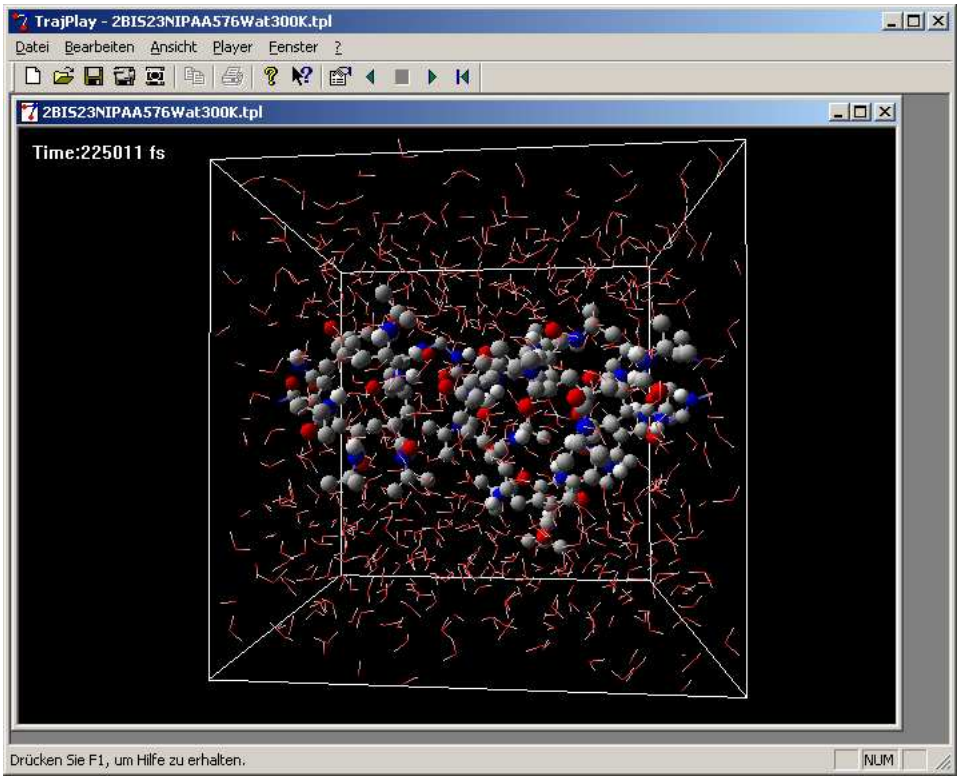


Abbildung 7.1: Das Programm „TrajPlay“ mit der Darstellung eines Systems bestehend aus einem Gel mit einem Gerüst aus zwei BIS- und 23 NIPAM-Monomeren, so wie 576 Wassermolekülen bei 300 K.

Ein neuer Visualisierungsrahmen wird über den Menüpunkt „Datei“ → „Neu...“, die Tastenkombination Strg - N oder das Symbol 🗋 erzeugt, hierfür müssen in drei Dialogen nacheinander die Konfigurations-Datei (*.cfg), die Trajektorien-Datei (*.tap) und die Informations-Datei (*.info.txt) ausgewählt werden. Auf eine bestehende, zuvor mit „TrajPlay“ erzeugte TrajPlay-Datei kann über den Menüpunkt „Datei“ → „Öffnen...“, die Tastenkombination Strg - O oder das Symbol 🗁 zugegriffen werden. Zum Speichern eines Visualisierungsrahmens existieren vier Wege. Über den Menüpunkt „Datei“ → „Speichern“, so wie über die Tastenkombination Strg - S oder das Symbol 🖫 wird eine bereits gespeicherte Datei mit den aktuellen Einstellungen überschrieben bzw. die Einstellungen in eine in einem Dialog festzulegende Datei gespeichert. Die Einstellungen können auch über den Menüpunkt „Datei“ → „Speichern unter...“ in eine beliebige Datei gespeichert werden. Die gespeicherten Einstellungen bestehen aus den drei bei der Erzeugung der Visualisierung festgelegten Dateien. Diese werden ergänzt durch die Einstellungen zur Visualisierung, die einerseits aus dem Betrachtungswinkel, Abstand und der Verschiebung bestehen, und andererseits aus Eigenschaften wie Farben und Größen zur Darstellung.

Die Eigenschaften und Einstellungen der Visualisierung lassen sich einteilen in die Parameter für die Beobachtungsperspektive und die Parameter für Atome und Bindungen. Die Einstellungen für die Perspektive werden über die Maus gesteuert. Der Betrachtungswinkel des Beobachters wird bei gedrückter linker Maustaste durch Bewegungen der Maus geändert. Wird zusätzlich zur linken Maustaste die Shift-Taste gedrückt, so lässt sich der optische Abstand zum Simulationsraum variieren. Bei gleichzeitig gedrückter linker Maustaste und Strg-Taste kann der Simulationsraum im Anzeigebereich verschoben werden, wird zusätzlich die Shift-Taste gedrückt, ist es unter Berücksichtigung der periodischen Randbedingungen möglich, die Teilchen im Simulationsraum zu verschieben.

Die Parameter der Atome und Bindungen werden über einen mit „Ansicht“ → „Eigenschaften...“ bzw. über das Symbol aufzurufenden Dialog verändert. Innerhalb des Dialoges werden die betreffenden Parameter in drei Bereiche aufgeteilt, die in entsprechenden, durch Reiter auszuwählende, Teildialogen verändert werden können. Den ersten Teildialog zeigt Abbildung 7.2. Im oberen Teil werden die Größen der Kugeln festgelegt, die in der Visualisierung die einzelnen Atome repräsentieren, außerdem wird die Dicke der, die Bindungen repräsentierenden, Zylinder festgelegt. Im unteren Teil des Dialoges wird sowohl der augenblicklich betrachtete Zeitpunkt bzw. die aktuelle Position in der Trajektorien-Datei eingestellt, als auch die Geschwindigkeit mit der die spätere Animation abläuft. Der zweite Teildialog, in Abbildung 7.3 der linke Dialog, bestimmt die in der Visualisierung verwendeten Farben. Die Farben können durch „Klicken“ auf die entsprechend eingefärbte Fläche geändert werden, zurück zu der in der Abbildung zu sehende Standardeinstellung gelangt man durch betätigen des Schalters „Standard“. Die neuen Farbeinstellungen werden jedoch erst nach betätigen des OK-Schalters in die Darstellung übernommen. Der letzte Teildialog der Eigenschaften verwaltet die folgenden drei Parameter. Mit dem ersten Parameter kann eine Stereo-Darstellung eingeschaltet werden, in der durch ein Bild für jedes Auge ein 3D-Effekt erzeugt wird. Weiter können ein Beleuchtungseffekt zugeschaltet und die Güte der dargestellten Kugeln eingestellt werden.

Die mit diesem Programm erzeugbaren Animationen können durch vier Befehle gesteuert werden. Diese sind sowohl im Untermenü des Menüeintrags „Player“ als auch als Schalter in der Symbolleiste vorhanden. Gestartet wird die Animation über das Symbol oder den Untermenüeintrag „Play“. Eine laufende Animation kann mit bzw. dem Menüeintrag „Stop“ angehalten werden. Neben diesen beiden wichtigsten Kommandos sind zwei weitere vorhanden. Mit bzw. dem Eintrag „Anfang“ springt die Animation an den Anfang der Trajektorien-Datei. In die entgegengesetzte Richtung läuft die Animation mit bzw. dem Eintrag „Rückwärts.“

Aus den mit „TrajPlay“ erzeugten Visualisierungen können einzelne Bilder und Sequenzen in Dateien gespeichert werden. Unter dem Menüeintrag „Datei“ → „Bild speichern...“ bzw. über das Symbol kann die aktuell dargestellte Kon-

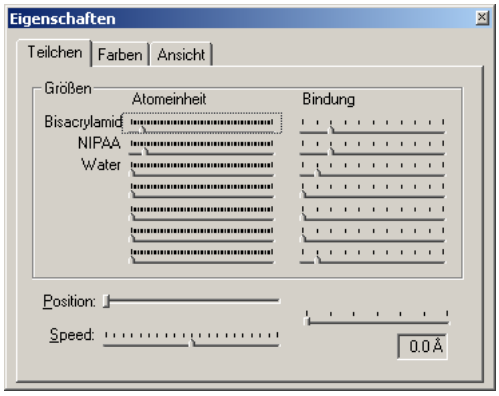


Abbildung 7.2: Teildialog zur Bearbeitung der Größen der Teilchen und Bindungen, so wie zur Position innerhalb der Trajektorie.

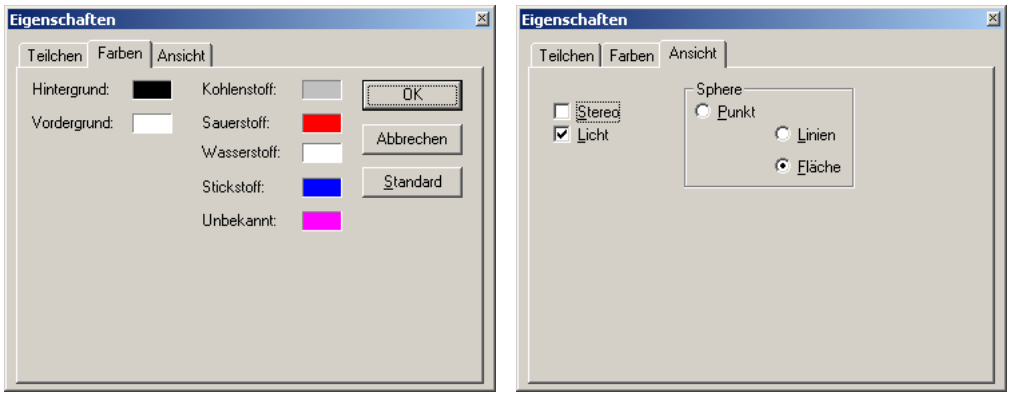


Abbildung 7.3: Teildialoge zur Bearbeitung der Farben und zur Darstellung der Flächen der Teilchen.

stellation, nach Festlegung von Größe und Auflösung des Bildes, als einfache Bitmap-Grafik gespeichert werden. Über die Menüauswahl „Datei“ → „AVI speichern...“ oder über das Symbol ist es möglich definierte Sequenzen als Animation im AVI-Format zu speichern und später mit einem entsprechenden Programm (z. B. dem Windows Mediaplayer) wiederzugeben.

7.2 Analyseverfahren

Die Analyse der Struktur erfolgt über statistische Häufigkeitsverteilungen, welche die Basis zur Bestimmung von Korrelationen zwischen Atomen bilden. Die Paarkorrelationsfunktion ist zum Beispiel eine einfache Methode, um die geometrische Verteilung von kugelsymmetrischen Teilchen zu untersuchen. Sie beschreibt die relative Abweichung der Atomkonzentration einer Sorte Atome von der mittleren Konzentration, in Abhängigkeit vom Abstand zu vorher bestimmten Referenz-

teilchen.

Neben der Struktur bildet die Dynamik den zweiten wichtigen Aspekt in der Analyse von Simulationen. Die Rotationsdiffusion beschreibt die zeitliche Korrelation des Drehverhaltens eines Moleküls oder Teilmoleküls. In vielen Anwendungen ist die Permeabilität eines Stoffes bezüglich eines anderen Stoffes eine wichtige Eigenschaft, die mit der mittleren quadratischen Verschiebung beschrieben werden kann. Diese beschreibt eigentlich die mittlere Beweglichkeit eines Atoms in einem Medium, ohne dabei seine Richtung zu berücksichtigen. Außerdem wird mit Hilfe der mittleren quadratischen Verschiebung und der Einsteinrelation (Gl. (7.27)) der Diffusionskoeffizient bestimmt. Dieser lässt sich über die Green-Kubo-Beziehung (Gl. (7.3)) auch aus der Geschwindigkeitsautokorrelation bestimmen:

$$D_T = \frac{1}{3}\int_0^\infty \langle \vec{\mathbf{v}}(t)\vec{\mathbf{v}}(0)\rangle \,\mathrm{d}t \tag{7.3}$$

Aus der Geschwindigkeitsautokorrelation kann, durch Fouriertransformation, auch die spektrale Dichte bestimmt werden. Eine weitere wichtige Eigenschaft eines wässrigen Systems bilden die Wasserstoffbrückenbindungen. Zu deren Untersuchung wurden Korrelationsfunktionen implementiert, die die Lebensdauer dieser Bindungen analysieren.

7.2.1 Die Korrelationsfunktionen

Betrachtet man zwei Observablen einer Simulation als Funktionen $f(t)$ und $g(t)$, so lässt sich zwischen beiden Funktionen eine Korrelationsfunktion K_{fg} aufstellen, dessen Werte die Größe der Korrelation, d. h. deren zeitliche Abhängigkeit von einander angibt. Für die Kreuzkorrelation K_{fg} ergibt sich folgende Definitionsgleichung:

$$\begin{aligned} K_{fg}(s) &= \langle f(t)g(t+s)\rangle \\ &= \lim_{\tau\to\infty}\frac{1}{\tau}\int_0^\tau f(t)g(t+s)\mathrm{d}t \end{aligned} \tag{7.4}$$

Die Größe dieser Kreuzkorrelation K_{fg} hängt im Ensemblegleichgewicht nur von der zeitlichen Verschiebung, der Zeitdifferenz s zwischen den Messpunkten, nicht jedoch vom absoluten Zeitpunkt t ab, somit gilt:

$$\begin{aligned} K_{fg}(s) &= \langle f(t)g(t+s)\rangle \\ &= \langle f(t_1)g(t_1+s\rangle \qquad \forall \quad t_1 \in \Re \end{aligned} \tag{7.5}$$

In einem realen System müssen bei hinreichend großer Zeitdifferenz s die Messwerte $f(t)$ und $g(t+s)$ unkorreliert sein, d. h.:

$$K_{fg}(s\to\infty) \;\to\; 0 \tag{7.6}$$

Einen Spezialfall der Kreuzkorrelation K_{fg} bildet die in den Algorithmen von „SiDAn“ häufig als Grundlage verwendete Autokorrelation K_f bzw. $K(f,s)$:

$$K(f,s) = K_f(s) = K_{ff}(s) = \langle f(t)f(t+s)\rangle \tag{7.7}$$

die durch Komponentenweise Anwendung der einfachen Autokorrelation $K(f,s)$ auf Vektorfunktionen $\vec{\mathbf{v}}$ erweitert werden kann:

$$K\left(\vec{\mathbf{v}},s\right) = K\left((\vec{\mathbf{v}})_X,s\right) + K\left((\vec{\mathbf{v}})_Y,s\right) + K\left((\vec{\mathbf{v}})_Z,s\right) \tag{7.8}$$

Die Autokorrelation birgt interessante Eigenschaften, so ist zum Beispiel $K_f(0)$ gleich dem Mittelwert des Quadrats der schwankenden Funktion f, d. h. falls $\langle f\rangle = 0$ ist sie gleich ihrem Schwankungsquadrat:

$$K_f(0) = \langle f(t)f(t)\rangle = \left\langle f^2(t)\right\rangle > 0 \tag{7.9}$$

Aus Gleichung (7.5) und Gleichung (7.9) lässt sich allgemein zeigen, das:

$$|K_f(s)| \leq K_f(0) \tag{7.10}$$

Denn ausgehend von $\left\langle [f(t) \pm f(t+s)]^2\right\rangle \geq 0$ gilt:

$$\begin{aligned}
0 &\leq \left\langle [f(t) \pm f(t+s)]^2\right\rangle \\
\Rightarrow 0 &\leq \left\langle f^2(t) + f^2(t+s) \pm 2f(t)f(t+s)\right\rangle \\
\Rightarrow 0 &\leq \left\langle f^2(t)\right\rangle + \left\langle f^2(t+s)\right\rangle \pm 2\left\langle f(t)f(t+s)\right\rangle \\
\Rightarrow 0 &\leq K_f(0) + K_f(0) \pm 2K_f(s) \\
\Rightarrow 0 &\leq K_f(0) \pm K_f(s) \\
\Rightarrow -K_f(0) &\leq K_f(s) \leq K_f(0)
\end{aligned}$$

Aus Gleichung (7.5) lässt sich auch herleiten, dass die Autokorrelationsfunktion eine gerade Funktion ist. Unter der Annahme $t_1 = t - s$ gilt:

$$\begin{aligned}
K_f(s) &= \langle f(t_1)f(t_1+s)\rangle \\
&= \langle f(t-s)f(t)\rangle \\
&= \langle f(t)f(t-s)\rangle \\
&= K_f(-s)
\end{aligned} \tag{7.11}$$

Die Autokorrelation ist somit in ihren Grenzen festgelegt. Sie hat im Ursprung ihr absolutes Maximum und fällt für $t \to \infty$ bzw. $t \to -\infty$ auf Null ab. Ausgenommen natürlich die konstante Funktion, sie produziert eine Autokorrelation mit konstantem Wert (Abbildung 7.4a).

Der Verlauf der Autokorrelation K_f wird unter anderem bestimmt durch die Erhaltungstendenz der ursprünglichen Funktion f. Eine Funktion mit starken oder häufigen „zufälligen“ Änderungen, d. h. mit einer hohen Rauschfrequenz, wird, wie in Abbildung 7.4b und c zu sehen ist, eine Autokorrelationsfunktion K_f mit einem schnelleren Abfall erzeugen, als eine Funktion f mit einer niedrigen Rauschfrequenz. Zeigt eine Funktion f ein periodisches Verhalten, wie es zum Beispiel bei einem Festkörpergitter oder abgeschwächt in Flüssigkeiten auftritt, so zeigt auch die Autokorrelation K_f einen entsprechenden periodischen Verlauf mit der selben Periodenlänge. Dies ist beispielhaft in Abbildung 7.4d gezeigt. Wird der periodische Verlauf durch Stöße mit anderen Molekülen gestört, bleibt, wie in Abbildung 7.4e dargestellt, der periodische Charakter der Autokorrelation erhalten, wobei eine einfache Trennung zwischen dem Signal und dem durch zufällige Stöße verursachten Rauschen möglich ist.

Im Gegensatz zu den bisher betrachteten kontinuierlichen Funktionen, werden die Simulationswerte nur über ein begrenztes Zeitintervall und nur zu equidistanten diskreten Zeitpunkten gewonnen. Aus diesem Grund muss die kontinuierliche Gleichung (7.4) für die Auswertung von Simulationen diskretisiert und auf ein endliches Datenintervall $[0, 1, \ldots, N-1]$ mit $0 \leq s \leq N-1$ eingeschränkt werden:

$$K_{fg}(s) = \frac{1}{N-s} \sum_{t=0}^{N-s-1} f(t)g(t+s) \tag{7.12}$$

Diese Einschränkungen beeinflussen das Verhalten der Korrelationsfunktionen K_{fg} in soweit, dass jetzt natürlich auch die Korrelationsfunktionen diskretisiert sind. Außerdem wird der Funktionsbereich auf ein Intervall beschränkt, wodurch die Güte der Funktionswerte mit zunehmender Zeitdifferenz s stetig abnimmt, denn die Anzahl der gemittelten Korrelationspunkte wird von s und N bestimmt.

7.2.2 Die Paarkorrelationsfunktion

Ein Spezialfall der Korrelation bildet die Paarkorrelationsfunktion. Betrachtet man ein ideales Gas der Dichte ρ, das keine intermolekularen Kräfte besitzt, so sind alle Teilchen gleichmäßig im Raum verteilt. In einem kleinen Volumenelement $\mathrm{d}V$ werden im Mittel $\rho \mathrm{d}V$ Teilchen gefunden, dies gilt natürlich auch für jedes Volumenelement im Abstand r zu einem beliebig gewählten Referenzteilchen a. Die Paarkorrelationsfunktion $g_{AB}(r)$, auch radiale Verteilungsfunktion genannt, beschreibt die relative Verteilung, d. h. die Abweichung von der mittleren Verteilung, der Teilchen b der Sorte B um die Teilchen a der Sorte A in Abhängigkeit von deren Distanz r_{ab}, wobei auch der Spezialfall g_{AA} zulässig ist. Für ein ideales Gas gilt daher immer: $g_{AA}(r) = 1$ für alle r.

In realen Gasen oder Flüssigkeiten existiert ein intermolekulares Potential U, das bei großen Abständen r verschwindet ($U(r \to \infty) \to 0$) und für kleine Abstände r repulsiv divergiert ($U(r \to 0) \to \infty$). D. h. für große Abstände

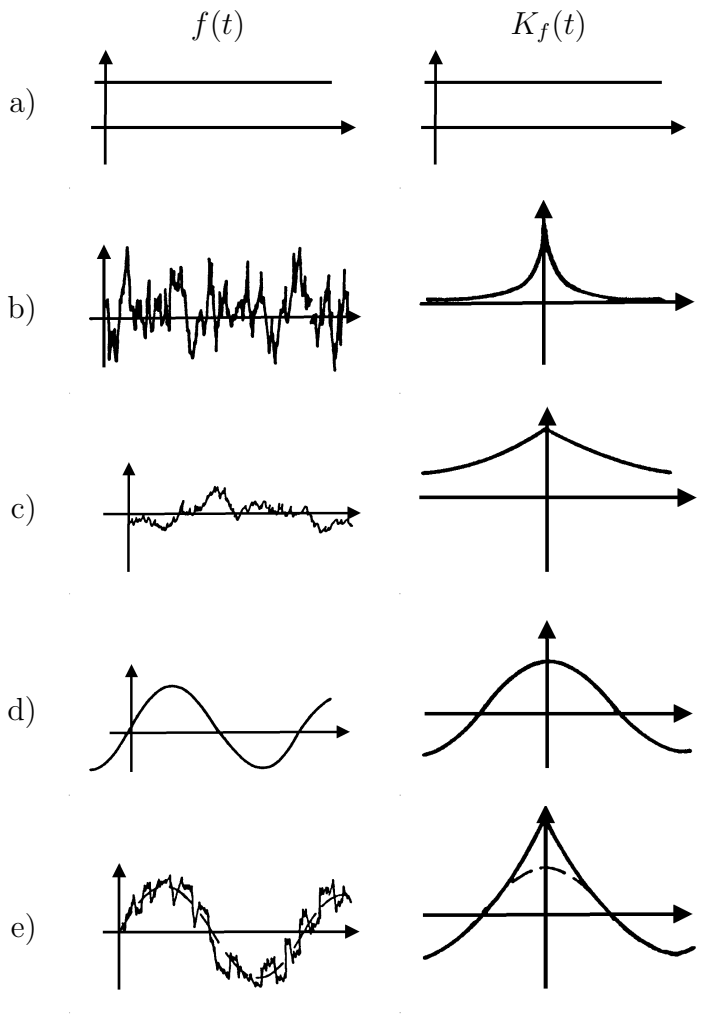


Abbildung 7.4: Autokorrelationen einfacher Signale (teilw. aus [60]).
a) fester Signalwert
b) Hochfrequentes Rauschen
c) Niederfrequentes Rauschen
d) harmonisches Signal
e) harmonisches Signal mit Rauschüberlagerung.

verhalten sich die Teilchen weiterhin wie ein ideales Gas, aber in einer sehr kleinen Umgebung um ein Teilchen kann sich nun kein weiteres Teilchen befinden. Für die Paarkorrelationsfunktion gilt somit:

$$g_{AA}(r \to 0) = 0 \tag{7.13}$$

$$g_{AA}(r \to \infty) = 1 \tag{7.14}$$

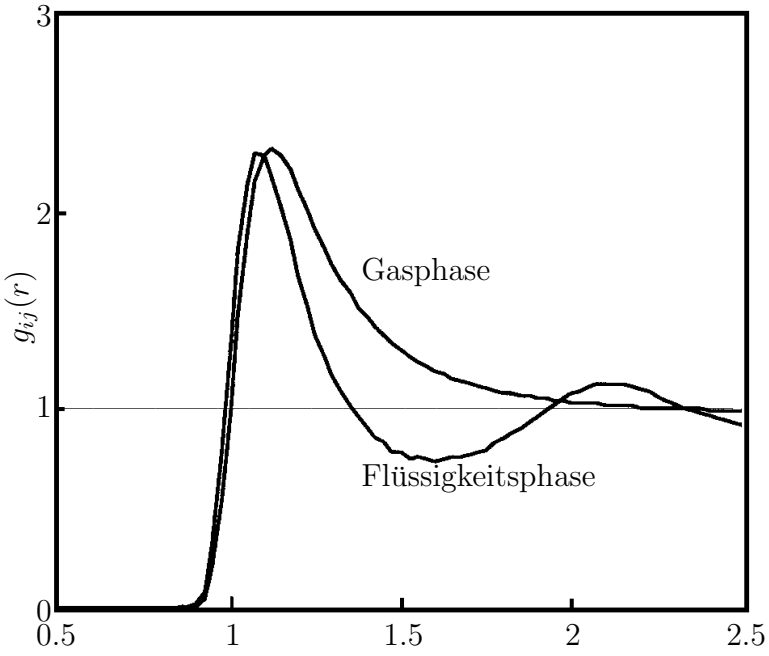


Abbildung 7.5: Paarkorrelation einer Gasphase und einer Flüssigkeitsphase (aus [61]).

Vergleicht man die Paarkorrelation eines realen Gases und einer Flüssigkeit, so zeigen beide in den Extrema ($r \to 0, r \to \infty$) das selbe Verhalten, jedoch unterscheiden sie sich in den mittleren Abständen wesentlich. Wie in Abbildung 7.5 zu erkennen ist, verringert sich die Dichte um ein Teilchen in einem Gas nach dem einzigen Maximum kontinuierlich, dem gegenüber bilden sich in einer Flüssigkeit mehrere lokale Maxima (Schalen) mit einer, mit dem Abstand r, abnehmenden Dichte.

7.2.3 Die Translationsdiffusion

Die Translationsdiffusion wird mit Hilfe der mittleren quadratischen Verschiebung bestimmt, diese lässt sich aus der Geschwindigkeitsautokorrelation bestimmen oder direkt aus den Fickschen Gesetzen herleiten.

Sei $\vec{\mathbf{x}}(0) = \vec{\mathbf{0}}$, so gilt:

$$\vec{\mathbf{x}}(t) = \int_0^t \vec{\mathbf{v}}(t')\mathrm{d}t' \tag{7.15}$$

für die mittlere quadratische Verschiebung $\langle \vec{\mathbf{x}}^2(t) \rangle$ gilt dann:

$$\begin{aligned} \left\langle \vec{\mathbf{x}}^2(t) \right\rangle &= \left\langle \int_0^t \vec{\mathbf{v}}(t')\mathrm{d}t' \int_0^t \vec{\mathbf{v}}(t'')\mathrm{d}t'' \right\rangle \\ &= \int_0^t \mathrm{d}t' \int_0^t \mathrm{d}t'' < \vec{\mathbf{v}}(t')\vec{\mathbf{v}}(t'') > \end{aligned} \tag{7.16}$$

Bei Änderung des Integranden in $s = t'' - t'$ und Vertauschung der Integrationsreihenfolge folgt aus (7.16):

$$\begin{aligned} \left\langle \vec{\mathbf{x}}^2(t) \right\rangle &= \int_0^t \mathrm{d}t' \int_{-t'}^{t-t'} \mathrm{d}s \, \langle \vec{\mathbf{v}}(t')\vec{\mathbf{v}}(s) \rangle \\ &= \int_0^t \mathrm{d}t' \int_{-t'}^{t-t'} \mathrm{d}s K(\vec{\mathbf{v}}, s) \\ &= \int_0^t \mathrm{d}s \int_0^{t-s} \mathrm{d}t' K(\vec{\mathbf{v}}, s) + \int_{-t}^0 \mathrm{d}s \int_{-s}^t \mathrm{d}t' K(\vec{\mathbf{v}}, s) \\ &= \int_0^t \mathrm{d}s (t-s) K(\vec{\mathbf{v}}, s) + \int_{-t}^0 \mathrm{d}s (t+s) K(\vec{\mathbf{v}}, s) \\ &= \int_0^t \mathrm{d}s (t-s) K(\vec{\mathbf{v}}, s) + \int_0^t \mathrm{d}s (t-s) K(\vec{\mathbf{v}}, -s) \end{aligned} \tag{7.17}$$

mit

$$K(\vec{\mathbf{v}}, s) = \langle \vec{\mathbf{v}}(t')\vec{\mathbf{v}}(s) \rangle \tag{7.18}$$

Aus Gleichung (7.17) folgt, wegen der Bedingung $K_f(s) = K_f(-s)$ aus Gleichung (7.11), dass:

$$\left\langle \vec{\mathbf{x}}^2(t) \right\rangle = 2 \int_0^t \mathrm{d}s (t-s) K(\vec{\mathbf{v}}, s) \tag{7.19}$$

Die mittlere quadratische Verschiebung $\langle \vec{\mathbf{x}}^2(t) \rangle$ lässt sich somit aus der Geschwindigkeitsautokorrelation $K(\vec{\mathbf{v}}, s)$ bestimmen.

Einen Ansatz zur Bestimmung des Diffusionskoeffizienten liefert das Ficksche Gesetz, welches die Diffusion von Teilchen durch ein Volumen beschreibt. Die Teilchenstromdichte $\vec{\mathbf{j}}_n$ wird nach dem 1. Fickschen Gesetz berechnet aus dem Diffusionskoeffizienten D und der Teilchenzahldichte $n(\vec{\mathbf{x}}, t)$:

$$\vec{\mathbf{j}}_n(\vec{\mathbf{x}}, t) = -D \nabla n(\vec{\mathbf{x}}, t) \tag{7.20}$$

Strömen aus einem Volumen mehr Teilchen hinaus als hinein, so nimmt die Teilchenzahldichte $n(\vec{\mathbf{x}}, t)$ ab, es besteht folglich der folgende Zusammenhang:

$$\frac{\partial n(\vec{\mathbf{x}}, t)}{\partial t} = -\nabla \vec{\mathbf{j}}_n(\vec{\mathbf{x}}, t) \tag{7.21}$$

Die Kombination des 1. Fickschen Gesetztes (Gl. (7.20)) mit der letzten Gleichung (7.21) führt zum 2. Fickschen Gesetz:

$$\frac{\partial n(\vec{\mathbf{x}},t)}{\partial t} = D\nabla^2 n(\vec{\mathbf{x}},t) \tag{7.22}$$

Die Teilchenzahldichte $n(\vec{\mathbf{x}},t)$ wird zudem normiert, so dass:

$$\int \mathrm{d}\vec{\mathbf{x}} n(\vec{\mathbf{x}},t) = 1 \tag{7.23}$$

Ausgehend von diesen Voraussetzungen, multipliziert man die Gleichung (7.22) des 2. Fickschen Gesetzes mit $\vec{\mathbf{x}}^2$ und integriert anschließend über den Raum, es ergibt sich:

$$\begin{aligned}
\frac{\partial}{\partial t}\int \mathrm{d}\vec{\mathbf{x}}\ \vec{\mathbf{x}}^2 n(\vec{\mathbf{x}},t) &= D\int \mathrm{d}\vec{\mathbf{x}}\ \vec{\mathbf{x}}^2\nabla^2 n(\vec{\mathbf{x}},t) \\
&= D\int \mathrm{d}\vec{\mathbf{x}}\nabla\left(\vec{\mathbf{x}}^2\nabla n(\vec{\mathbf{x}},t)\right) - D\int \mathrm{d}\vec{\mathbf{x}}\nabla\vec{\mathbf{x}}^2\nabla n(\vec{\mathbf{x}},t) \\
&= 0 - 2D\int \mathrm{d}\vec{\mathbf{x}}\ \vec{\mathbf{x}}\nabla n(\vec{\mathbf{x}},t) \\
&= -2D\int \mathrm{d}\vec{\mathbf{x}}\left(\nabla\vec{\mathbf{x}} n(\vec{\mathbf{x}},t)\right) + 2D\int \mathrm{d}\vec{\mathbf{x}}\left(\nabla\vec{\mathbf{x}}\right) n(\vec{\mathbf{x}},t) \\
&= 2*3D\int \mathrm{d}\vec{\mathbf{x}} n(\vec{\mathbf{x}},t) \\
&= 6D
\end{aligned} \tag{7.24}$$

Die mittlere quadratische Verschiebung lässt sich auch als zweites Moment der Teilchenzahldichte $n(\vec{\mathbf{x}},t)$ schreiben, womit:

$$\left\langle \vec{\mathbf{x}}^2(t)\right\rangle = \int \mathrm{d}\vec{\mathbf{x}} n(\vec{\mathbf{x}},t)\vec{\mathbf{x}}^2 \tag{7.25}$$

Diese in Gleichung (7.24) eingesetzt, führt zu:

$$\frac{\partial}{\partial t}\left\langle \vec{\mathbf{x}}^2(t)\right\rangle = 6D \tag{7.26}$$

Die letzte Gleichung, für die, die am Anfang diese Abschnitts eingeführte Einschränkung $\vec{\mathbf{x}}(0) = \vec{\mathbf{0}}$ gilt, kann einfach durch Ersetzen von $\vec{\mathbf{x}}(t)$ durch $\vec{\mathbf{r}}(t) - \vec{\mathbf{r}}(0)$ in eine allgemeine Form überführt werden. Betrachtet man die mittlere quadratische Verschiebung für große Zeiten, so geht sie in eine Gerade über, deren zeitliche Ableitung dem Quotienten $\left\langle \vec{\mathbf{x}}^2(t)\right\rangle/t$ zu strebt. Aus Gleichung (7.26) ergibt sich somit das Einsteinsche Verschiebungsquadrat mit einem Translationsdiffusionskoeffizienten von $D_T = D$:

$$D_T = \lim_{t\to\infty}\frac{\left\langle\left|\vec{\mathbf{r}}_i(t) - \vec{\mathbf{r}}_i(0)\right|^2\right\rangle}{6t} \tag{7.27}$$

7.2.4 Die Rotationsdiffusion

Neben der Translation lässt sich auch die Rotation von freien Molekülen, so wie von drehfähigen funktionellen Gruppen, wie zum Beispiel den Seitengruppen eines Polymers, betrachten. Zur Beschreibung der Rotation werden charakteristische Richtungsvektoren, wie das Dipolmoment oder der Hauptachsenvektor verwendet, um über deren Autokorrelation den Diffusionskoeffizienten D_R der Rotation zu bestimmen.

Die Rotationsdiffusion eines Moleküls lässt sich, wie die Translationsdiffusion aus dem Fickschen Gesetz herleiten. Beschreibe $\omega(\theta, \phi, t)$ die Wahrscheinlichkeit, die Achse eines Moleküls zum Zeitpunkt t mit der Orientierung θ, ϕ zu finden und wird der Zeitpunkt $t = 0$ so gewählt, dass auch die Orientierung $\theta = 0$ ist, so bildet die Anfangswahrscheinlichkeitsverteilung $\omega(\theta, \phi, 0)$ eine δ-Funktion in $\theta = 0$ Richtung.

Die Gleichsetzung der Wahrscheinlichkeitsverteilung $\omega(\theta, \phi, t)$ und der Teilchenzahldichte n der klassischen Beschreibung der Fickschen Gesetze liefert, bei Anwendung auf das 2. Ficksche Gesetz zur Diffusion (Gleichung (7.22)), eine Beschreibung der Rotationsdiffusion von Molekülen [62]:

$$\begin{aligned} \frac{\partial \omega}{\partial t} &= D_R \nabla^2 \omega \\ &= D_R \left\{ \frac{1}{\sin(\theta)\partial\theta} \left(\sin(\theta) \frac{\partial \omega}{\partial \theta} \right) + \frac{1}{\sin^2(\theta)} \frac{\partial^2 \omega}{\partial \phi^2} \right\} \end{aligned} \tag{7.28}$$

Gelöst wird diese Differentialgleichung durch die bekannten Legendre Polynome:

$$\omega(\theta, \phi, t) = \sum_l \left(\frac{2l+1}{4\pi} \right) P_l(\cos(\theta(t)) e^{-l(l+1)D_R t} \tag{7.29}$$

Um eine allgemeine Aussage unabhängig von einem festen Zeitpunkt über die Rotationsdiffusion treffen zu können, ist es notwendig, die Zeit-Autokorrelationsfunktionen heranzuziehen. Die Zeit-Autokorrelationsfunktionen $C_{la}(t)$ für das l-te Legendre Polynom des Moleküls a lautet:

$$\begin{aligned} C_{la}(t) &= < P_l\left(\cos\theta_a(t)\right) > \\ &= < P_l\left(\cos\theta_a(0)\right)\ P_l\left(\cos\theta_a(t)\right) > \\ &= K\Big(P_l\big(\cos\left(\theta_a\right)\big), t\Big) \end{aligned} \tag{7.30}$$

Die Güte der Korrelationsfunktion lässt sich durch eine Mittelung über alle Moleküle einer Sorte A verbessern:

$$C_{lA}(t) = \frac{1}{N_A} \sum_{a=1}^{N_A} C_{la}(t) \tag{7.31}$$

Die hierbei erzeugten Autokorrelationen $C_{lA}(t)$ zeigen für lange Korrelationszeiten t einen exponentiellen Abfall des Korrelationswertes. Dieser exponentielle Abfall kann durch anpassen einer Exponentialfunktion genauer bestimmt werden, hierbei ergibt sich aus dem Exponenten der Funktion die Orientierungskorrelationszeit τ_{Al}:

$$C_{lA}(t) \quad = \quad be^{-t/\tau_{lA}} \tag{7.32}$$

Diese kann wiederum zur Bestimmung des Rotationsdiffusionskoeffizienten D_R der Moleküle der Sorte A verwendet werden:

$$D_R \quad = \quad \frac{1}{l(l+1)\tau_{lA}} \tag{7.33}$$

7.2.5 Die Spektraldichte

Die Spektraldichte beschreibt die Verteilung der Frequenzen, der den Bewegungen eines Atoms zugrunde liegenden Schwingungen. Zur Bestimmung der Spektraldichte $J_{\vec{\mathbf{v}}}(\omega)$ wird die Geschwindigkeits-Autokorrelationsfunktion $K_{\vec{\mathbf{v}}}(t)$ normiert und deren Spektraldichte durch Fouriertransformation berechnet:

$$J_{\vec{\mathbf{v}}}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \frac{K_{\vec{\mathbf{v}}}(t)}{K_{\vec{\mathbf{v}}}(0)} e^{-i\omega t} \mathrm{d}t \tag{7.34}$$

Die Rücktransformation dieser Gleichung liefert wieder, die ursprüngliche normierte Korrelationsfunktion:

$$\frac{K_{\vec{\mathbf{v}}}(t)}{K_{\vec{\mathbf{v}}}(0)} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} J_{\vec{\mathbf{v}}}(\omega) e^{-i\omega s} \mathrm{d}\omega \tag{7.35}$$

Die Geschwindigkeits-Autokorrelationsfunktion $K_{\vec{\mathbf{v}}}(t)$ erfüllt zwei Symmetrieeigenschaften, die eine vereinfachte Berechnung der Spektraldichte $J_{\vec{\mathbf{v}}}(\omega)$ ermöglichen. Zum Einen ist die Korrelationsfunktion $K_{\vec{\mathbf{v}}}(t)$ eine rein reelle Funktion und zum Anderen ist sie wegen Gleichung (7.11) eine gerade Funktion:

$$K^*_{\vec{\mathbf{v}}}(t) \quad = \quad K_{\vec{\mathbf{v}}}(t) \qquad K_{\vec{\mathbf{v}}}(-t) = K_{\vec{\mathbf{v}}}(t) \tag{7.36}$$

Für die Spektraldichte $J_{\vec{\mathbf{v}}}(\omega)$ lassen sich die gleichen Symmetrieeigenschaften herleiten, denn aus Gleichung (7.34) folgt:

$$J^*_{\vec{\mathbf{v}}}(\omega) \quad = \quad \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} K_{\vec{\mathbf{v}}}(t) e^{i\omega s} \mathrm{d}t = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} K_{\vec{\mathbf{v}}}(t) e^{-i\omega s} \mathrm{d}t = J_{\vec{\mathbf{v}}}(\omega) \tag{7.37}$$

$$J_{\vec{\mathbf{v}}}(-\omega) \quad = \quad \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} K_{\vec{\mathbf{v}}}(t) e^{i\omega s} \mathrm{d}t = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} K_{\vec{\mathbf{v}}}(t) e^{-i\omega s} \mathrm{d}t = J_{\vec{\mathbf{v}}}(\omega) \tag{7.38}$$

Die Fourier-Transformationen (7.34) und (7.35) kann man auf Grund der Euler Formel $e^{\pm\omega} = \cos(\omega) \pm i \sin(\omega)$ in einen reellen und einen imaginären

Anteil aufspalten, wobei der imaginäre Anteil von der ungeraden Sinusfunktion bestimmt wird. Da aber die Sinusfunktion wegen den Gleichungen (7.36) und (7.37) verschwinden muss, verschwindet auch das Integral für den Imaginärteil. Die Fourier-Transformationen (7.34) und (7.35) lassen sich zu Fourier-Kosinustransformationen vereinfachen:

$$J_{\vec{\mathbf{v}}}(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} K_{\vec{\mathbf{v}}}(t) \cos(\omega t) \mathrm{d}t = \sqrt{\frac{2}{\pi}} \int_{0}^{\infty} K_{\vec{\mathbf{v}}}(t) \cos(\omega t) \mathrm{d}t \quad (7.39)$$

$$K_{\vec{\mathbf{v}}}(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} J_{\vec{\mathbf{v}}}(\omega) \cos(\omega t) \mathrm{d}\omega = \sqrt{\frac{2}{\pi}} \int_{0}^{\infty} J_{\vec{\mathbf{v}}}(\omega) \cos(\omega t) \mathrm{d}\omega \quad (7.40)$$

Bei der numerischen Umsetzung der Integration geht diese in eine Summation zu diskreten Zeitpunkten t_i, den gespeicherten Punkten der Trajektorie, über. Im Fourierraum wird durch diese Diskretisierung der Zeit, die größtmögliche berechenbare Frequenz $f_{\max}$ begrenzt. Nach dem Theorem von Nyquist [63] und Shannon [64] ist die maximale Frequenz $f_{\max}$ auf die halbe Abtastrate f_{Ny} begrenzt:

$$f_{\max} \leq \frac{1}{2} f_{\mathrm{Ny}} \quad (7.41)$$

mit

$$f_{\mathrm{Ny}} = \frac{1}{t_{i+1} - t_i}$$

7.2.6 Die Bindungsdynamik der Wasserstoffbrückenbindungen

Zur Detektion von Wasserstoffbrückenbindungen wird eine Zustandsfunktion $S_{ij}(t)$ zwischen allen Donator-Akzeptor-Paaren i, j definiert. Eine Wasserstoffbrückenbindung zwischen dem Donator i und dem Akzeptor j wird angenommen, falls die dazu gehörige Zustandsfunktion $S_{ij}(t)$ den Wert 1 hat. Die Zustandsfunktion $S_{ij}(t)$ wird entweder durch das zwischen den Partnern wirkende Potential [5] oder durch die relative Lage [65,66] der Partner zueinander bestimmt. Beide Varianten liefern ähnliche Zustandsfunktionen, jedoch ist eine von der Geometrie abhängige Zustandsfunktion einfacher und effektiver zu implementieren.

Eine Wasserstoffbrückenbindung kann über drei geometrische Randbedingungen beschrieben werden. Die ersten beiden Randbedingungen definieren sich über den Abstand r_{ij} zwischen Donator i und Akzeptor j und über den Abstand r_{iH} zwischen Donator i und Wasserstoffatom. Beide Abstände werden in erster Näherung durch das erste Minimum der zugehörigen radialen Verteilungsfunktion festgelegt. Das dritte Kriterium bestimmt den maximalen Öffnungswinkel θ zwischen r_{ij} und r_{iH}. Die Zustandsfunktion wird mit Hilfe der genannten Randbedingungen

somit wie folgt definiert:

$$S_{ij}(t) = \begin{cases} 1 \quad \text{falls} & r_{ij}(t) \leq 3.60\text{Å}, \\ & r_{Hi}(t) \leq 2.45\text{Å}, \\ & \theta(t) \leq 30^o \\ 0 \quad \text{sonst} & \end{cases} \tag{7.42}$$

Zur zeitunabhängigen Analyse wird die Zeit-Autokorrelation $K(S_{ij}, t)$ der Zustandsfunktion $S_{ij}(t)$ verwendet. Die oben definierte Zustandsfunktion $S_{ij}(t)$ kann zur Bestimmung der Dauer einer Bindung oder zur Analyse des Bindungsverhaltens zu diskreten Zeitpunkten entsprechend erweitert werden. Für die Beschreibung des Bindungsverhalten zu diskreten Zeitpunkten wird die Zustandsfunktion $S_{ij}(t)$ in der speziellen Zustandsfunktion $SD_{ij}(t, \Delta t)$ verwendet, die nicht vom intermediären Bindungsverhalten zwischen den Zeitpunkten t und $t + \Delta t$ abhängt, sondern nur vom Bindungsverhalten am Anfang und am Ende der vorgegebenen Zeitspanne Δt:

$$SD_{ij}(t, \Delta t) = \begin{cases} 1 \quad \text{falls} \quad S_{ij}(t + t') = 1 \quad \forall\, t' \in \{0, \Delta t\} \\ 0 \quad \text{sonst} \end{cases}$$

Die zweite Erweiterung von $S_{ij}(t)$ beschreibt die Dauer einer Bindung. Die von $S_{ij}(t)$ abgeleitete Zustandsfunktion $SC_{ij}(t, \Delta t)$ besitzt nur dann den Wert 1, falls die Bindung im gesamten vorgegebenen Intervall bestanden hat:

$$SC_{ij}(t, \Delta t) = \begin{cases} 1 \quad \text{falls} \quad S_{ij}(t + t') = 1 \quad \forall\, t' \in [0, \Delta t] \\ 0 \quad \text{sonst} \end{cases}$$

Verwendung finden diese Erweiterungen zum Beispiel bei der Analyse der Frequenz eines Bindungswechsels zwischen den Seitengruppen, die über die Bindungen einen Einfluss auf die Beweglichkeit der Seitengruppen ausüben. Ein weiteres Anwendungsgebiet bildet die Beschreibung des „Hopping-Mechanismus“ von Probemolekülen entlang von Polymer-Seitengruppen.

7.3 Auswertung mit SiDAn

Das Programm „SiDAn“ analysiert die vom Programm „dynC“ (Abschnitt 4.4 und 9.4) erstellten Trajektorien. Bearbeitet werden können, aus Gründen der Kompatibilität, nur Trajektorien des alten Typs, d. h. zur Auswertung wird die alte Datenstruktur verwendet. Für „dynC“ bedeutet dies, dass die Trajektorien über das Schlüsselwort „out TrajOld“ erzeugt werden müssen.

Die Auswertungsmethode und die für die Auswertung benötigten Dateien und Parameter werden über die in Abbildung 7.6 abgebildete Eingabemaske festgelegt. Eine neue leere Eingabemaske wird über den Menüpunkt „Datei“ → „Neu“, Strg-N oder den Eintrag 🗋 in der Symbolleiste erzeugt. Die Speicherung der

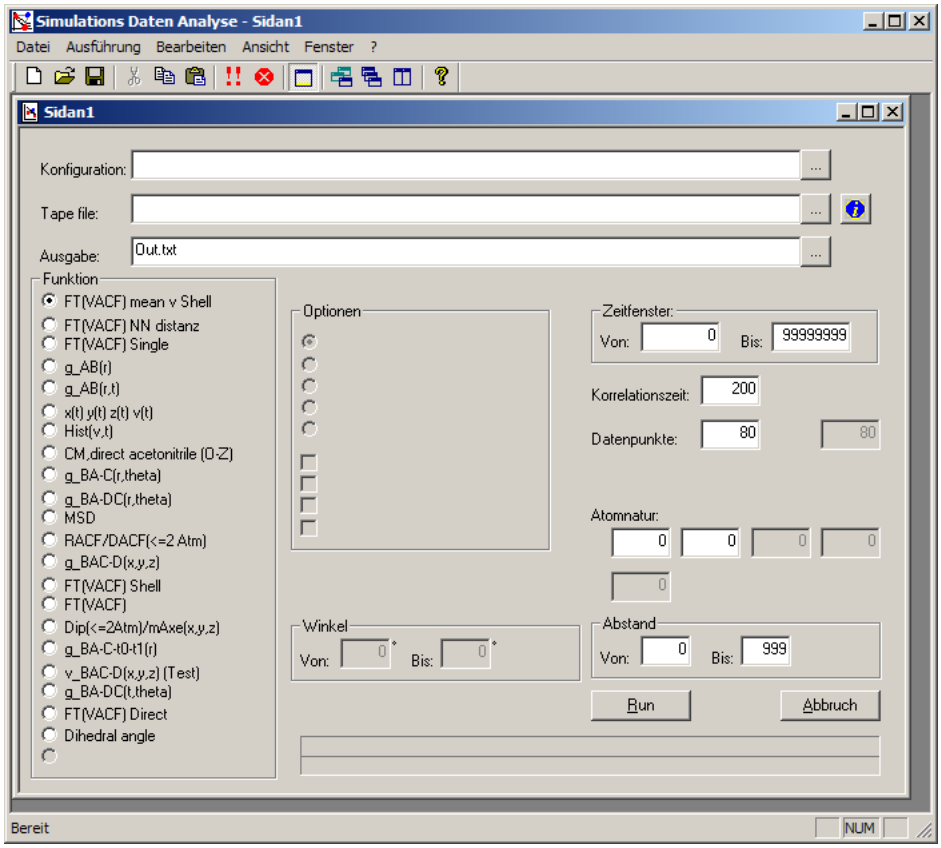


Abbildung 7.6: Eingabemaske von SiDAn.

Daten erfolgt über die Menüpunkte „Datei“ → „Speichern“, → „Speichern unter...“, der Tastenkombination Strg-S bzw. über das Symbol der Symbolleiste, die gespeicherte Eingabemaske kann dann über „Datei“ → „Öffnen...“, Strg-O oder wieder eingelesen werden. Mit Hilfe der Methoden des Menüpunktes „Bearbeiten“ können einzelne Parameter aus anderen Programmen bzw. Einträgen ausgeschnitten (), kopiert () und eingefügt () werden. Sind Konfigurationsdatei und Trajektorien-Datei angegeben, so sind die in Abbildung 7.7 gezeigten Informationen über den Schalter abrufbar. Beide Dateien können auch einzeln über die Schalter ... festgelegt werden, dort werden die Dateien mit Hilfe des Dialoges zur Auswahl von Dateien aus dem Verzeichnisbaum ausgewählt.

Mehrere Auswertungen können automatisch sequentiell abgearbeitet werden, wenn die zugehörigen Eingabemasken in SiDAn geladen sind, und die sequentielle Ausführung über den Menüpunkt „Ausführung“ → „Start“ oder über das Symbol gestartet wurde. Durch drücken des Symbols oder die Schaltfläche Abbruch wird die aktuelle Auswertung abgebrochen.

Im Folgenden werden die im Bereich „Funktion“ aufgeführten Auswertefunktionen beschrieben, zur Vereinfachung der hierfür notwendigen Formeln und zur

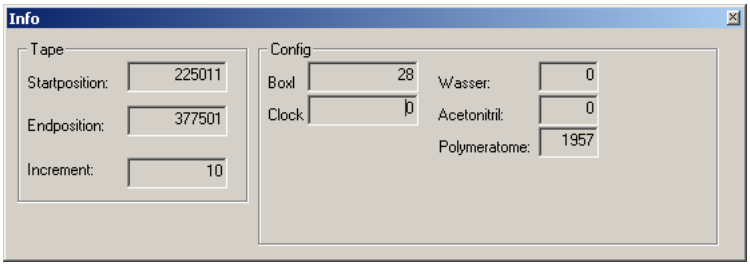


Abbildung 7.7: Informationsfenster zur Simulation.

Kürzung der Schreibweise wird eine vereinfachende Nomenklatur verwendet. Das Volumen L^3 eines Simulationsraumes mit der Kantenlänge L wird durch V ersetzt. Der Ortsvektor des a-ten Atoms der Sorte A zum Zeitpunkt t wird mit dem Ausdruck $\vec{\mathbf{r}}_{Aa}(t)$ beschrieben, außerdem wird der Abstand zwischen zwei Atomen vereinfacht durch:

$$\vec{\mathbf{r}}_{Aa;Bb}(t) = \vec{\mathbf{r}}_{Bb}(t) - \vec{\mathbf{r}}_{Aa}(t) \qquad \text{und} \qquad r_{Aa;Bb}(t) = |\vec{\mathbf{r}}_{Aa;Bb}(t)| \tag{7.43}$$

Eine weitere Vereinfachung der Schreibweise erfolgt durch Einführung des Einheitsvektors $\vec{\mathbf{e}}_{Aa;Bb}(t)$:

$$\vec{\mathbf{e}}_{Aa;Bb}(t) = \frac{\vec{\mathbf{r}}_{Aa;Bb}(t)}{r_{Aa;Bb}(t)} \tag{7.44}$$

Eine häufig verwendete Funktion, die auch in „SiDAn“ implementierte wurde, ist die in Abschnitt 7.2.2 eingeführte Paarkorrelation- bzw. radiale Verteilungsfunktion $g_{AB}(r)$. Die Paarkorrelation $g_{AB}(r)$ beschreibt die Abweichung der lokalen Dichte von der mittleren Dichte als Funktion der Distanz $\vec{\mathbf{r}}_{Aa;Bb}(t)$ zwischen den Atomen aus A und B. Zur Bestimmung der Paarkorrelation $g_{AB}(r)$ ist es notwendig das Volumen einer Kugelschale zu bestimmen: Das Volumen $V_S(r_1, r_2)$ einer Kugelschale mit Innendurchmesser r_1 und Außendurchmesser r_2 ergibt sich aus:

$$V_S(r_1, r_2) \;=\; \frac{4}{3}\pi \left(r_2^3 - r_1^3\right) \tag{7.45}$$

Um die Atome in einem bestimmten Teilvolumen zu zählen, ist es außerdem notwendig, festzustellen, ob sich ein Atom in einem bestimmten Raumelement befindet, hierfür wird folgende Rechteckimpuls-Funktion δ als Hilfsfunktion definiert, die sich auch aus der Überlagerung zweier Heaviside-Funktionen H ergibt:

$$\begin{aligned} \delta(a)_b &= H(a) - H(a-b) \\ &= \begin{cases} 1 & 0 \le a < b \\ 0 & \text{sonst} \end{cases} \end{aligned} \tag{7.46}$$

Mit $\delta(r_{Aa} - r)_{\Delta r}$ lässt sich zum Beispiel feststellen, ob ein Atom a zum Zeitpunkt t in einer Kugelschale der dicke Δr und dem inneren Radius r war. Für

die Paarkorrelation $g_{AB}(r)$ ergibt sich unter Zuhilfenahme von (7.45) und (7.46):

$$\begin{aligned} g_{AB}(r) &= \frac{V}{V_S(r, r+\Delta r)\,(t_1 - t_0 + 1)\,N_A N_B} \\ &\quad \sum_{t=t_0}^{t_1} \sum_{a=1}^{N_A} \sum_{b=1}^{N_B} \bar{\delta}_{Aa;Bb}(t) \delta\Big(r_{Aa;Bb}(t) - r\Big)_{\Delta r} \end{aligned} \tag{7.47}$$

mit der, die Rechteckimpuls-Funktion δ invertierenden, Funktion $\bar{\delta}$:

$$\begin{aligned} \bar{\delta}_{Aa;Bb}(t) &= 1 - \delta\left(\vec{\mathbf{r}}_{Bb}(t) - \vec{\mathbf{r}}_{Aa}(t)\right) \\ &= 1 - \delta\left(r_{Aa;Bb}(t)\right) \end{aligned} \tag{7.48}$$

Teilt man die Trajektorien der Atome einer Simulation in lokale Zeitfenster Δt auf und wendet die Paarkorrelation $g_{AB}(r)$ auf die einzelnen Zeitfenster an, so erhält man eine zeitabhängige radiale Verteilungsfunktion $g_{AB}(r,t)$ zur Verfolgung individueller Atome der Sorte A relativ zu Atomen der Sorte B. Eingesetzt wurde die zeitabhängige radiale Verteilungsfunktion $g_{AB}(r,t)$ zum Beispiel in [67,68,69] zur Verfolgung von Acetonitril A an den Polymer-Seitengruppen B.

$$\begin{aligned} g_{AB}(r,t) &= \frac{V}{V_S(r, r+\Delta r)\Delta t N_A N_B} \\ &\quad \sum_{\tau=t}^{t+\Delta t-1} \sum_{a=1}^{N_A} \sum_{b=1}^{N_B} \bar{\delta}_{Aa;Bb}(\tau) \delta\Big(r_{Aa;Bb}(\tau) - r\Big)_{\Delta r} \end{aligned} \tag{7.49}$$

Die radiale Verteilungsfunktion $g_{AB}(r)$ ist ausgelegt für radial symmetrische Moleküle, bei unsymmetrischen Molekülen verfälscht deren gerichteter Anteil aber die Aussagekraft der radialen Verteilungsfunktion $g_{AB}(r)$. Deshalb ist bei unsymmetrischen Molekülen oder Gruppen eine winkelabhängige Verteilungsfunktion aussagekräftiger [69,70,71]. Die Lage und die Zuordnung der einzelnen Atome einer winkelabhängigen Verteilungsfunktion $g_{BA-C}(r,\theta)$ charakterisiert das Schema in Abbildung 7.8. Hierin ist die Verteilung zwischen einem Molekül bzw. einer Gruppe, dessen Hauptachse von den Atomen a und b gebildet wird, und dem korrelierten Atom c dargestellt.

Für die Bestimmung einer winkelabhängigen Verteilungsfunktion ist es notwendig das Volumen des betrachteten Kugelschalenausschnitts zu bestimmen. Das Volumen eines Kugelausschnitts einer Kugel vom Radius r mit Ausschnitthöhe h (Abbildung 7.9a) ist nach Bronstein [72]:

$$V(r,h) = \frac{2}{3}\pi r^2 h \tag{7.50}$$

Die Ausschnitthöhe h ist in der Verteilungsfunktion jedoch nicht explizit bekannt. Sie muss aus dem Winkel θ und dem Radius r bestimmt werden, hierzu

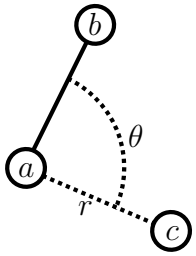


Abbildung 7.8: Schematischer Aufbau der winkelabhängigen Verteilungsfunktion $g_{BA-C}(r, \theta)$ zwischen einer aus den Atomen a und b der Sorten A und B gebildeten Hauptachse und den Atomen c der Sorte C.

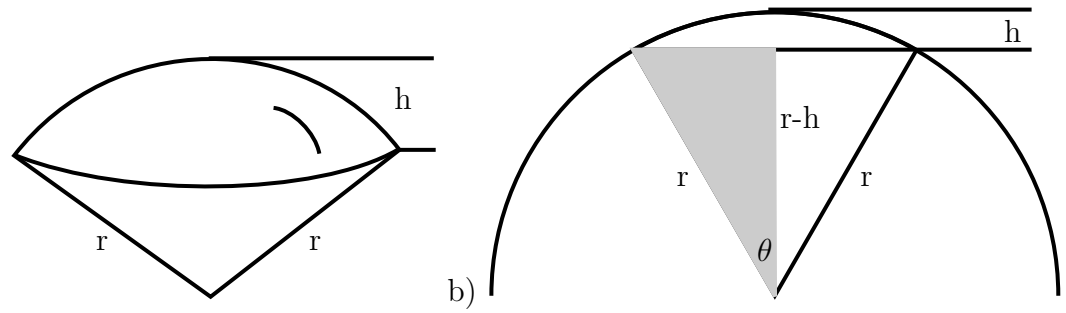


Abbildung 7.9: a) Ausschnitt aus einer Kugel vom Radius r mit Ausschnitthöhe h. b) Schnitt durch Kugelausschnitt zur Bestimmung von h aus Radius r und Öffnungswinkel θ.

wird ein rechtwinkliges Dreieck mit Kathete $r - h$ und Hypotenuse r in den Kugelausschnitt gelegt (Abbildung 7.9b). Es gilt:

$$\begin{aligned} \cos(\theta) &= \frac{r-h}{r} \\ \Rightarrow r - h &= r\cos(\theta) \\ \Rightarrow h &= r\Big(1 - \cos(\theta)\Big) \end{aligned} \tag{7.51}$$

Das Volumen $V_{KA}(r, \theta_1, \theta_2)$ des zirkularen Ausschnittes von θ_1 bis θ_2 eines Kugelausschnitts vom Radius r ergibt sich somit aus den Gleichungen (7.50) und (7.51) zu:

$$\begin{aligned} V_{KA}(r, \theta_1, \theta_2) &= V\Big(r, r\big(1 - \cos(\theta_2)\big)\Big) - V\Big(r, r\big(1 - \cos(\theta_1)\big)\Big) \\ &= \frac{2}{3}\pi r^2 * r\Big(1 - \cos(\theta_2)\Big) - \frac{2}{3}\pi r^2 * r\Big(1 - \cos(\theta_1)\Big) \\ &= \frac{2}{3}\pi r^3\Big(1 - \cos(\theta_2) - \Big(1 - \cos(\theta_1)\Big)\Big) \\ &= \frac{2}{3}\pi r^3\Big(\cos(\theta_1) - \cos(\theta_2)\Big) \end{aligned} \tag{7.52}$$

Dem entsprechend ist das Volumen des Kugelschalenausschnitts der Schale (r_1, r_2) im zirkularen Ausschnitt zwischen θ_1 und θ_2 gerade $V_{SA}(r_1, r_2, \theta_1, \theta_2)$:

$$\begin{aligned} V_{SA}(r_1, r_2, \theta_1, \theta_2) &= V_{KA}(r_2, \theta_1, \theta_2) - V_{KA}(r_1, \theta_1, \theta_2) \\ &= \frac{2}{3}\pi(r_2^3 - r_1^3)\Big(\cos(\theta_1) - \cos(\theta_2)\Big) \end{aligned} \tag{7.53}$$

Neben dem Volumen des gewählten Kugelschalenausschnitts muss für jedes Atom a der Sorte A die Anzahl der Atome der Sorte C in diesem Kugelschalenausschnitt bestimmt werden. Diese Anzahl wird für jedes Atom a über die Summation aller Atome der Sorte C kombiniert mit einer die Atome selektierenden Rechteckimpuls-Funktion bestimmt. Hierfür ist es, ergänzend zur radialen Verteilungsfunktion, notwendig zusätzlich eine Rechteckimpuls-Funktion zur Auswahl des Winkelbereichs zu definieren. Mit Hilfe des Arkuskosinus wird der Winkel zwischen den Richtungsvektoren von a nach b ($\vec{\mathbf{e}}_{Aa;B,A(a,b)}(t)$) und von a nach c ($\vec{\mathbf{e}}_{Aa;Cc}(t)$) bestimmt. Die Rechteckimpuls-Funktion selektiert für die Paarverteilung die Atome, dessen Winkel des Arkuskosinus zwischen θ und $\theta + \Delta\theta$ liegen. Für die winkelabhängige Verteilungsfunktion $g_{BA-C}(r, \theta)$ gilt:

$$\begin{aligned} g_{BA-C}(r, \theta) = & \frac{V}{V_{SA}(r, r+\Delta r, \theta, \theta+\Delta\theta)(t_1 - t_0 + 1)N_A N_B N_C} \\ & * \sum_{t=t_0}^{t_1} \sum_{a=1}^{N_A} \sum_{c=1}^{N_C} \bar{\delta}_{Aa;Cc}(t) \delta\Big(r_{Aa;Cc}(t) - r\Big)_{\Delta r} \\ & * \sum_{b=1}^{N_{BA}} \delta\left(\arccos\left(\vec{\mathbf{e}}_{Aa;B,A(a,b)}(t)\vec{\mathbf{e}}_{Aa;Cc}(t)\right) - \theta\right)_{\Delta\theta} \end{aligned} \tag{7.54}$$

Mit Hilfe der Funktion $A(a, b)$ und über die Laufvariable b werden die N_{BA} Atome aus B aufgelistet, die zum a-ten Atom der Sorte A gehören.

In vielen Fällen ist es günstiger nicht alle möglichen Kombinationen der Atome $A(a, b)$, die zum a-ten Atom der Sorte A gehören, in die Verteilungsfunktion aufzunehmen, sondern die Ortskoordinate $\langle\vec{\mathbf{r}}_a(t)\rangle$ eines virtuellen, aus den N_{BA} Atomen $A(a, b)$ gemittelten, Atoms zu verwenden. Zum Beispiel ergibt sich für Wasser (a=O, A(a,b)={H, H}) daraus automatisch die Dipolachse bzw. Hauptachse des Moleküls:

$$\langle\vec{\mathbf{r}}_a(t)\rangle = \frac{1}{N_{BA}} \sum_{b=1}^{N_{BA}} \vec{\mathbf{r}}_{AaBA(a,b)}(t) \tag{7.55}$$

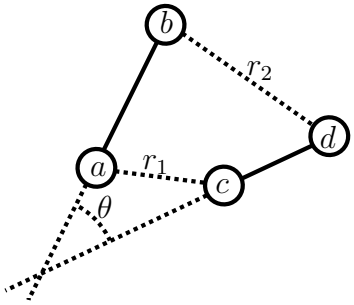


Abbildung 7.10: Schematischer Aufbau der winkelabhängigen Verteilungsfunktion $g_{BA-CD}(r,\theta)$ der aus den Atomen a und b der Sorten A und B und aus Atomen c und d der Sorten C und D gebildeten Hauptachsen.

Für die modifizierte Verteilungsfunktion gilt damit:

$$\begin{aligned} g_{BA-C}(r,\theta) &= \frac{V}{V_{SA}(r,r+\Delta r,\theta,\theta+\Delta\theta)(t_1-t_0+1)N_AN_C} \\ &\quad * \sum_{t=t_0}^{t_1}\sum_{a=1}^{N_A}\sum_{c=1}^{N_C} \bar{\delta}_{Aa;Cc}(t)\delta\Big(r_{Aa;Cc}(t)-r\Big)_{\Delta r} \\ &\quad * \delta\left(\arccos\left(\frac{\langle\vec{\mathbf{r}}_a(t)\rangle}{|\langle\vec{\mathbf{r}}_a(t)\rangle|}\vec{\mathbf{e}}_{Aa;Cc}(t)\right)-\theta\right)_{\Delta\theta} \end{aligned} \tag{7.56}$$

Mit der zu letzt beschriebenen Verteilungsfunktion ist es nicht möglich, die relative Verteilung zweier Moleküle oder Gruppen in Abhängigkeit von ihrer Distanz r und dem relativen Winkel θ ihrer Hauptachsen zu bestimmen. Hierfür wurde die Verteilungsfunktion $g_{BA-CD}(r,\theta)$ implementiert. Bei ihr werden, wie der Abbildung 7.10 zu sehen ist, zwei Hauptachsen und deren relative Lage zueinander als Kriterium für die Verteilungsfunktion verwendet. Anwendung findet diese Funktion zum Beispiel bei der Bestimmung der Ausrichtung von Acetonitril Molekülen in Abhängigkeit vom Abstand zu einem weiteren Molekül [69]. Durch leichte Modifikation der Bezugsvektoren der Gleichung (7.56) ergibt sich:

$$\begin{aligned} g_{BA-DC}(r,\theta) &= \frac{V}{V_{SA}(r,r+\Delta r,\theta,\theta+\Delta\theta)(t_1-t_0+1)N_AN_CN_{BA}N_{DC}} \\ &\quad * \sum_{t=t_0}^{t_1}\sum_{a=1}^{N_A}\sum_{b=1}^{N_{BA}} \bar{\delta}_{Aa;B,A(a,b)}(t)\sum_{c=1}^{N_C}\bar{\delta}_{Cc;Aa}(t)\bar{\delta}_{Cc;B,A(a,b)}(t) \\ &\quad * \sum_{d=1}^{N_{DC}} \bar{\delta}_{D,C(c,d);Aa}(t)\bar{\delta}_{D,C(c,d);B,A(a,b)}(t)\bar{\delta}_{D,C(c,d);Cc}(t) \\ &\quad * \delta\Big(r_{AaCc}(t)-r\Big)_{\Delta r} \\ &\quad * \delta\left(\arccos\left(\vec{\mathbf{e}}_{Aa;B,A(a,b)}(t)\vec{\mathbf{e}}_{Cc;D,C(c,d)}(t)\right)-\theta\right)_{\Delta\theta} \end{aligned} \tag{7.57}$$

mit

$$N_{BA} = \frac{N_B}{N_A} \qquad N_{DC} = \frac{N_D}{N_C} \tag{7.58}$$

Die in Gleichung (7.57) aufgeführte Variante verwendet den in Abbildung 7.10 mit r_1 bezeichneten Abstand als Atomabstand r. Soll statt dieses Abstandes, der in Abbildung 7.10 mit r_2 bezeichnete Abstand verwendet werden, so muss der grau unterlegter Bereich in Gleichung (7.57) durch folgenden Therm ersetzt werden:

$$\delta\Big(r_{B,A(a,b);D,C(c,d)}(t) - r\Big)_{\Delta r} \tag{7.59}$$

Weitere Varianten der Verteilungsfunktionen $g_{BA-C}(r,\theta)$ und $g_{BA-CD}(r,\theta)$ ergeben sich zum Beispiel durch die Beschränkung auf einen festen Winkelbereich $[\theta_L, \theta_U]$, diese führt zu einer Modifikation der Verteilungsfunktion in den rot gedruckten Bereichen:

$$\begin{aligned} g_{BA-C}(r,\theta_L,\theta_U) = {} & \frac{V}{V_{SA}(r, r+\Delta r, \theta_F, \theta_T)(t_1 - t_0 + 1) N_A N_C N_{BA}} \\ & * \sum_{t=t_0}^{t_1} \sum_{a=1}^{N_A} \sum_{c=1}^{N_C} \bar{\delta}_{Aa;Cc}(t) \sum_{b=1}^{N_{BA}} \bar{\delta}_{B,A(a,b);Aa}(t) \bar{\delta}_{B,A(a,b);Cc}(t) \delta\Big(r_{Aa;Cc}(t) - r\Big)_{\Delta r} \\ & * \begin{cases} \theta_L \leq \arccos\left(\vec{\mathbf{e}}_{Aa;B,A(a,b)}(t) \vec{\mathbf{e}}_{Aa;Cc}(t)\right) \leq \theta_U & 1 \\ \text{sonst} & 0 \end{cases} \end{aligned} \tag{7.60}$$

Eine weitere Modifikation der Verteilungsfunktion erlaubt es, die Orientierung von Molekülen in einem festgelegten Bereich, wie zum Beispiel den Hydratschalen um Wasser zu bestimmen.

In der Verteilungsfunktion $g_{BA-DC}(t, \Delta t, \theta, r_L, r_U)$ wird eine durch r_L und r_U begrenzte Schale definiert, in der die Auswertung erfolgt. Zur Verfolgung des sogenannten „Hopping“-Mechanismusses, der das Lösen und Anlagern von Molekülen an ausgewählten Seitengruppen beschreibt, ist es möglich die Trajektorie in Intervalle der Länge δt aufzuspalten und auszuwerten, und so eine zeitliche Auflösung der Verteilungsfunktion zu erhalten:

$$\begin{aligned} g_{BA-DC}(t, \Delta t, \theta, r_L, r_U) = {} & \frac{V}{V_{SA}(r, r+\Delta r, \theta, \theta + \Delta\theta)(\Delta t + 1) N_A N_{BA} N_C N_{DC}} \\ & * \sum_{\tau=t}^{t+\Delta t} \sum_{a=1}^{N_A} \sum_{b=1}^{N_{BA}} \bar{\delta}_{Aa;B,A(a,b)}(t) \sum_{c=1}^{N_C} \bar{\delta}_{Cc;Aa}(t) \bar{\delta}_{Cc;B,A(a,b)}(t) \\ & * \sum_{d=1}^{N_{DC}} \bar{\delta}_{D,C(c,d);Aa}(t) \bar{\delta}_{D,C(c,d);B,A(a,b)}(t) \bar{\delta}_{D,C(c,d);Cc}(t) \\ & * \delta\left(\arccos\left(\vec{\mathbf{e}}_{Aa;B,A(a,b)}(t) \vec{\mathbf{e}}_{Cc;D,C(c,d)}(t)\right) - \theta\right)_{\Delta\theta} \\ & * \begin{cases} r_L \leq r_{Aa;B,A(a,b)}(t) \leq r_U & 1 \\ \text{sonst} & 0 \end{cases} \end{aligned} \tag{7.61}$$

Wie in Gleichung (7.57) ist es auch hier möglich, nicht den Abstand zwischen der Standardauswahl der Atome der Sorte A und B, sondern den Abstand zwischen den Atomen der Sorte C und D für die Verteilungsfunktion zu verwenden. Hierzu wird der grau unterlegte Bereich in Gleichung 7.61 durch folgenden Term ersetzt:

$$r_{Cc;D,C(c,d)}(t) \tag{7.62}$$

Eine weitere Verteilungsfunktion beschreibt die relative Verteilung zweier Gruppen in Abhängigkeit von einem ausgewählten Torsionswinkel ϕ einer Molekülsorte:

$$\begin{aligned} g_{ABC-BCD}(\phi) = & \frac{1}{(t_1 - t_0 + 1) N_U N_{AU} N_{BU} N_{CU} N_{DU}} \\ & * \sum_{t=t_0}^{t_1} \sum_{u=1}^{N_U} \sum_{a=1}^{N_{AU}} \sum_{b=1}^{N_{BU}} \bar{\delta}_{A,U(u,a);B,U(u,b)}(t) \sum_{c=1}^{N_{CU}} \bar{\delta}_{C,U(u,c);A,U(u,a)}(t) \bar{\delta}_{C,U(u,c);B,U(u,b)}(t) \\ & * \sum_{d=1}^{N_{DU}} \bar{\delta}_{D,U(u,d);A,U(u,a)}(t) \bar{\delta}_{D,U(u,d);B,U(u,b)}(t) \bar{\delta}_{D,U(u,d);C,U(u,c)}(t) \\ & * \delta \Bigg(\arccos \Big(\left(\vec{\mathbf{e}}_{B,U(u,b);A,U(u,a)}(t) \times \vec{\mathbf{e}}_{B,U(u,b);C,U(u,c)}(t) \right) \\ & \qquad * \left(\vec{\mathbf{e}}_{B,U(u,b);C,U(u,c)}(t) \times \vec{\mathbf{e}}_{C,U(u,c);D,U(u,d)}(t) \right) \Big) - \phi \Bigg)_{\Delta\phi} \end{aligned} \tag{7.63}$$

Der Torsionswinkel ϕ wird hierbei, wie in Abbildung 7.11 gezeigt, durch die von den Atomen a, b und c der entsprechenden Sorten A, B, C und den Atomen b, c und d (der Sorte D) aufgespannten Ebenen abc und bcd bestimmt. Anwendung findet diese spezielle Verteilungsfunktion beispielsweise bei der Analyse der Wahrscheinlichkeitsverteilung einer cis- bzw. trans-Stellung.

Die Auswertung, der bereits in Abschnitt 7.2.4 beschriebene Rotationsdiffusion bzw. Rotationsautokorrelation, wird ebenfalls von „SiDAn“ unterstützt. Zur Detektion der Rotation ist es notwendig, einen Richtungsvektor $\vec{\mathbf{P}}_a(t)$ zu definieren, der die Ausrichtung der durch die Atome a der Sorte A festgelegten Moleküle bzw. Seitengruppen bestimmt. Neben den Atomen a wird für jedes Molekül auch mindestens ein Hilfsatom b der Sorte B des selben Moleküls bzw. der selben Seitengruppe benötigt. Bei mehreren Hilfsatomen b pro Molekül wird ein mittlerer Richtungsvektor $\vec{\mathbf{P}}_a(t)$ berechnet, der sich aus den N_{BA} Richtungsvektoren zwischen a und b ergibt:

$$\vec{\mathbf{P}}_a(t) = \frac{\left({}^1\!/_{N_{BA}} \sum_{b=1}^{N_{BA}} \vec{\mathbf{r}}_{B,A(a,b)}(t) \right) - \vec{\mathbf{r}}_{Aa}(t)}{\left| \left({}^1\!/_{N_{BA}} \sum_{b=1}^{N_{BA}} \vec{\mathbf{r}}_{B,A(a,b)}(t) \right) - \vec{\mathbf{r}}_{Aa}(t) \right|} \tag{7.64}$$

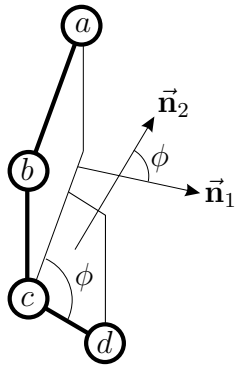


Abbildung 7.11: Torsionswinkel ϕ zwischen den von den Atomen a, b, c und b, c, d aufgespannten Ebenen mit den Normalenvektoren $\vec{\mathbf{n}}_1 = (\vec{\mathbf{e}}_{Bb;Aa} \times \vec{\mathbf{e}}_{Bb;Cc})$ und $\vec{\mathbf{n}}_2 = (\vec{\mathbf{e}}_{Bb;Cc} \times \vec{\mathbf{e}}_{Cc;Dd})$.

Die Rotationsautokorrelation $C_{1AB}(s)$ berechnet sich, unter Verwendung der Richtungsvektoren $\vec{\mathbf{P}}_a(t)$ und von Gleichung (7.30), aus der Mittelung der Korrelation über alle Atome der Sorte A:

$$\begin{aligned} C_{1AB}(s) = \langle P_1\left(\cos\left(\theta_{AB}(s)\right)\right)\rangle &= \\ \mathrm{RACF}_{AB}(s) &= \frac{1}{N_A}\sum_{a=1}^{N_A} K\left(\vec{\mathbf{P}}_a, s\right) \\ &= \frac{1}{N_A}\sum_{a=1}^{N_A}\frac{1}{N-s}\sum_{t=0}^{N-s-1}\vec{\mathbf{P}}_a(t)\vec{\mathbf{P}}_a(t+s) \\ &= \frac{1}{N-s}\sum_{t=0}^{N-s-1}\frac{1}{N_A}\sum_{a=1}^{N_A}\vec{\mathbf{P}}_a(t)\vec{\mathbf{P}}_a(t+s) \end{aligned} \quad (7.65)$$

Hierbei bestimmt das Vektorprodukt $\vec{\mathbf{P}}_a(t)\vec{\mathbf{P}}_a(t+s)$ die Drehung, genauer den Winkel zwischen dem a-ten Atom der Sorte A zum Zeitpunkt t und zum Zeitpunkt $(t+s)$.

Die mittlere quadratische Verschiebung, die bereits in Abschnitt 7.2.3 behandelt wurde, ist wie folgt implementiert:

$$\mathrm{MSD}_A(\tau) = \left\langle \left|\vec{\mathbf{u}}_A(\tau) - \vec{\mathbf{u}}_A(0)\right|^2 \right\rangle = \frac{1}{tN_A}\sum_{s=1}^{t}\sum_{a=1}^{N_A}\left|\vec{\mathbf{u}}_a(\tau+s) - \vec{\mathbf{u}}_a(s)\right|^2 \quad (7.66)$$

wobei die Vektoren $\vec{\mathbf{u}}_A(t)$ die realen Koordinaten der Atome A enthalten. D. h. für die $\vec{\mathbf{u}}_A(t)$ werden die periodischen Randbedingungen mit Hilfe der Funktion InvPBC aus Abschnitt 9.1.2 wieder aus den Ortsvektoren $\vec{\mathbf{r}}_{Aa}(t)$ herausgerechnet:

$$\vec{\mathbf{u}}_a(t) = \mathrm{InvPBC}\Big(\vec{\mathbf{r}}_{Aa}(t) - \vec{\mathbf{u}}_a(t-\Delta t)\Big) + \vec{\mathbf{u}}_a(t-\Delta t) \quad (7.67)$$

Aus der mittleren quadratischen Verschiebung wird unter Verwendung der Einsteinrelation (Gleichung (7.27)) der Diffusionskoeffizient D_T bestimmt. Mit den hier verwendeten Variablen ergibt sich:

$$D_T = \lim_{t\to\infty} \frac{1}{6t} \left\langle \left| \vec{\mathbf{u}}_A(t) - \vec{\mathbf{u}}_A(0) \right|^2 \right\rangle$$

Ebenfalls auf die mit Gleichung (7.67) definierten realen Koordinaten $\vec{\mathbf{u}}_A(t)$ greift die Funktion „$X(t)Y(t)Z(t)v(t)$“ zu, sie speichert die Realkoordinaten $\vec{\mathbf{u}}_a(t)$ und die Geschwindigkeiten $v_a(t)$ aller Atome a der anzugebenden Sorte A in die ausgewählte Ausgabe-Datei. Weitere einfache Funktionen sind „Hist(v,t)“, die die Häufigkeitsverteilung für Atom a aus A bei Geschwindigkeiten zwischen $v - {}^{\Delta v}/_2$ und $v + {}^{\Delta v}/_2$ im Zeitintervall $[t - {}^{\Delta t}/_2, t + {}^{\Delta t}/_2]$ bestimmt, so wie die Funktion „CM, direct acetonitrile (O-Z)“. Sie gibt für jeden Zeitpunkt im Auswertungsintervall die Position der Massenschwerpunkte und die Geschwindigkeiten der Acetonitril- bzw. Wassermoleküle aus.

Eine eigene Gruppe bilden die Funktionen, die die Geschwindigkeit zur Datenbasis nehmen und ebenfalls die Dynamik beschreiben. Hervorzuheben ist hier die Geschwindigkeits-Autokorrelationsfunktionen. Sie selbst ist jedoch nicht von Interesse, sondern deren Fourier-Transformierte, die die in Abschnitt 7.2.5 beschriebene Spektraldichte liefert. Korrelationen sind auf Grund von Gleichung (7.11) immer gerade Funktionen. Für sie genügt eine Fourier-Kosinustransformation $\hat{f}_C(x) = \hat{f}(x)$, da der Sinus $\hat{f}_S(x)$ nur einen Beitrag zu ungeraden Funktionsanteilen liefert, die hier fehlen. In SiDAn implementiert wurde die Fourier-Kosinustransformation $\hat{f}_C(x)$ nach Filon [73] aus dem Buch von Allen und Tildesley [32], die folgende Integration ausführt:

$$F_C(y) = \sqrt{\frac{2}{\pi}} \int_0^\infty f(x) \cos(xy) \mathrm{d}x \tag{7.68}$$

In SiDAn sind insgesamt sechs verschiedene Varianten der Geschwindigkeitsautokorrelation und dessen durch Fouriertransformation berechneten Spektren implementiert.

Die Funktion „FT(VACF) Single“ liefert die Geschwindigkeitsautokorrelation $\mathrm{vacf}_a(t)$ eines Atoms a, für deren Berechnung auf die Autokorrelationsfunktion aus Gleichung (7.7) zurückgegriffen wird:

$$\mathrm{vacf}_a(t) = \frac{K\left(\vec{\mathbf{v}}_a, t\right)}{K\left(\vec{\mathbf{v}}_a, 0\right)} \tag{7.69}$$

Eine größere Genauigkeit bietet die Beobachtung aller Atome einer Sorte A. Deshalb bestimmt die Funktion „FT(VACF)“ die allgemeine Geschwindigkeitsauto-

korrelation $\mathrm{vacf}_A(t)$ aller Atome a der Sorte A:

$$\mathrm{vacf}_A(t) = \frac{\sum_{a=1}^{N_A} K\Big(\vec{\mathbf{v}}_a, t\Big)}{\sum_{a=1}^{N_A} K\Big(\vec{\mathbf{v}}_a, 0\Big)} \tag{7.70}$$

Häufig werden durch Mittelungen lokale Effekte vernachlässigt, deshalb wurde auch die Funktion „FT(VACF) mean v shell“ zur Bestimmung der Geschwindigkeitsautokorrelation $\mathrm{vacf}_{A<B>}(t)$ implementiert. Sie berechnet die Geschwindigkeitsautokorrelation der Atome b der Sorte B in einer Schale um die Atome a der Sorte A. Formal ergibt sich damit:

$$\mathrm{vacf}_{A<B>}(t) = \frac{\sum_{a=1}^{N_A} K\Big(\vec{\mathbf{s}}_a, t\Big)}{\sum_{a=1}^{N_A} K\Big(\vec{\mathbf{s}}_a, 0\Big)} \tag{7.71}$$

mit der mittleren Geschwindigkeit $\vec{\mathbf{s}}_a$ aller Atome b der Sorte B im Abstand zwischen r_0 und r_1:

$$\vec{\mathbf{s}}_a = \frac{\sum_{b=1}^{N_B} \delta\left(r_{ab}, r_0, r_1\right) \vec{\mathbf{v}}_b(t)}{\sum_{b=1}^{N_B} \delta\left(r_{ab}, r_0, r_1\right)} \tag{7.72}$$

Die durch r_0 und r_1 begrenzte Schale wird durch folgende Rechteckimpuls-Funktion δ beschrieben:

$$\begin{aligned} \delta\left(r, r_0, r_1\right) &= \delta\left(r - r_0\right)_{r_1 - r_0} \\ &= H(r - r_0) - H(r - r_1) \\ &= \begin{cases} 1 & r_0 \leq r < r_1 \\ 0 & \text{sonst} \end{cases} \end{aligned} \tag{7.73}$$

Die Funktion „FT(VACF) mean v shell“ berechnet außerdem die Geschwindigkeitsautokorrelation $\mathrm{dvacf}_{A<B>}(t)$ der relativen Geschwindigkeit, gemittelt über die Atome b der Sorte B bezüglich der Atome a der Sorte A:

$$\mathrm{dvacf}_{A<B>}(t) = \frac{\sum_{a=1}^{N_A} K\Big(\vec{\mathbf{v}}_a - \vec{\mathbf{s}}_a, t\Big)}{\sum_{a=1}^{N_A} K\Big(\vec{\mathbf{v}}_a - \vec{\mathbf{s}}_a, 0\Big)} \tag{7.74}$$

Es ist natürlich auch möglich, die Korrelation nicht über die mittlere Geschwindigkeit $\vec{\mathbf{s}}_a$ zu bestimmen, sondern die Geschwindigkeitsautokorrelation über alle Atome b der Sorte B zu bestimmen, die sich in den durch r_0, r_1 und A definierten Schalen befinden. Die Funktion „FT(VACF) Shell“ liefert die ent-

sprechende Geschwindigkeitsautokorrelation $\mathrm{vacf}_{AB}(s)$:

$$\begin{aligned}\mathrm{vacf}_{AB}(s) &= \frac{\sum_{a=1}^{N_A}\sum_{b=1}^{N_B} K\left(\delta\left(r_{ab}, r_0, r_1\right)\vec{\mathbf{v}}_b, s\right)}{\sum_{a=1}^{N_A}\sum_{b=1}^{N_B} K\left(\delta\left(r_{ab}, r_0, r_1\right)\vec{\mathbf{v}}_b, 0\right)} \\ &= \frac{\sum_{a=1}^{N_A}\sum_{b=1}^{N_B}\sum_{t=1}^{T} \delta\left(r_{ab}(t), r_0, r_1\right)\vec{\mathbf{v}}_b(t)\delta\left(r_{ab}(t+s), r_0, r_1\right)\vec{\mathbf{v}}_b(t+s)}{\sum_{a=1}^{N_A}\sum_{b=1}^{N_B}\sum_{t=1}^{T} \delta\left(r_{ab}(t), r_0, r_1\right)\delta\left(r_{ab}(t+s), r_0, r_1\right)}\end{aligned} \tag{7.75}$$

Analog zur vorhergehenden Funktion liefert auch diese zusätzlich die entsprechende Auswertung über die relativen Geschwindigkeiten:

$$\mathrm{dvacf}_{AB}(s) = \frac{\sum_{a=1}^{N_A}\sum_{b=1}^{N_B} K\left(\delta\left(r_{ab}, r_0, r_1\right)\left(\vec{\mathbf{v}}_a - \vec{\mathbf{v}}_b\right), s\right)}{\sum_{a=1}^{N_A}\sum_{b=1}^{N_B} K\left(\delta\left(r_{ab}, r_0, r_1\right)\left(\vec{\mathbf{v}}_a - \vec{\mathbf{v}}_b\right), 0\right)} \tag{7.76}$$

In der Anlagerungsphase, in der sich Atome der Sorte A an andere Atome b der Sorte B von Seitengruppen anlagern, wird die Funktion „FT(VACF) NN“ zur Untersuchung des Spektrums der Atome der Sorte A verwendet. Die Funktion bestimmt die Geschwindigkeitsautokorrelation $\mathrm{vacf}_{A-B}(s)$ der Atome A, die sich in einer Schale vom Radius r_0 bis r_1 um A und in nächster Nachbarschaft zu einem Atom der Sorte B befinden. D. h. es werden nur die Atome von A berücksichtigt, die den kleinsten Abstand zu einem Atom der Sorte B besitzen, aber immer noch innerhalb der festgelegten Schale liegen.

$$\mathrm{vacf}_{A-B}(s) = \frac{\sum_{a=1}^{N_A} K\Big(\delta\left(d_a, r_0, r_1\right)\vec{\mathbf{v}}_a, s\Big)}{\sum_{a=1}^{N_A} K\Big(\delta\left(d_a, r_0, r_1\right)\vec{\mathbf{v}}_a, 0\Big)} \tag{7.77}$$

mit

$$d_a(t) = \min_{b\in N_B}\left\{r_{ab}(t)\right\} \tag{7.78}$$

Alternativ können auch Atome, die innerhalb des Radius r_0 liegen, durch entsprechende Wahl im Bereich der Optionen, a priori ausgeschlossen werden, dann ändert sich $d_a(t)$ in:

$$d_a(t) = \min_{b\in N_B}\left\{r_{ab}(t) | r_{ab}(t) \geq r_0\right\} \tag{7.79}$$

Die Funktion „FT(VACF) Direct“ bestimmt die Geschwindigkeitsautokorrelation $\vec{\mathrm{v}}_{ABC}(s)$ von A in einem, mit Hilfe der in Gleichung (7.81) beschriebenen Transformationsmatrix $\mathrm{M}_{abc}(t)$, aus dem von A, B und C lokal aufgespannten

Koordinatensystem:

$$\vec{\mathrm{v}}_{ABC}(s) = \begin{pmatrix} \frac{\sum_{a=1}^{N_A}\sum_{b=1}^{N_{BA}}\sum_{c=1}^{N_{CA}} K\left((\mathrm{M}_{abc}(t)\vec{\mathbf{v}}_a)_X, s\right)}{\sum_{a=1}^{N_A}\sum_{b=1}^{N_{BA}}\sum_{c=1}^{N_{CA}} K\left((\mathrm{M}_{abc}(t)\vec{\mathbf{v}}_a)_X, 0\right)} \\ \\ \frac{\sum_{a=1}^{N_A}\sum_{b=1}^{N_{BA}}\sum_{c=1}^{N_{CA}} K\left((\mathrm{M}_{abc}(t)\vec{\mathbf{v}}_a)_Y, s\right)}{\sum_{a=1}^{N_A}\sum_{b=1}^{N_{BA}}\sum_{c=1}^{N_{CA}} K\left((\mathrm{M}_{abc}(t)\vec{\mathbf{v}}_a)_Y, 0\right)} \\ \\ \frac{\sum_{a=1}^{N_A}\sum_{b=1}^{N_{BA}}\sum_{c=1}^{N_{CA}} K\left((\mathrm{M}_{abc}(t)\vec{\mathbf{v}}_a)_Z, s\right)}{\sum_{a=1}^{N_A}\sum_{b=1}^{N_{BA}}\sum_{c=1}^{N_{CA}} K\left((\mathrm{M}_{abc}(t)\vec{\mathbf{v}}_a)_Z, 0\right)} \end{pmatrix} \tag{7.80}$$

Die beiden letzten Auswertemethoden bestimmen die räumliche Verteilungen von Atomen bzw. Ausrichtungen um ein Molekül bzw. einer funktionellen Gruppe. Die Visualisierung dieser Verteilungen erfolgt mit dem im nächsten Kapitel beschriebenen Programm „TTPlot.“ Für die Bestimmung der relativen Lage eines Atoms d zu einer durch a, b und c festgelegten funktionellen Gruppe, ist es notwendig, die Koordinaten von d in ein von der funktionellen Gruppe bestimmtes Koordinatensystem zu transformieren. Die Basisvektoren $\vec{\mathbf{b}}_1, \vec{\mathbf{b}}_2, \vec{\mathbf{b}}_3$ dieses neuen Koordinatensystems werden nach folgenden vier Kriterien aus den Ortsvektoren der Atome a, b und c der funktionellen Gruppe bestimmt:

- Ursprung des neuen Koordinatensystems in $\vec{\mathbf{r}}_{Aa}(t)$.
- Bindung zwischen a und b bestimmt Richtung des ersten Basisvektors $\vec{\mathbf{b}}_1(t)$: $\vec{\mathbf{b}}_1(t) = \vec{\mathbf{e}}_{Aa;Bb}(t)$
- Normalenvektor der von $\vec{\mathbf{b}}_1(t)$ und $\vec{\mathbf{r}}_{Aa;Cc}(t)$ aufgespannten Ebene legt den zweiten Basisvektor $\vec{\mathbf{b}}_2(t)$ fest: $\vec{\mathbf{b}}_2(t) = \vec{\mathbf{b}}_1(t) \times \vec{\mathbf{e}}_{Aa;Cc}(t)$
- Der dritte Basisvektor $\vec{\mathbf{b}}_3(t)$ liegt senkrecht auf den beiden anderen Basisvektoren: $\vec{\mathbf{b}}_3(t) = \vec{\mathbf{b}}_1(t) \times \vec{\mathbf{b}}_2(t)$

Aus den Zeilenvektoren der Basisvektoren $\vec{\mathbf{b}}_1, \vec{\mathbf{b}}_2, \vec{\mathbf{b}}_3$ wird die Transformationsmatrix $\mathrm{M}_{abc}(t)$ gebildet:

$$\mathrm{M}_{abc}(t) = \begin{pmatrix} \vec{\mathbf{b}}_1^T(t) \\ \vec{\mathbf{b}}_2^T(t) \\ \vec{\mathbf{b}}_3^T(t) \end{pmatrix} \tag{7.81}$$

Die Funktion „g_BAC-D(x,y,z)“ bestimmt die räumliche Paarverteilungsfunktion $g_{BAC-D}(\vec{\mathbf{r}})$ der Atome d der Sorte D in Abhängigkeit von dem durch die Atome a, b und c der entsprechenden Atomsorten erzeugten und durch die obige

Transformationsmatrix $\mathrm{M}_{abc}(t)$ festgelegten lokalen Koordinatensystemen:

$$g_{BAC-D}\left(\vec{\mathbf{r}}\right) = \frac{V}{\left(t_1 - t_0 + 1\right) N_A N_{BA} N_{CA} N_D V\left(\Delta\vec{\mathbf{r}}\right)} \sum_{t=t_0}^{t_1} \sum_{a=1}^{N_A} \sum_{b=1}^{N_{BA}} \bar{\delta}_{Aa;B,A(a,b)}(t) \sum_{c=1}^{N_{CA}} \bar{\delta}_{Aa;C,A(a,c)}(t) \bar{\delta}_{B,A(a,b);C,A(a,c)}(t) \sum_{d=1}^{N_D} \bar{\delta}_{Aa;Dd}(t) \bar{\delta}_{B,A(a,b);Dd}(t) \bar{\delta}_{C,A(a,c);Dd}(t) \delta\left(\mathrm{M}_{abc}(t)\vec{\mathbf{r}}_{Aa;Dd}(t) - \vec{\mathbf{r}}\right)_{\Delta\vec{\mathbf{r}}} \tag{7.82}$$

mit der räumlichen Rechteckimpuls-Funktion $\delta\left(\vec{\mathbf{a}}\right)_{\Delta\vec{\mathbf{r}}}$ über das Raumelement $\Delta\vec{\mathbf{r}}$, die sich aus den Rechteckimpuls-Funktionen der Gleichung (7.46) zusammensetzt:

$$\delta\left(\vec{\mathbf{a}}\right)_{\Delta\vec{\mathbf{r}}} = \delta\left(\left(\vec{\mathbf{a}}\right)_x\right)_{\left(\Delta\vec{\mathbf{r}}\right)_x} \delta\left(\left(\vec{\mathbf{a}}\right)_y\right)_{\left(\Delta\vec{\mathbf{r}}\right)_y} \delta\left(\left(\vec{\mathbf{a}}\right)_z\right)_{\left(\Delta\vec{\mathbf{r}}\right)_z} \tag{7.83}$$

so wie dem Volumen $V\left(\Delta\vec{\mathbf{r}}\right)$ eines Raumelements $\Delta\vec{\mathbf{r}}$:

$$V\left(\Delta\vec{\mathbf{r}}\right) = \left(\Delta\vec{\mathbf{r}}\right)_x \left(\Delta\vec{\mathbf{r}}\right)_y \left(\Delta\vec{\mathbf{r}}\right)_z \tag{7.84}$$

Beispiele für die Anwendung der räumlichen Paarverteilungsfunktion auf Wasser, Acetonitril oder Polymer-Seitengruppen sind in [69, 70, 71] und in den Abbildungen 8.7 bis 8.10 zu finden.

Die zweite Funktion „Dip(<= 2Atm)/mAxe(x,y,z)“ wurde zur Visualisierung der Orientierung eines Dipols bzw. einer durch zwei Atome bestimmten Molekülachse relativ zu einer, ein Koordinatensystem definierenden, Gruppe von drei Atomen bestimmt. Mit Hilfe der Transformationsmatrix $\mathrm{M}_{abc}(t)$ und den drei Atomen a, b und c wird ein standardisiertes, lokales Koordinatensystem erzeugt, durch das die relative Lage des Dipols bzw. der Achse bestimmt wird. Der Dipol bzw. die Achse selbst werden durch die Atome d und e festgelegt. Wie Abbildung 7.12 zeigt, wird der Winkel ϕ_0 durch die von d und e und die von a und d gebildeten Achsen festgelegt (Anwendung in Abbildung 7.13a). Bei entsprechender Wahl im Optionsfeld, kann auch der Winkel ϕ_1 verwendet werden, der durch die von d und e und die von a und b gebildeten Achsen (X-Achsenvektor) festgelegt wird (Anwendung in Abbildung 7.13b).

7.4 Visualisierung mit TTPlot

Die beiden Funktionen „g_BAC-D(x,y,z)“ und „Dip(<= 2Atm)/mAxe(x,y,z)“ liefern Verteilungsdaten bzw. Ausrichtungsinformationen für jeden Raumpunkt des Simulationsraumes. Zur graphischen Visualisierung dieser Daten wurde „TTPlot“ entwickelt. Das Programm stellt die Daten entweder farbkodiert in zwei dimensionalen Schnittebenen, wie in Abbildung 7.14 gezeigt, oder im drei dimensionalen Raum, durch Festlegung eines Wertes als Grenzfläche (Abbildung 8.11), dar. Die

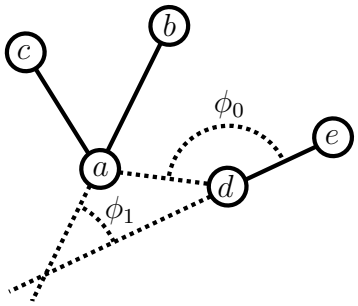


Abbildung 7.12: Bestimmung der Raumwinkel ϕ_0 bzw. ϕ_1 zwischen den Atomen a, b und c einer funktionellen Gruppe und den Atomen d und e eines Moleküls.

a) b)

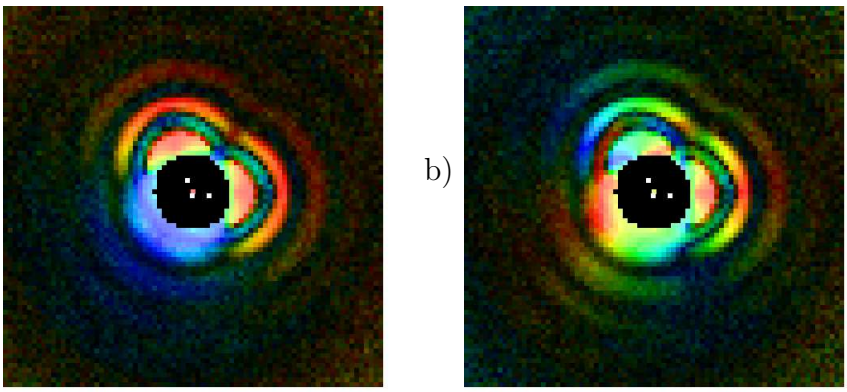


Abbildung 7.13: Farbkodierte Darstellung der Ausrichtung des Dipolvektors des Wassers bezüglich der O-O Bindung mit einem zweiten Wassermolekül (a) bzw. der X-Achse (b).

Steuerung des Programms erfolgt generell über die Menüauswahl, häufig gibt es auch eine Tastenkombination oder einen Schalter in der Symbolleiste. Im Folgenden werden die einzelnen Menüeinträge zusammen mit deren Tastenkombination und dem zugehörigen Symbol kurz erklärt:

- „Datei“ → „Neu“, Strg-N, : Erzeugt ein Dokument mit Testdaten, die in einem neuen Fenster dargestellt werden.
- „Datei“ → „Öffnen...“, Strg-O, : Fragt nach einer TTPlot Datendatei (*.ttd), deren Daten zur Erzeugung eines neuen Dokuments verwendet werden, das Dokument wird als 2D Schnitt in einem Fenster dargestellt.
- „Datei“ → „Schließen“: Schließt ein Fenster und gegebenenfalls auch das zugehörige Dokument.
- „Datei“ → „Speichern unter...“ Strg-S (): Speichert die Darstellung des aktiven Fenster in einer Bitmap-Datei (*.BMP), wobei die in „Optionen“

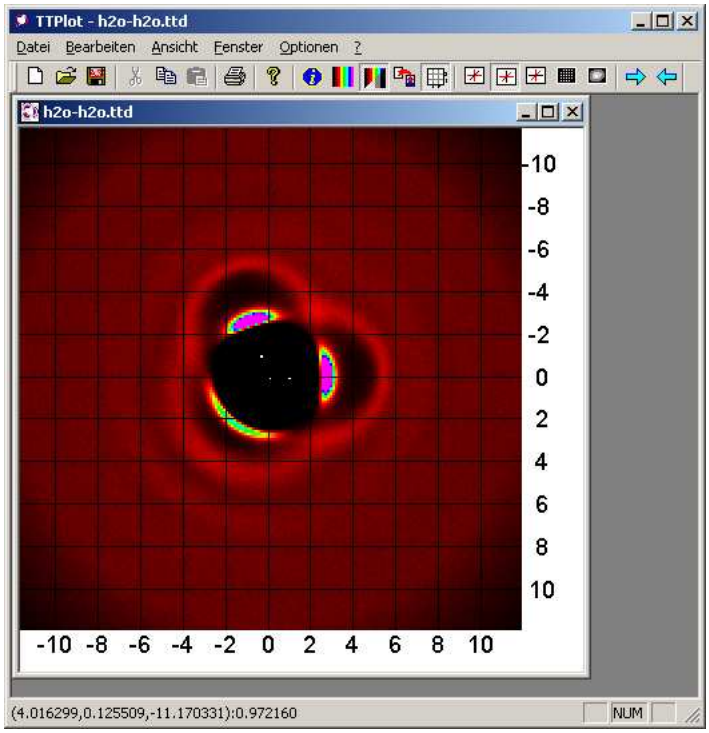


Abbildung 7.14: Das Programm TTPlot mit einer farblich dargestellten Häufigkeitsverteilung der Sauerstoffatome von Wassermolekülen an einem Wassermolekül (Skala in Ångström).

→ „Auflösung BMP...“ festgelegte Auflösung verwendet wird.

- „Datei“ → „Drucken...“ Strg-P (🖶): Öffnet den Windows-Druckdialog, mit dem der Inhalt des aktiven Fensters ausgedruckt werden kann.
- „Datei“ → „Seitenansicht...“: Stellt das Druckbild des aktiven Fensters dar.
- „Datei“ → „Druckereinrichtung...“: Öffnet den Windows-Druckdialog, um die Einstellungen für den Drucker festzulegen.
- „Bearbeiten“ → „Kopieren“ Strg-C (🗐): Kopiert den Inhalt des aktiven Fensters in die Zwischenablage.
- „Ansicht“ → „Symbolleiste“: Deaktiviert oder aktiviert die Darstellung der Symbolleiste.
- „Ansicht“ → „Statusleiste“: Deaktiviert oder aktiviert die Darstellung der Statusleiste.
- „Ansicht“ → „Farbe“ (🎨): Wechselt zwischen der Farbdarstellung und der Grauwertdarstellung des 2D Schnitts im aktiven Fenster.

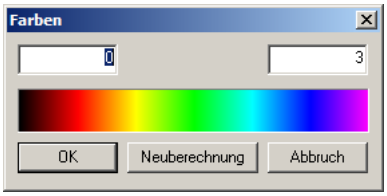


Abbildung 7.15: Im Dialog Farben werden der untere Grenzwert, bis zu dem alle Werte schwarz dargestellt werden, und der obere Grenzwert, ab dem alle Werte violett dargestellt werden, festgelegt.

- „Ansicht“ → „Koordinatensystem“ ( ): Blendet das verwendete Koordinatensystem im 2D Schnitt ein bzw. aus.
- „Ansicht“ → „Farben...“ ( ): In der 2D Darstellung wird der in Abbildung 7.15 gezeigten Dialog geöffnet, mit dem der obere und untere Grenzwert des Farbspektrums für alle 2D Darstellungen festgelegt werden kann. Alle Werte unterhalb des unteren Grenzwertes werden schwarz dargestellt, alle oberhalb des oberen Grenzwertes violett und alle Werte zwischen den beiden Grenzwerten werden in Abhängigkeit vom Wert selbst in der entsprechenden Spektralfarbe dargestellt. In einer 3D Darstellung kann der Grenzwert für die Grenzflächen angezeigt oder geändert werden.
- „Ansicht“ → „XY“ ( ): Wechselt in die 2D Ansicht mit einem Schnitt in der XY-Ebene.
- „Ansicht“ → „XZ“ ( ): Wechselt in die 2D Ansicht mit einem Schnitt in der XZ-Ebene.
- „Ansicht“ → „YZ“ ( ): Wechselt in die 2D Ansicht mit einem Schnitt in der YZ-Ebene.
- „Ansicht“ → „3D-Punkte“ ( ): Wechselt in die 3D Ansicht mit einer durch Punkte dargestellten Grenzfläche. Diese Darstellung ist wesentlich schneller als die Flächendarstellung.
- „Ansicht“ → „3D-Fläche“ ( ): Wechselt in die 3D Ansicht mit einer durch Flächenelemente dargestellten Grenzfläche, zwei Beispiele sind in Abbildung 8.11 gezeigt.
- „Ansicht“ → „Vor“ [+] ( ): In der 2D Ansicht wird die Schnittebene um ein Teilstück weiter entlang der gewählten Achse geschoben. In der 3D Ansicht wird der Grenzwert für die Grenzfläche um 0.1 erhöht.
- „Ansicht“ → „Zurück“ [-] ( ): In der 2D Ansicht wird die Schnittebene um ein Teilstück entlang der gewählten Achse zurück genommen. In der 3D Ansicht wird der Grenzwert für die Grenzfläche um 0.1 verringert.

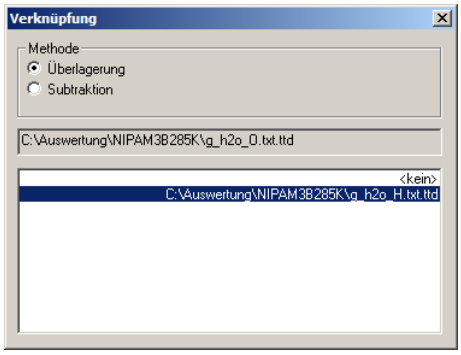


Abbildung 7.16: Die Verknüpfung zweier Dokumente zu einer gemeinsamen Darstellung erfolgt über diesen Dialog, in dem die Methode der Verknüpfung und das zu verknüpfende Dokument ausgewählt werden.

- „Ansicht“ → „Verknüpfen...“ ( ): Eine sehr intuitive Möglichkeit zwei Dokumente miteinander zu verbinden bietet die Verknüpfung beider in einer gemeinsamen Darstellung. Implementiert sind, wie in Abbildung 7.16 zu sehen ist, zwei Varianten der Verknüpfung:

  Überlagerung: Bei der Überlagerung werden in beiden Dokumenten die unterschiedlichen Werte durch unterschiedliche Intensitäten dargestellt, wobei die Daten des ursprünglichen Dokumentes in rot, die Daten des verknüpften Dokumentes in blau dargestellt werden. Die Anwendung dieses Verfahrens auf Polymere, Wasser und Acetonitril zeigen die Abbildungen 8.7 bis 8.10 so wie Abbildungen in den Veröffentlichungen [69, 70, 71].

  Subtraktion: Die Subtraktion berechnet an jedem Raumpunkt einen neuen Funktionswert, der sich aus der Subtraktion des Wertes des verknüpften Dokuments vom Wert des ursprünglichen Dokumentes ergibt. Der neue Funktionswert wird bei positivem Vorzeichen in rot, bei negativem Vorzeichen in blau dargestellt. Die Intensität wird, wie bei der Überlagerung vom Funktionswert bestimmt.

- „Fenster“: Mit den Menüeinträgen in „Fenster“ lässt sich die Anordnung der Fenster ändern oder es lassen sich neue Fenster von bestehenden Dokumenten erzeugen, die das Dokument in einer neuen Ansicht darstellen können.

- „Optionen“ → „Schrift...“: Bestimmt die Schriftart und Farbe der Achsenbeschriftungen.

- „Optionen“ → „Auflösung BMP...“: Bestimmt die Auflösung bzw. Größe der in die Zwischenablage kopierten oder über „Datei“ → „Speichern unter...“ gespeicherten Bilder.

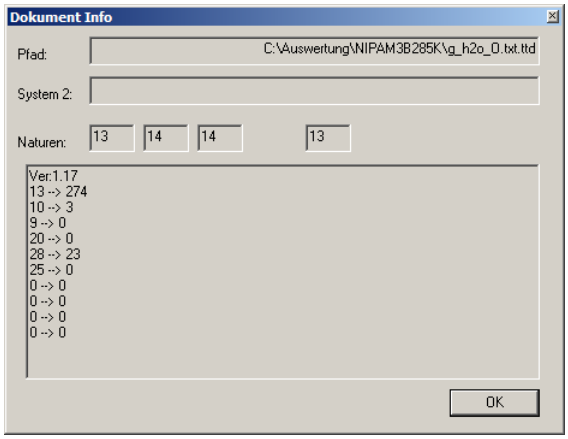
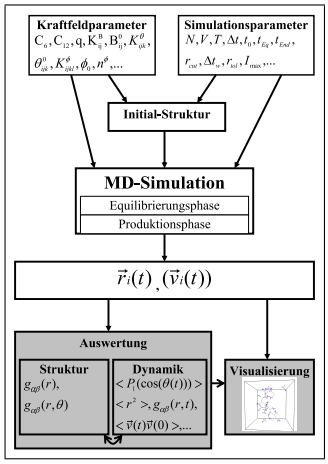


Abbildung 7.17: Der Dialog „Dokument Info“ listet die wichtigsten das System charakterisierende Parameter auf, dies sind:
1) Die Nummern der Teilchensorten (Naturen), die die Verteilungsfunktion bestimmen (in Gleichung (7.82) entsprechen sie A,B,C und D).
2) Die wichtigsten Teilchen und deren Häufigkeiten.
In dieser Darstellung: 13 = Wasser O, 10 = BIS CH_2, 9 = PAM N und 28 = NIPAM N.

- „Optionen“ → „Der Maus folgen“: Aktiviert bzw. deaktiviert die automatische Aktivierung des Fensters, das sich unter dem Mauszeiger befindet.
- „?“ → „Info über ttplot...“ (): Zeigt den Informations-Dialog zu TTPlot an.
- „?“ → „Info über Dokument...“ Alt-I (): In dem hier aufgerufenem Dialog (Abbildung 7.17) werden die wichtigsten das Dokument beschreibenden Parameter angezeigt. Dies sind der Name der Datendatei, die Nummern der für die Wechselwirkung wichtigen Teilchensorten, die das Koordinatensystem bilden, so wie eine Liste der wichtigsten Teilchensorten mit der zugehörigen Anzahl an Teilchen.

Kapitel 8

Auswertung von P-NIPAM



Wie bei allen Gelen werden auch beim P-NIPAM-Gel die makroskopischen Eigenschaften im wesentlichen durch den Einfluss des Polymers auf die Mobilität des Lösungsmittels und durch die Verteilung des Lösungsmittels im Polymer bestimmt. Die Beweglichkeit und Reorientierung einzelner Lösungsmittelmoleküle wird hierbei von der individuellen Struktur des Polymernetzwerks in der Umgebung des Moleküls beeinflusst. Die Auswertung wird, wie obige Abbildung zeigt (siehe auch Abbildung 1.1), in zwei Teilbereiche aufgespalten. In die Analyse der Struktur und in die Analyse der Dynamik, die sich zu einem Gesamtbild ergänzen.

Die Struktur von NIPAM und sein Einfluss auf die Verteilung des Wasser wird in Abschnitt 8.1 durch unterschiedliche Paarverteilungsfunktionen beschrieben. Sie beschreiben die Verteilung von Teilchen um ein oder eine Gruppe von Teilchen. Die Kombination dieser verschiedenen Paarkorrelationsfunktionen führt letztendlich zu einem Gesamtbild der Simulation, welche den strukturellen Aufbau von, sowohl des Polymers als auch des Lösungsmittels beschreibt.

Auf der anderen Seite werden im Abschnitt 8.2 die Ergebnisse der Dynamik der Teilchen vorgestellt. Die Dynamik der Seitengruppen des Polymers und der Lösungsmittelmoleküle wird dort über die mittlere quadratische Verschiebung $\left\langle |\vec{\mathbf{u}}_A(\tau) - \vec{\mathbf{u}}_A(0)|^2 \right\rangle$ und diverser Spektren beschrieben.

8.1 Die Struktur von NIPAM

Die zur Bestimmung der Struktur verwendeten Paarverteilungsfunktionen wurden in Abschnitt 7.2.2 eingeführt und gehören im Prinzip wiederum zur Gruppe der Korrelationsfunktionen aus Abschnitt 7.2.1. Die Analyse der Struktur erfolgt, ausgehend von einer groben Betrachtungsweise, hin zu einer immer detaillierter werdenden Spezialisierung der Paarverteilungsfunktionen.

In erster Näherung kann Wasser, unter Vernachlässigung der Wasserstoffa-

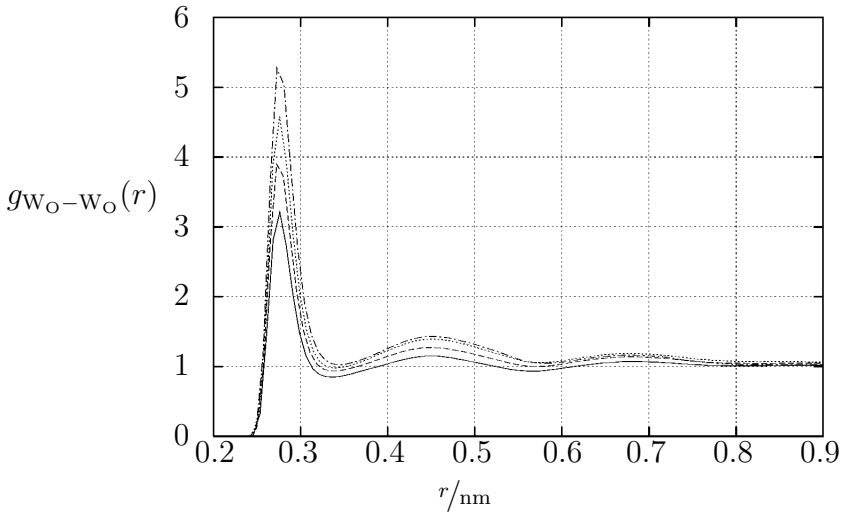


Abbildung 8.1: Paarkorrelation der Sauerstoffatome von Wasser in reinem Wasser (——), im System A (− − −), im System B (· · ·) und im System C (− · − · −) bei jeweils 300 K (aus [70]).

tome und der Ladungen, als ein strukturloses Teilchen betrachtet werden. Aus diesem Grund wird, insbesondere bei Abständen größer als 0,15 nm, die übliche radiale Paarverteilungsfunktion $g(r)$ aus Gleichung (7.47) für das Wasser im P-NIPAM Hydrogel zur Analyse Verwendung finden. Die Verteilung des Wassers in den unterschiedlichen Simulationssystemen (Tabelle 5.1) kann mit der einfachen radialen Paarverteilungsfunktionen $g_{\mathrm{W_O-W_O}}(r)$ der Sauerstoffatome der Wassermoleküle untereinander beschrieben werden. Ein Vergleich der Paarverteilungsfunktionen $g_{\mathrm{W_O-W_O}}(r)$ bei fester Temperatur, wie in Abbildung 8.1 zeigt eine starke Abhängigkeit der ersten ($r = 0,27$ nm) und zweiten ($r = 0,45$ nm) Hydratationsschale von der Polymerkonzentration. Die Teilchendichte nimmt dort mit steigendem Polymeranteil zu.

Aber nicht nur die Konzentration des Polymers, sondern auch die Temperatur des Systems bestimmt die Verteilung des Wassers. Die Paarverteilungsfunktionen $g_{\mathrm{W_O-W_O}}(r)$ des Systems C zeigen, wie in Abbildung 8.2 zu sehen, eine reziproke Abhängigkeit der ersten Hydratationsschale von der Temperatur, mit geringeren Schwankungen bei höheren Temperaturen.

Die einfache Paarverteilungsfunktion $g(r)$ ist nicht zur Beschreibung der Verteilung von Teilchen um eine Seitengruppe eines größeren Moleküls geeignet, da durch diese Moleküle eine radial homogene Verteilung verhindert wird. Derartige Verteilungen sind durch eine Winkel aufgelöste Paarverteilungsfunktion $g(r, \theta)$ besser zu beschreiben. Am Beispiel der Paarverteilungsfunktion $g_{\mathrm{HN-O_W}}(r, \theta)$ in Abbildung 8.3, die die Verteilung des Wasser Sauerstoffs um die Wasserstoff-Stickstoff-Bindung des NIPAM Monomers beschreibt, ist die Aufteilung in einen durch die Seitengruppe gesperrten Bereich und einen vom Wasser zugänglichen

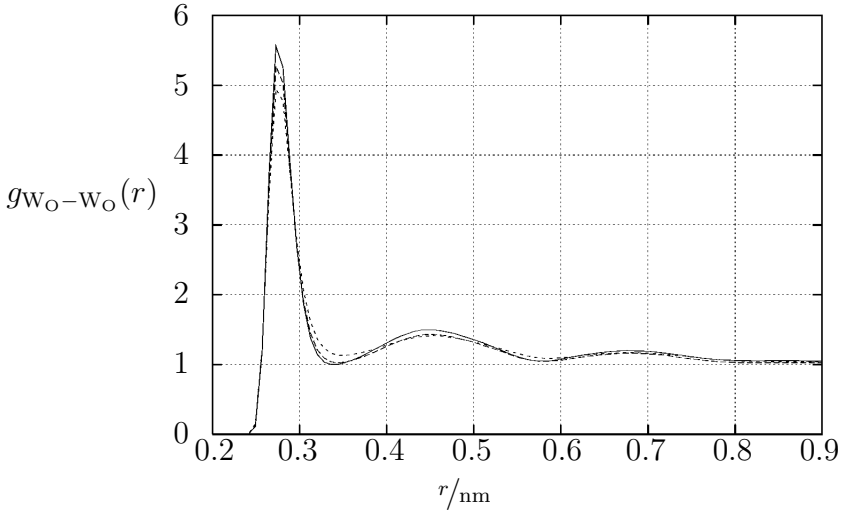


Abbildung 8.2: Paarkorrelation der Sauerstoffatome von Wasser im System C bei 285 K (——), 300 K (− − −) und 325 K (· · ·) (aus [70]).

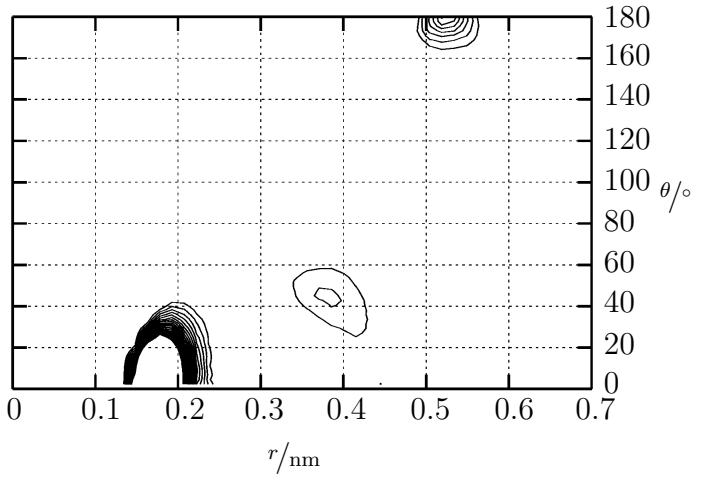


Abbildung 8.3: Winkelabhängige Paarverteilungsfunktion $g_{\mathrm{HN}-\mathrm{O_W}}(r, \theta)$ aus Gleichung (7.54) der Wasserstoff-Stickstoff-Bindung am NIPAM Monomer mit dem Sauerstoff vom Wasser in System C bei 300 K (aus [70]).

Bereich gut zu erkennen. Insbesondere bei kleinen Abständen zur Seitengruppe verdrängen die Teilchen der Seitengruppe alle anderen Atome in einem weiten Winkel. Bei dem in den Simulationen verwendeten Lösungsmittel Wasser handelt es sich um ein polares Lösungsmittel, weshalb für die Analyse der Wechselwirkungen zwischen Lösungsmittel und Polymer insbesondere die Teile des Gels interessant sind, die eine Partialladung tragen. So zeigt die in Abbildung 8.3 dar-

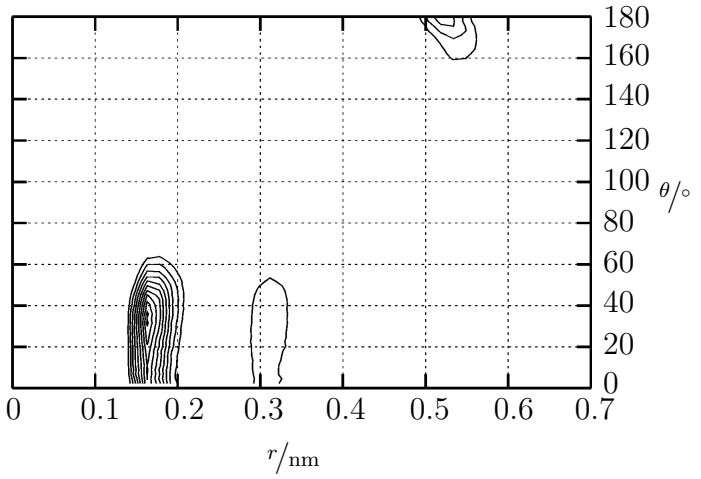


Abbildung 8.4: Winkelabhängige Paarverteilungsfunktion $g_{\mathrm{OC-H_W}}(r, \theta)$ der Sauerstoff-Kohlenstoff-Bindung am NIPAM Monomer mit dem Wasserstoff vom Wasser in System C bei 300 K (aus [70]).

gestellte Paarverteilungsfunktion $g_{\mathrm{HN-O_W}}(r, \theta)$, um den aus dem Stickstoff und dem Wasserstoff gebildeten Dipol des Polymers, charakteristische Schalensegmente höherer Dichte. Das innerste Schalensegment befindet sich in einem Abstand r von 0,13 nm bis 0,24 nm zwischen dem Stickstoff des NIPAM und dem Sauerstoff des Wassers, sein Winkelsegment zwischen den Teilchen liegt in einem Winkel θ bis zu 30°. Diese Abgrenzung charakterisiert nach Luzar und Chandler [65] (siehe Gleichung (7.42)) gerade eine Wasserstoffbrückenbindung. Eine zweites Schalensegment existiert in einem Abstand zwischen 0,32 nm und 0,45 nm und einem Winkelbereich bis zu 60°, als Resultat der Wasserstoffbrückenbindungen des Wassers aus dem inneren Schalensegment mit weiterem Wasser.

In entgegengesetzter Richtung zur H-N-Bindung ist auch noch die Bindung des Wassers mit dem Sauerstoff des Polymers zu erkennen, diese lässt sich mit Hilfe von Abbildung 8.4 ebenfalls als Wasserstoffbrückenbindung identifizieren. Das innere Schalensegment in Abbildung 8.4 erfüllt die für Wasserstoffbrückenbindungen charakteristischen Randbedingungen, da es in einem Abstand zwischen 0,14 nm und 0,22 nm und einem Winkel kleiner als 70° zum Sauerstoff des Polymers liegt. Dieses Schalensegment ist nicht so sehr ausgeprägt, wie das am Wasserstoff des Stickstoffs, es hat eine wesentlich geringere Dichte, bildet dennoch ein zweites Segment aus.

Im Prinzip besteht der Quervernetzer im Polymer aus den selben Bestandteilen wie ein NIPAM-Monomer der Polymerkette, jedoch zeigen beide ein leicht unterschiedliches Verhalten. Zwar zeigt Abbildung 8.5 für die Bindung am BIS die gleiche Verteilung der Schalensegmente, aber die Schalensegmente am BIS haben eine sehr viel geringere Dichte, als die aus Abbildung 8.3. Jedoch stimmen,

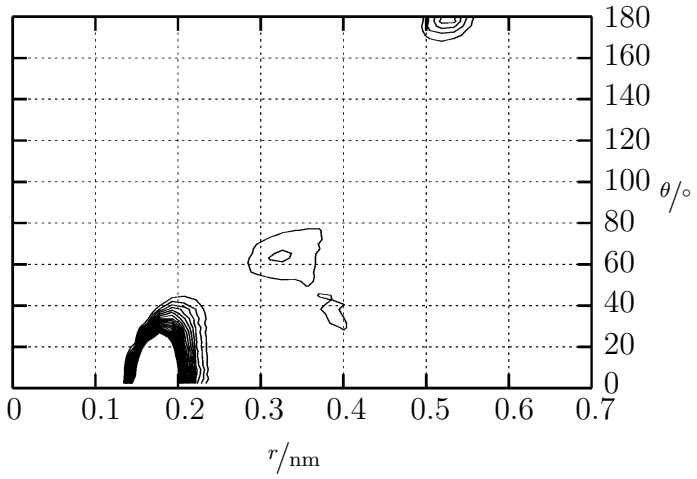


Abbildung 8.5: Winkelabhängige Paarverteilungsfunktion $g_{\mathrm{HN-O_W}}(r, \theta)$ der Wasserstoff-Stickstoff-Bindung am BIS Monomer mit dem Sauerstoff vom Wasser in System C bei 300 K (aus [70]).

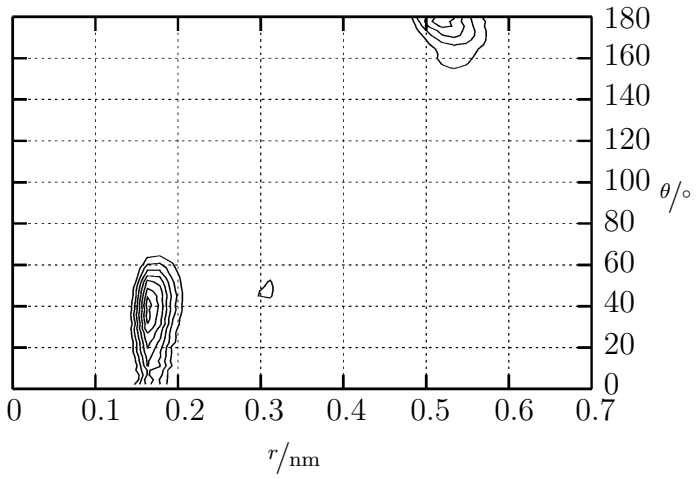


Abbildung 8.6: Winkelabhängige Paarverteilungsfunktion $g_{\mathrm{OC-H_W}}(r, \theta)$ der Sauerstoff-Kohlenstoff-Bindung am BIS Monomer mit dem Wasserstoff vom Wasser in System C bei 300 K (aus [70]).

wie Abbildung 8.6 zeigt, das innerste Schalensegment am Sauerstoff des BIS nicht nur in Abstand und Winkelverteilung mit dem am NIPAM überein, sondern auch in ihrer Intensität. Lediglich das zweite Schalensegment ist gegenüber dem am NIPAM wesentlich schwächer ausgeprägt.

Mit den winkelabhängigen Paarverteilungsfunktionen lässt sich die Verteilung

des Wassers um das Polymer bereits recht genau beschreiben. Eine auf das visuelle Verständnis ausgelegte Variante der Paarverteilungsfunktionen ist die räumliche Paarverteilungsfunktion $g(\vec{\mathbf{r}})$, sie ist am detailliertesten, benötigt jedoch für eine genaue Verteilung eine möglichst lange Trajektorie. Abbildung 8.7a zeigt einen Schnitt durch die räumliche Paarverteilungsfunktion des Wassers am Wasserstoff des NIPAM Stickstoffs. Die Bestandteile des Wassers werden hierbei durch eine Überlagerung der Verteilungen des Sauerstoffs (rot) und des Wasserstoffs (blau) dargestellt. Der gezeigte Schnitt erfolgte durch die H-N-Bindung und dem Sauerstoff der Amid-Seitengruppe. Aufgrund der Überlagerung ist es möglich, die Orientierung des Wassers an der Seitengruppe einfach anhand der Verteilungen von Sauerstoff (rot) und Wasserstoff (blau) zu identifizieren. Der Dipolvektor des Wassers zeigt, wie zu erwarten ist, in die gleiche Richtung wie der Vektor der N-H-Bindung. Sucht man nach Regionen höchster partieller Dichte, so sind diese in einem Gebiet 0,13 bis 0,24 nm vom Wasserstoff des Amid entfernt, wobei dieses Gebiet in einem Winkel von bis zu 30° zur H-N-Bindung liegt. Nach Luzar und Chandler [65] befindet es sich genau in dem Bereich, in dem Wasserstoffbrückenbindungen anzutreffen sind.

Zur Repräsentation der Amid-Seitengruppe werden sowohl im Polymer (NIPAM) als auch im Quervernetzer (BIS) die selben Atomtypen benutzt, d. h. die Wechselwirkungen und deren Parameter sind in beiden Fällen idealisiert. Vergleicht man jedoch die räumlichen Paarverteilungsfunktionen $g_{\mathrm{NHO-O_W}}(\vec{\mathbf{r}})$ und $g_{NHO-H_W}(\vec{\mathbf{r}})$ zwischen NIPAM (Abbildung 8.7a) und BIS (Abbildung 8.7b), so zeigt Erstere gegenüber der Zweiten eine höhere lokale Dichte. Im BIS ist, im Gegensatz zur symmetrischen Verteilung am Amid des NIPAM, das Gebiet der Wasserstoffbrückenbindungen mit dem Wasser nicht symmetrisch vor dem Wasserstoff verteilt, auf der Seite des Sauerstoffatoms ist das Gebiet kleiner als auf der gegenüberliegenden Seite. Zurückgeführt werden kann das auf den, im Vergleich mit dem NIPAM, geringeren Abstand zwischen Stickstoffatom und benachbarten Sauerstoffatom im BIS, der sich aus dessen zweiter Amid-Gruppe ergibt.

Um die Bindungen mit dem Sauerstoff des NIPAM zu visualisieren, ist eine auf den Sauerstoff des Polymers zentrierte Darstellung, wie in Abbildung 8.8a geeigneter, als die oben beschriebene aus Abbildung 8.7a. Wieder ist die Ausrichtung der Dipole des Wassers leicht zu erkennen und die Lage des Bereichs der höchsten Dichte liegt ebenso in den von Luzar und Chandler [65] vorgegebenen Grenzen einer Wasserstoffbrückenbindung. Vergleicht man beide Wasserstoffbrückenbindungen des NIPAM, so ist die an der N-H-Bindung aus Abbildung 8.7a von höherer Intensität, aber auf einen kleineren Winkelbereich begrenzt.

Im Gegensatz zum unsymmetrischen Aufbau des Gebiets der Wasserstoffbrückenbindungen am Wasserstoff des BIS (Abbildung 8.7b), gibt es keine entsprechende Verschiebung des Gebietes am Sauerstoff des BIS (Abbildung 8.8b). Es ist lediglich, verglichen mit der relativen Dichte am NIPAM (Abbildung 8.8a), eine deutlich geringere relative Dichte des Wassers innerhalb des Gebietes des Wasserstoffbrückenbindungen zu finden.

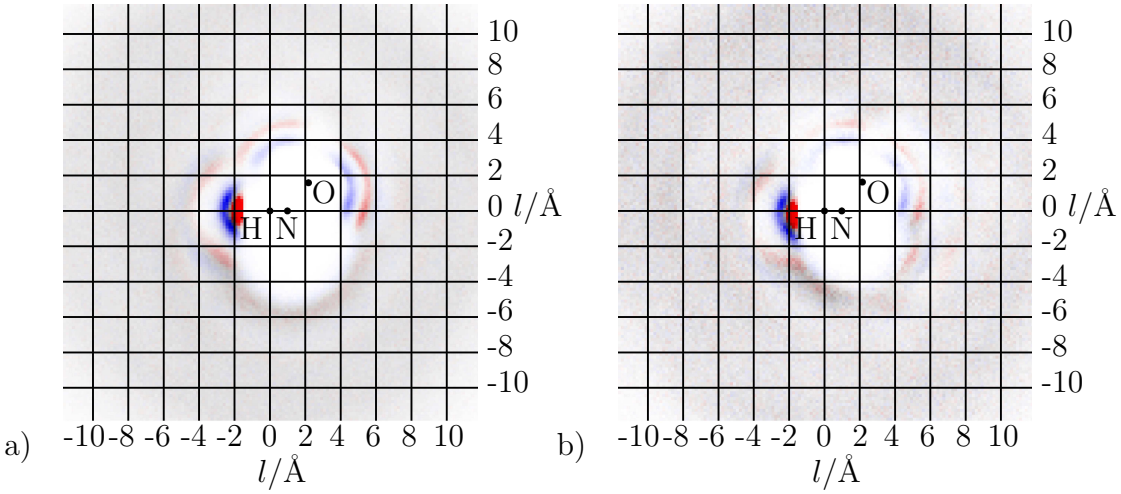


Abbildung 8.7: Schnittebene der räumlichen Paarverteilungsfunktion des NIPAM (a) und des BIS (b) Wasserstoffs am Stickstoff mit dem Sauerstoff ($g_{\mathrm{NHO-O_W}}(\vec{\mathbf{r}})$, rot) und dem Wasserstoff ($g_{\mathrm{NHO-H_W}}(\vec{\mathbf{r}})$, blau) des Wassers in der durch die H-N Bindung und dem O der Amid-Seitengruppe aufgespannten Ebene des Systems C bei 300 K (aus [70]).

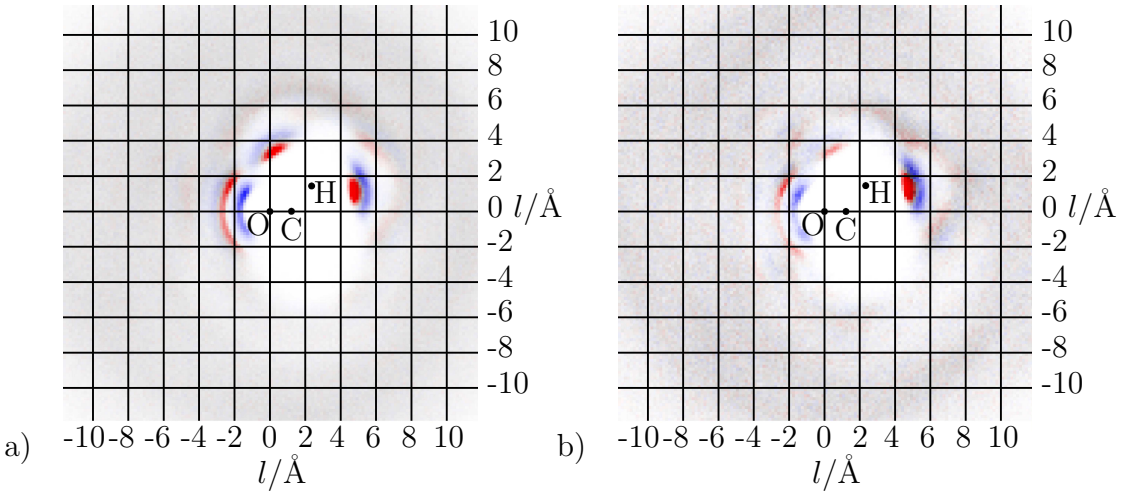


Abbildung 8.8: Schnittebene der räumlichen Paarverteilungsfunktion des NIPAM (a) und des BIS (b) Sauerstoffs mit dem Sauerstoff ($g_{\mathrm{COH-O_W}}(\vec{\mathbf{r}})$, rot) und dem Wasserstoff ($g_{\mathrm{COH-H_W}}(\vec{\mathbf{r}})$, blau) des Wassers in der durch die O-C Bindung und dem H der Amid-Seitengruppe aufgespannten Ebene des Systems C bei 300 K (aus [70]).

Die Darstellungen der bisherigen Paarverteilungen wurden von der Warte des Polymers aus betrachtet. Abbildung 8.9 zeigt die Verteilung des Sauerstoffs vom NIPAM und vom Wasser ausgehend von der Lage eines Wassermoleküls. Wie zu sehen ist, gibt es einen Bereich (schwarz) in dem sowohl der Sauerstoff des NIPAM, als auch der Sauerstoff der anderen Wassermoleküle die größte Auf-

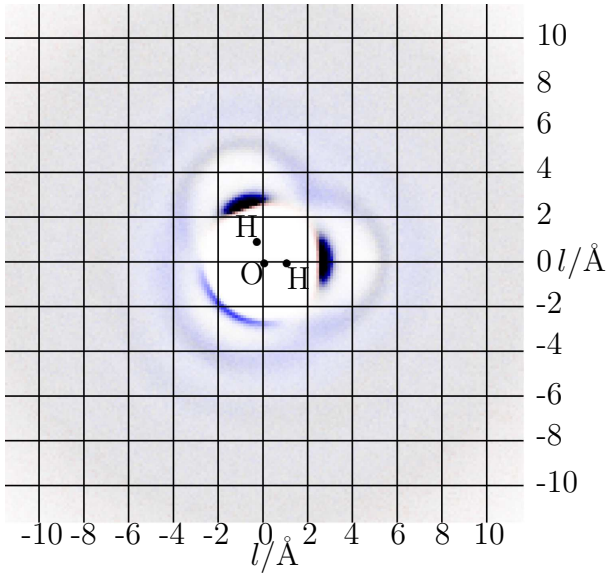


Abbildung 8.9: Schnittebene der räumlichen Paarverteilungsfunktion $g_{\mathrm{HOH-O}}(\vec{\mathbf{r}})$ der Wassermoleküle mit den Sauerstoffatomen des NIPAM (rot) und den Sauerstoffatomen der anderen Wassermoleküle (blau) in der durch das Wasser festgelegten Ebene im Systems C bei 285 K.

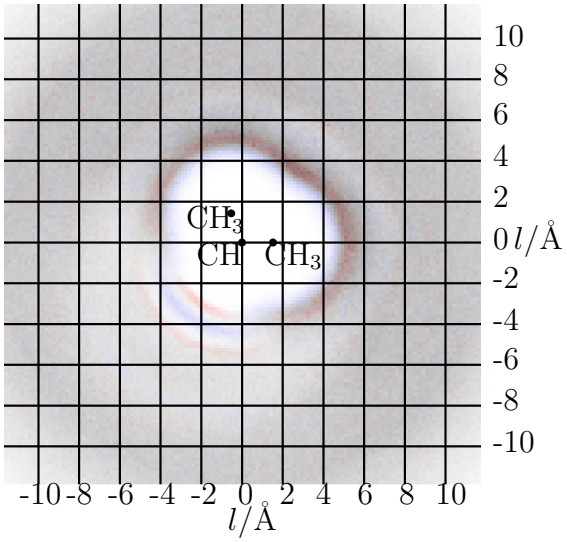


Abbildung 8.10: Schnittebene der räumlichen Paarverteilungsfunktion des Methylteilchens der Isopropyl-Gruppe des NIPAM mit dem Sauerstoff ($g_{(\mathrm{CH_3})(\mathrm{CH})(\mathrm{CH_3})-\mathrm{O_W}}(\vec{\mathbf{r}})$, rot) und dem Wasserstoff ($g_{(\mathrm{CH_3})(\mathrm{CH})(\mathrm{CH_3})-\mathrm{H_W}}(\vec{\mathbf{r}})$, blau) des Wassers in der durch die $\mathrm{CH_3}-\mathrm{CH}-\mathrm{CH_3}$ Bindungen aufgespannten Ebene des Systems C bei 300 K (aus [70]).

enthaltswahrscheinlichkeit besitzen. Zu erkennen ist aber auch, dass der Bereich hoher Aufenthaltswahrscheinlichkeit (schwarz, blau) beim Sauerstoff des Wassers

weit ausgedehnter ist, als an den Atomen der Seitengruppe des NIPAM (schwarz, rot). Der Abstand zwischen dem Wasserstoff und den nächsten Sauerstoffatomen beim NIPAM ist etwas geringer und diffuser als beim Wasser. Gegenüber den beiden Hauptaufenthaltsgebieten des Wassers befindet sich ein weiteres Gebiet mit hoher Aufenthaltswahrscheinlichkeit. Dieses wird durch die Sauerstoffatome der Wassermoleküle verursacht, dessen Wasserstoffatome mit dem Sauerstoff des betrachteten Wassermoleküls eine Wasserstoffbrückenbindung eingehen.

Die Verteilung des Wassers an einer unpolaren Seitengruppe des Polymers sieht völlig anders aus. Abbildung 8.10 zeigt die Verteilung des Wassers um die Isopropyl-Gruppe des NIPAM, mit einem Methylteilchen im Zentrum. Zwischen der Isopropyl-Gruppe und dem Wasser bestehen keine elektrostatischen Wechselwirkungen. Die Wechselwirkungen werden lediglich durch ein Lennard-Jones-Potential bestimmt, welches ein Ausschlussbereich um die Isopropyl-Gruppe erzeugt, in dem sich kein Lösungsmittelmolekül aufhalten kann. Der Ausschlussbereich wird an der Grenze zum Lösungsmittel von einer 80 pm dicken Schale umgeben, in der die Dichte auf das 2 bis 3,5-fache der mittleren Dichte des Lösungsmittels ansteigt. Die Schale wird jedoch im Bereich der Bindung zwischen dem Stickstoff und dem CH-Teilchen, aufgrund der Wechselwirkung mit den restlichen Teilchen des Monomers, abgeschwächt bzw. unterbrochen. Aufgrund der fehlenden Lennard-Jones-Wechselwirkung der Wasserstoffatome des Wassers gibt es keine direkten Wechselwirkungen mit der Isopropyl-Gruppe. Die Abstoßung zwischen der Isopropyl-Gruppe und dem Wasserstoff erfolgt indirekt durch Abstoßung des an den Wasserstoff gebundenen Sauerstoffs. Der innere Rand der Schale der Wasserstoffatome ist deshalb um 30 pm näher an der Isopropyl-Gruppe als die Schale der Sauerstoffatome des Wassers. Dieser, durch ein fehlendes Lennard-Jones-Potential mit dem Wasserstoff, entstandene Fehler ist allerdings nicht auf das hier verwendete SPC/E [74] Modell beschränkt, auch die Wasserstoffatome im TIP4P [75] oder im ST2-Modell [76] besitzen kein eigenes Lennard-Jones-Potential.

Die räumliche Anordnung der Wassermoleküle im Wasser wird im wesentlichen durch die elektrostatischen Wechselwirkungen bestimmt, sie wird ergänzt durch ein, den Abstand der Sauerstoffatome beeinflussendes Lennard-Jones-Potential. Wie in Abbildung 8.11a zu sehen ist, bildet sich nicht nur an den beiden Wasserstoffatomen ein Gebiet mit hoher Sauerstoffkonzentration aus, sondern über weiter entfernte Wassermoleküle bilden sich zwei weitere Gebiete senkrecht zur Ebene des Wassermoleküls aus. Gebildet werden die beiden Gebiete von den Sauerstoffatomen der Wassermoleküle, deren Wasserstoffatome mit dem Sauerstoffatomen der betrachteten Wassermoleküle eine Wasserstoffbrückenbindung eingehen. Die Wasserstoffatome dieser Moleküle sind im linken Teil der Abbildung 8.11b zu sehen. Hervorzuheben ist außerdem, dass sich die Wasserstoffatome an den „freien Elektronenpaaren“ ausrichten, ohne das diese, wie etwa im TIP5P [77] oder im ST2-Modell [76], explizit im Wassermodell modelliert sind. Die Begrenzungen der Gebiete ergeben sich ausschließlich aus den abstoßenden

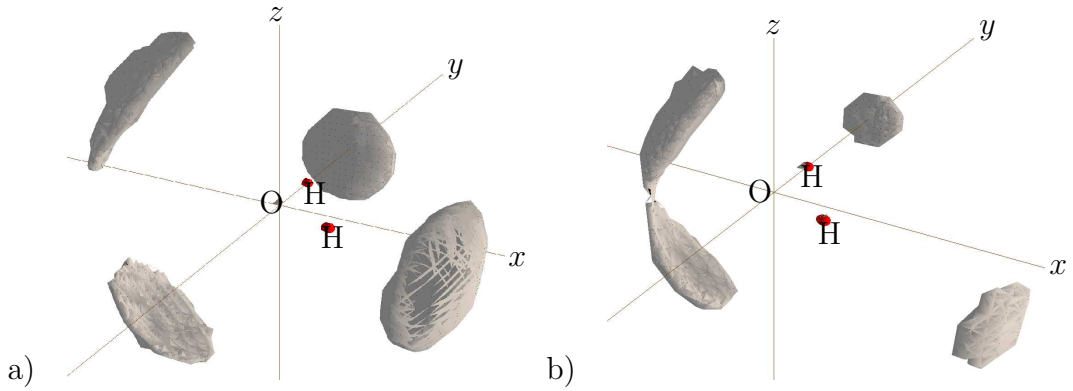


Abbildung 8.11: Darstellung der räumlichen Verteilung $g_{\mathrm{HOH-O}}(\vec{\mathbf{r}})$ des Sauerstoffs (a) und $g_{\mathrm{HOH-H}}(\vec{\mathbf{r}})$ des Wasserstoffs (b) des Wassers um Wasser in der vom Wassermolekül definierten Ebene, durch die von der Verteilung festgelegte Grenzfläche der zehnfachen Standarddichte im System C bei 300 K.

elektrostatischen Wechselwirkungen mit den Wasserstoffatomen des betrachteten Moleküls.

8.2 Analyse der Dynamik

Die Dynamik wird im folgenden anhand der mittleren quadratischen Verschiebung $\left\langle \left|\vec{\mathbf{u}}_A(\tau) - \vec{\mathbf{u}}_A(0)\right|^2 \right\rangle$ und diverser Spektren beschrieben, die durch weitere in [67, 68] veröffentlichte Methoden, wie die Reorientierungsautokorrelation $\langle P_1(\cos(\theta(s)))\rangle$ oder die zeitaufgelöste Paarkorrelationsfunktion $g(r,t)$ ergänzt werden.

Die mittlere quadratische Verschiebung $\left\langle \left|\vec{\mathbf{u}}_A(\tau) - \vec{\mathbf{u}}_A(0)\right|^2 \right\rangle$ der Wassermoleküle stellt einen Indikator für die Steifigkeit des Hydrogels dar. Je weniger die Wassermoleküle in ihrer Beweglichkeit gehindert werden, desto flexibler ist das Gel. In Abbildung 8.12 ist die mittlere quadratische Verschiebung des Sauerstoffatoms der Wassermoleküle, dass in erster Näherung auch dem Schwerpunkt des Wassermoleküls entspricht, in Abhängigkeit von der Zeit t für die drei hier betrachteten Systeme bei 300 K aufgetragen.

Aus der Steigung der mittleren quadratischen Verschiebung lässt sich mit Hilfe von Gleichung (7.27) der Diffusionskoeffizienten D_T bestimmen. Zu beobachten ist, dass dieser mit steigender Polymerkonzentration abnimmt. Für reines Wasser ist der Diffusionskoeffizient D_T am größten, dem entsprechend ist sie am anderen Ende, für das System C mit der höchsten Polymerkonzentration, am geringsten. Die sich aus der mittleren quadratischen Verschiebung ergebenen Diffusionskoeffizienten D_T sind in Tabelle 8.1 für alle vier Systeme aus Abbildung 8.12 bei 300 K aufgelistet. Ergänzt werden die Diffusionskoeffizienten D_T der Tabelle um die Koeffizienten der Polymersysteme A, B und C bei 285 K und bei 325 K. Deut-

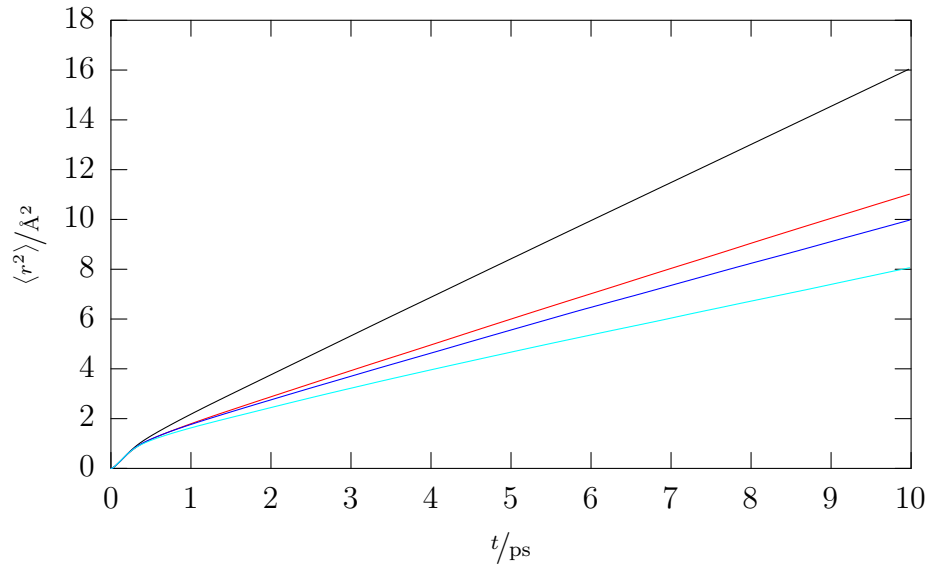


Abbildung 8.12: Mittlere quadratische Verschiebung der Wassermoleküle in reinem Wasser(—) und im P-NIPAM Hydrogel der Systeme A (—), B (—) und C (—) bei jeweils 300 K.

System	T/K	$D_T/10^{-5}\text{cm}^2\text{s}^{-1}$
A	285	1,28
B		0,90
C		0,83
A	300	1,64
B		1,47
C		1,14
Wasser		2,55
A	325	2,72
B		2,15
C		1,83
Tanaka et al. [78]	297	1,72
	306	2,25

Tabelle 8.1: Translationsdiffusion D_T von Wasser in den Systemen A, B und C (aus [70]).

lich erkennt man hier deren starke Temperaturabhängigkeit, die mit zunehmender Polymerkonzentration weiter zunimmt. Zum Vergleich mit experimentellen Daten sind in der Tabelle außerdem die Messungen von Tanaka et al. [78] aufgeführt, die eine gut Übereinstimmung mit den aus der Simulation ermittelten Werten zeigen.

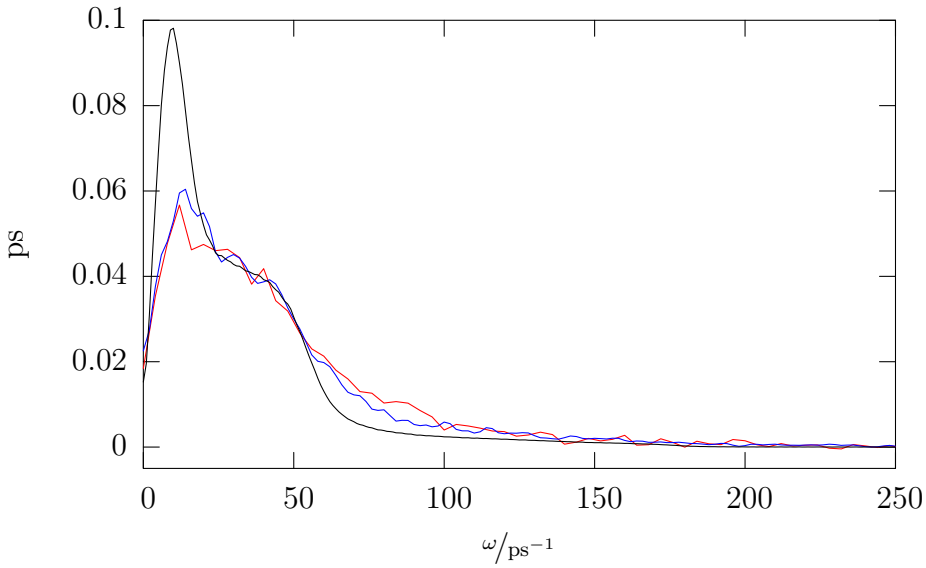


Abbildung 8.13: Spektraldichte von ungebundenem Wasser (—) und von Wasser in einem Umkreis von 3 Å um die Sauerstoffatome der BIS-Monomere (—) und der NIPAM-Monomere (—) in System B bei 285 K.

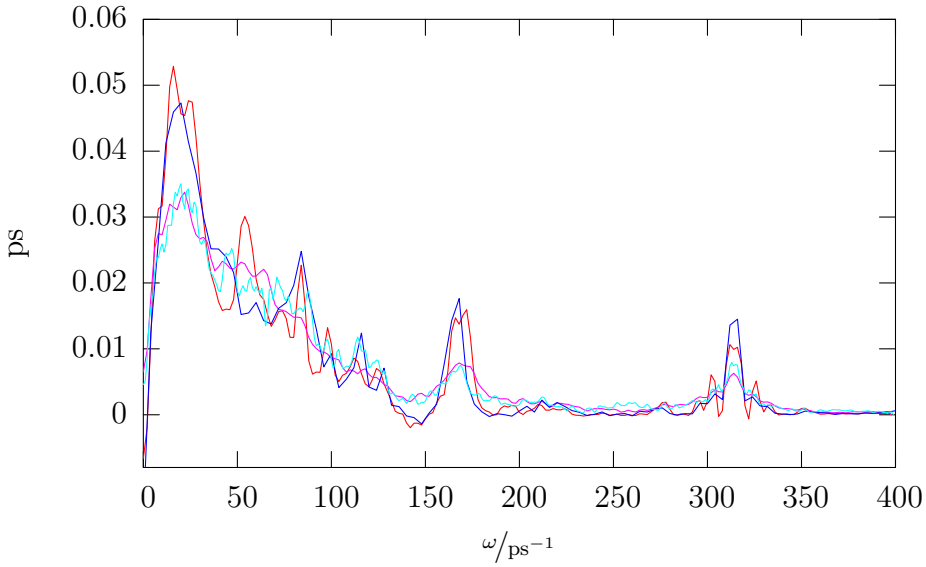


Abbildung 8.14: Mittlere Spektraldichte der Sauerstoffatome am Monomer (NIPAM:—, BIS:—) und bei Anwesenheit von Wasser in einem Radius von 3 Å (NIPAM:—, BIS:—).

Die Spektraldichte berechnet sich, wie in Abschnitt 7.2.5 gezeigt, aus der Fouriertransformation der Geschwindigkeits-Autokorrelationsfunktion $K_{\vec{\mathbf{v}}}(t)$. Mit ihr können die Frequenzen der verschiedenen Schwingungen eines Teilchens bzw.

einer Gruppe von Teilchen in einem Diagramm dargestellt und analysiert werden.

Das Lösungsmittel Wasser besitzt in dem in Abbildung 8.13 dargestellten Bereich zwei charakteristische Frequenzen. Am ausgeprägtesten ist das erste Maximum, verursacht von der O-O-O Biegeschwingung, zwischen den Wassermolekülen bei 10 ps^{-1}. An das Maximum schließt sich eine breite Schulter im Bereich um 32 ps^{-1} an, die der O-O Streckschwingung zugeordnet werden kann. Vergleicht man die Spektraldichte des Wassers in reinen Wasserregionen mit der Spektraldichte von Wasser in der Nähe (Abstand kleiner 3 Å) der Sauerstoffatome des Polymers, mit denen, wie bereits im vorherigen Abschnitt gezeigt, das Wasser Wasserstoffbrückenbindungen eingeht, so zeigt sich eine Erhöhung der Frequenz der Biegeschwingung auf 12 ps^{-1} am Sauerstoff des BIS und auf 14 ps^{-1} am Sauerstoff des NIPAM. Das Polymer beeinflusst ebenfalls die O-O Streckschwingung, was zu einem flacheren Abfall am Ende der von der O-O Streckschwingung erzeugten Schulter führt. Die O-O Streckschwingung wird zu höheren Frequenzen hin verschoben.

Genauso wie das Polymer das Wasser beeinflusst, beeinflusst auch das Wasser das Polymer und seine Seitengruppen in ihrem dynamischen Verhalten. Abbildung 8.14 zeigt die mittlere Spektraldichte der Sauerstoffatome des Polymers und deren Spektraldichte bei Anwesenheit von Wassermolekülen in einem Radius von 3 Å um das betrachtete Sauerstoffatom. An der Verteilung der Frequenzen fällt auf, dass diese wesentlich unstetiger ist als die der Wassermoleküle. Wie beim Wasser verschiebt sich, beim direkten Vergleich des mittleren Spektrums mit einem Spektrum mit Wasser in einer Schale vom Radius 3 Å, das erste Maximum und auch das Maximum bei 168 ps^{-1} zu höheren Frequenzen. Die Schulter nach dem ersten Maximum, die wegen der vielen lokalen Maxima auf der Schulter leicht versteckt ist, ist wesentlich flacher als bei den Verteilungen für Wasser aus Abbildung 8.13. Gegenüber der Frequenzverteilung des Wassers gibt es zwei zusätzliche markante Maxima, das eben erwähnte Maximum bei 168 ps^{-1} und das Maximum bei 315 ps^{-1}, das im Gegensatz zu den beiden Ersten keine Frequenzverschiebung mehr aufweist. Beide Maxima besitzen aber, wie das erste Maximum, verbreiterte Schultern bei der Berücksichtigung der Schale mit Wasser.

Kapitel 9

Aufbau von StructC und dynC

Bei der Implementation von „StructC“ und „dynC“ wurde die in C++ vorhandene Möglichkeit der objektorientierten Programmierung verwendet, in dieser werden die einzelnen Bestandteile des Simulationssystems auf Klassen aufgeteilt, die zusammen ein hierarchisches System bilden. Die in den beiden Programmen verwendeten Klassen lassen sich bezüglich ihrer Verwendung in drei Kategorien einteilen. Zum einen gibt die Klassen, die nur zur Strukturgenerierung benötigt werden, dann gibt es die Klassen, die nur in der Simulation von Interesse sind und schließlich die Klassen, die in beiden Programmen Verwendung finden. Von Letzteren werden bei der Strukturgenerierung bzw. Simulation meistens Nachfahren erzeugt, die die ursprünglichen Klassen um zusätzliche Attribute (Variablen) und Methoden (Funktionen oder Prozeduren) ergänzen, die diese dann in die Lage versetzen spezielle Aufgaben zu erfüllen.

Der hierarchische Aufbau der Klassen, so wie ihre enge Verzahnung und Zugehörigkeit zu den beiden Programmen, werden in den Abhängigkeits-Diagrammen der Abbildungen 9.1 bis 9.3 deutlich. Die Aufteilung, der den Kern der Programme bildenden Klassen, in `CAtom`, `CMolecule` und `CMoleculeList` folgt hierbei intuitiv dem Vorbild der Natur. Diese legt eine natürliche Abgrenzung zwischen den Atomen, den Molekülen und dem Gesamtsystem fest. Dem gegenüber werden Hilfsklassen, wie `Complex` oder `CFileIO`, allein durch ihre Aufgabe definiert.

Bei der Aufteilung der Klassen, aber insbesondere bei der Planung und Erstellung ihrer Methoden, wurde das Prinzip des „Teile und Herrsche“ angewandt, das den hierarchischen Aufbau mit prägte. Bezogen auf die Methoden bedeutet dieses Prinzip, deren Aufgaben solange in einfachere Aufgaben aufzuteilen und an Methoden von Subklassen weiter zu geben, bis diese in den abgeleiteten Methoden elementar sind.

Die Benennung der Attribute bzw. Variablen in den Programmen lehnt sich an die von Charles Simonyi (vgl. [79, 80]) eingeführte „Ungarischen Notation“. In dieser Notation wird der Name einer Variablen in einen Präfix- und einen Postfix-Anteil aufgespalten. Der Präfix-Anteil besteht aus einer Zeichenfolge, die

Präfix	Beschreibung
m_	Attribut einer Klasse (member)
f	Fließkommazahl (float)
d	Fließkommazahl (double)
n	Ganzzahlig (BYTE, WORD, DWORD, short, int, long)
b	Wahrheitswert (Boolean)
c	Zähler (counter)
p	Zeiger (Pointer)
sz	Zeichenfolge mit Null-Wert abgeschlossen
a	Feld (array)
m	Feld gekapselt durch `CMem<>`
mi	Iterator `CMemIter<>` auf ein Feld der Klasse `CMem<>)`
cx	Komplexe Zahl der Klasse `Complex`
v	3D-Vektor der Klasse `CVec3`
g	globale Variable
dbg_	Variable, nur im DEBUG-Modus benutzt

Tabelle 9.1: Bedeutung des Präfix eines Variablennamens in der Ungarischen Notation.

die Art einer Variablen und deren Datentyp charakterisiert. Der Postfix-Anteil ergänzt den Variablennamen durch einen den Zweck beschreibenden Ausdruck, wobei einzelne Worte des Ausdrucks durch Großbuchstaben gegeneinander abgegrenzt werden. Die verwendeten Präfixe sind in Tabelle 9.1 aufgeführt und können miteinander kombiniert werden. Am einfachsten lässt sich die Notation durch die beiden folgenden Beispiele erklären. Bei der Variablen `m_mpAtom` handelt es sich um ein Attribut (Präfix:`m_`) einer Klasse. Dieses Attribut enthält ein durch die Klasse `CMem<CAtom *>` gekapseltes Feld (Präfix:`m`) von Zeigern (Präfix:`p`) auf Atome (Postfix:`Atom`). Die Variable mit dem Namen `cnAtom` besitzt einen ganzzahligen Datentyp (Präfix:`n`) und wird verwendet, um Atome (Postfix:`Atom`) zu zählen (Präfix:`c`).

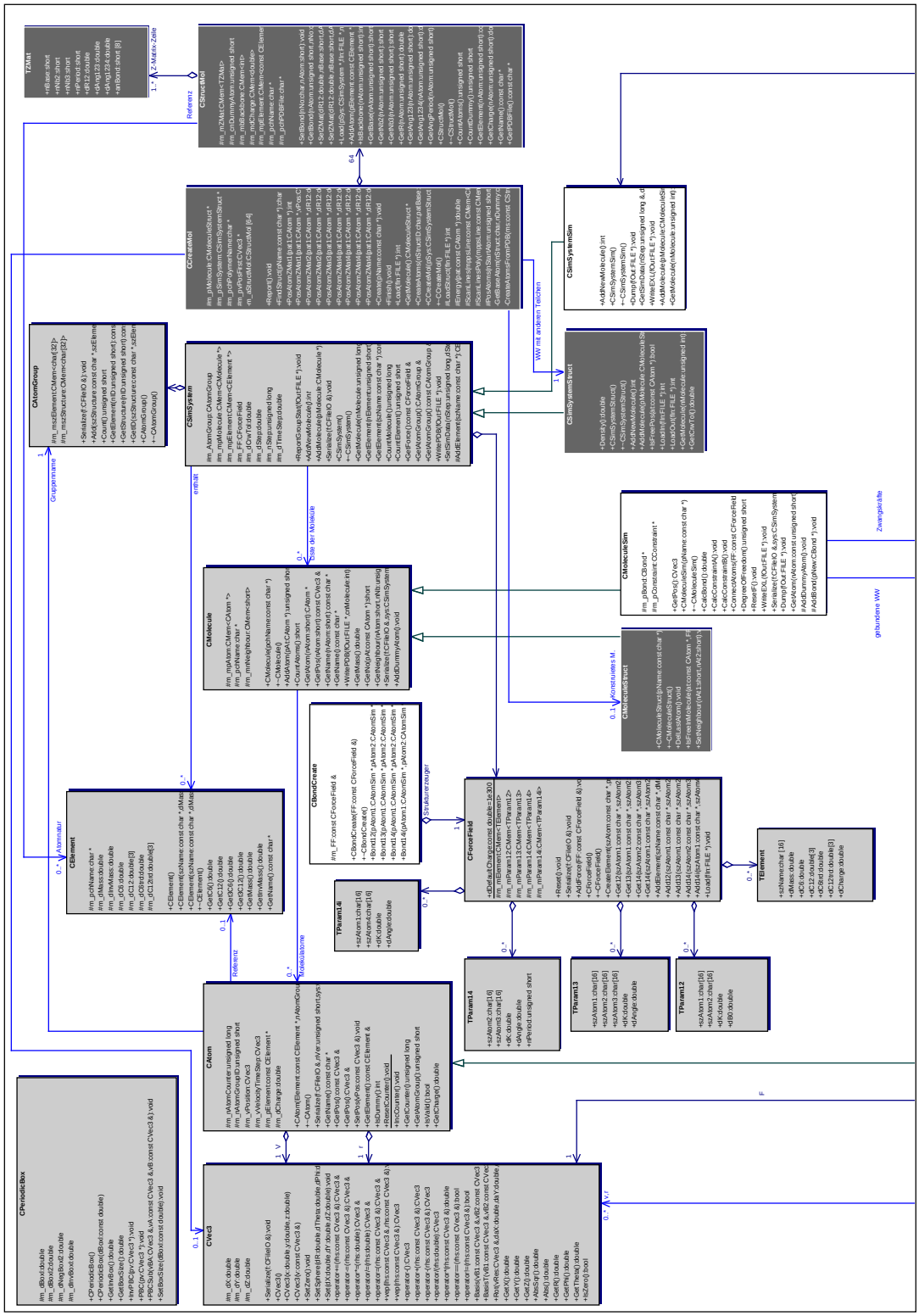


Abbildung 9.1: Teil 1 der Klassenstruktur von StructC (schwarzer Hintergrund) und dynC (weißer Hintergrund) (gemeinsame Klassen, grau).

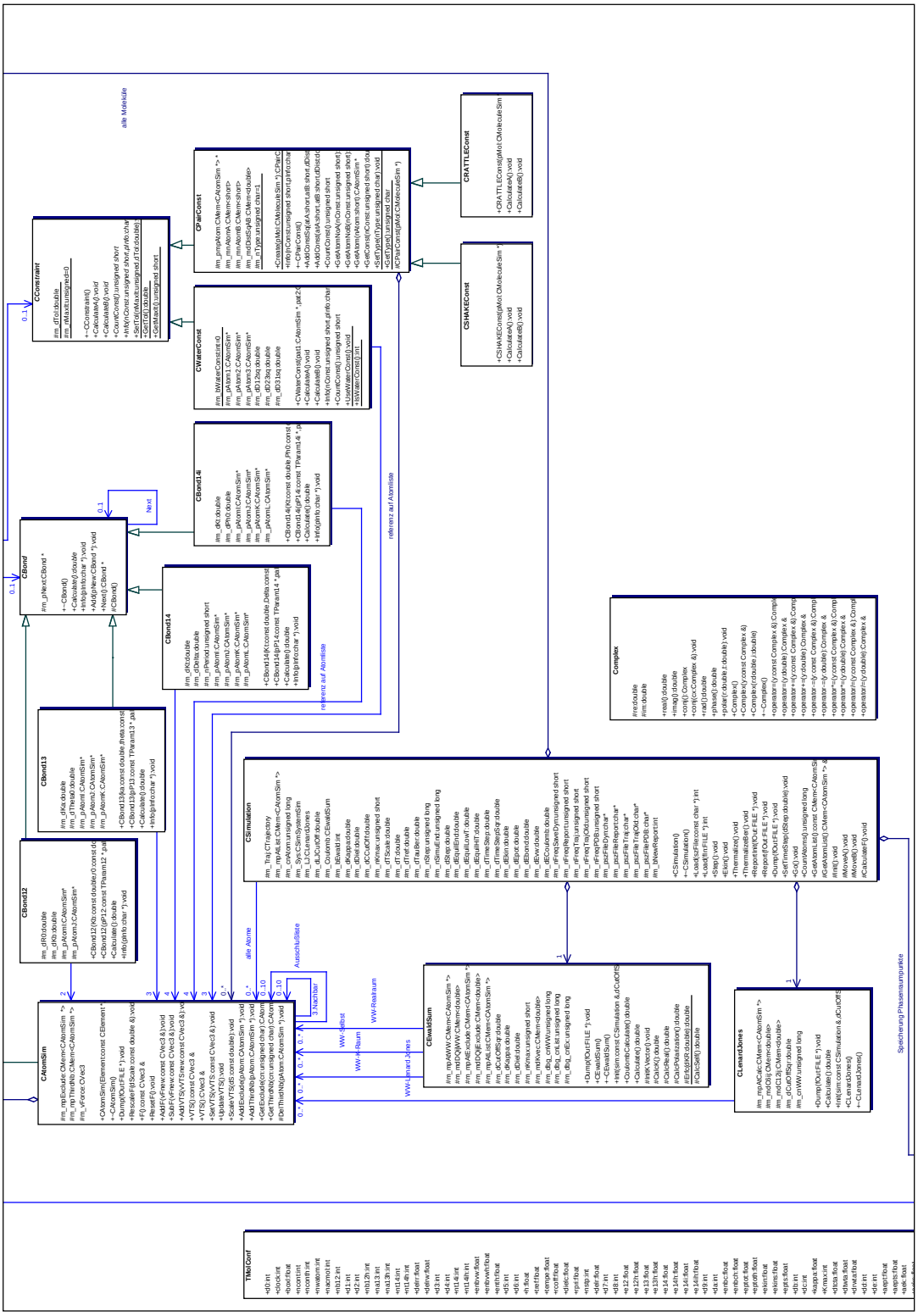


Abbildung 9.2: Teil 2 der Klassenstruktur von StructC (schwarzer Hintergrund) und dynC (weißer Hintergrund).

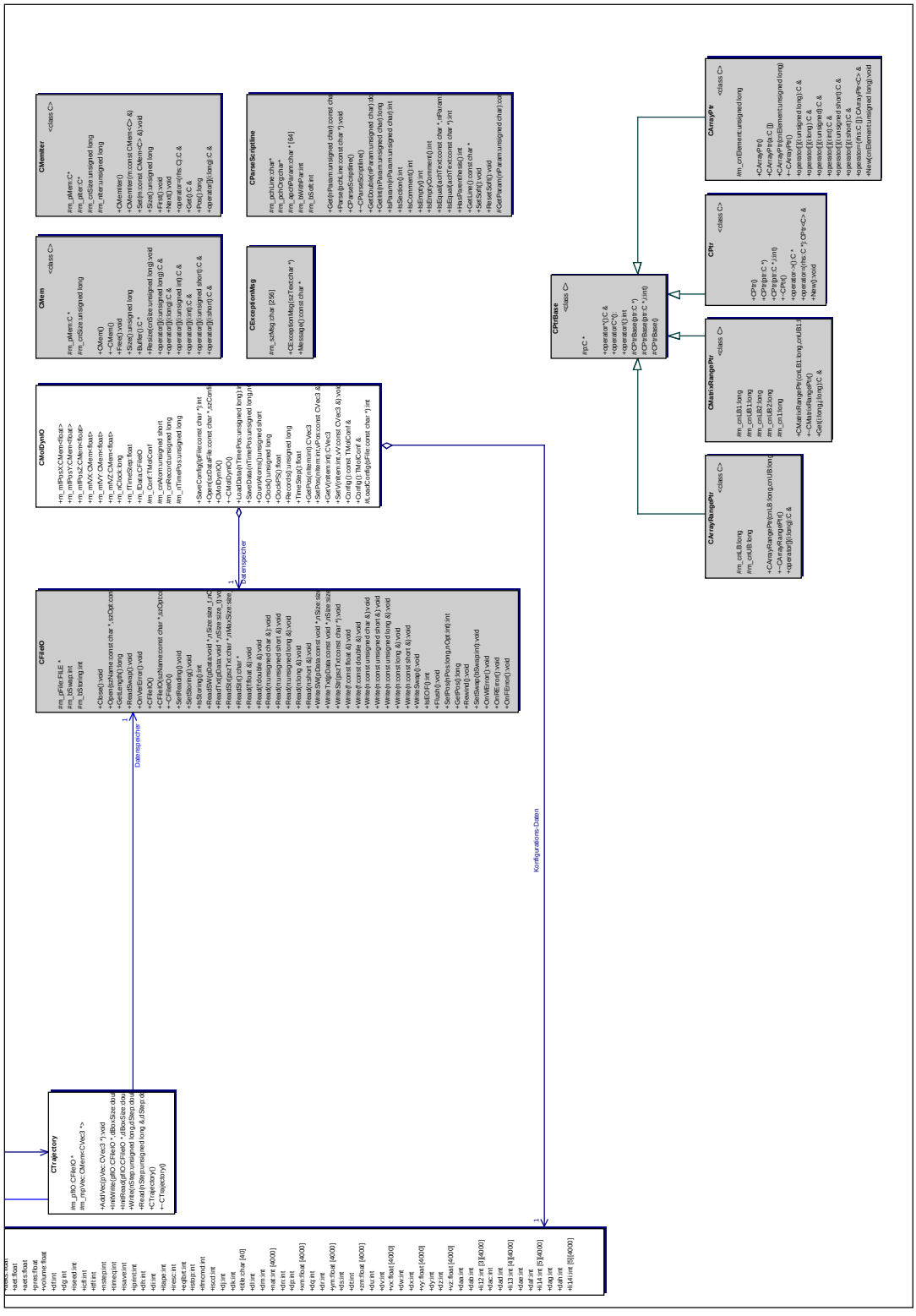


Abbildung 9.3: Teil 3 der Klassenstruktur von StructC (schwarzer Hintergrund) und dynC (weißer Hintergrund).

9.1 Hilfsklassen der Programme „StructC“ und „dynC“

Die Unterscheidung zwischen Hilfsklassen und anderen Klassen ist im allgemeinen nicht eindeutig. In dieser Arbeit werden die Klassen als Hilfsklassen betrachtet, die entweder für den eigentlichen Strukturaufbau bzw. der eigentlichen Simulation nicht benötigt werden, oder so grundlegende Aufgaben erfüllen, dass sie unabhängig von einem Simulationssystem sind.

9.1.1 Die Klasse CVec3

Die Atome einer Simulation bewegen sich in einem drei dimensionalen Raum, aus diesem Grund wird eine Klasse zur Behandlung drei dimensionaler Vektoren benötigt. Diese Aufgabe erfüllt hier die Klasse `CVec3`, sie besitzt drei Attribute `m_dX`, `m_dY` und `m_dZ`, die die drei Raumkoordinaten im kartesischen Koordinatensystem enthalten.

Zur Erzeugung einer Instanz gibt es drei Konstruktoren. Die Konstruktoren `CVec3(const CVec3 &v)` und `CVec3(double x,double y,double z)` erzeugen je eine Instanz eines Vektors bestehend aus den kartesischen Koordinaten `x`, `y` und `z` bzw. aus den Koordinaten des Vektors `v`. Wohingegen der Konstruktor `CVec3()` einen undefinierten Vektor erzeugt. Um den Vektor einer Instanz zu ändern, d. h. den Attributen der Instanz neue Werte zuzuweisen, gibt es den Zuweisungsoperator (operator=). Dieser weist dem lokalen Vektor der gewählten Instanz die Koordinaten des rechts vom Operator stehenden Vektors zu. Desweiteren besteht, mit der Methode `SetZero()` die Möglichkeit einen Nullvektor zu erzeugen.

Die Eigenschaften eines Vektors können mit verschiedenen Methoden ausgelesen werden. Die einzelnen Komponenten der kartesischen Koordinaten werden von den Methoden `GetX()`, `GetY()` und `GetZ()` zurückgegeben. Ein direktes Auslesen der Koordinaten aus den Attributen wird auf Grund der Datenkapselung, die zur Datensicherheit immer verwendet werden sollte, verhindert. Mit den Methoden `GetR()`, `GetPhi()` und `GetTheta()` ist es auch möglich die Koordinaten in Form von Kugelkoordinaten zu bekommen. Die in den Attributen gespeicherten kartesischen Koordinaten werden hierzu in die, in Abbildung 9.4 gezeigten, Kugelkoordinaten überführt. Die zu ihrer Transformation notwendigen mathematischen Formeln sind zum Beispiel bei Stöcker [81] nachzulesen. Zur Bestimmung der Länge eines Vektors sind zwei Methoden implementiert, zum Einen die Methode `GetAbs()`, die äquivalent zu `GetR()` ist und die Länge des Vektors liefert und zum Anderen die Methode `GetAbsSqr()`, die das hier häufig benötigte und schneller zu bestimmende Quadrat der Länge bestimmt.

Additionen, Subtraktionen und Multiplikationen mit Skalaren und Vektoren, so wie Divisionen durch skalare Werte sind ebenfalls durch entsprechende Opera-

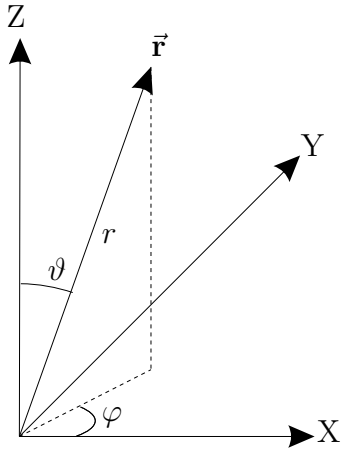


Abbildung 9.4: Darstellung des Vektors $\vec{\mathbf{r}}$ in kartesischen Koordinaten und in Kugelkoordinaten.

toren (+, -, * und /) möglich. Zu beachten ist, das die Multiplikation (*) zweier Vektoren das Skalarprodukt liefert. Das Kreuzprodukt bzw. Vektorprodukt wird von der Methode `vep` ausgeführt, die in zwei Varianten existiert. Bei der Übergabe eines Vektors wird das Vektorprodukt zwischen dem lokalen Vektor (den eigenen Attributen) und dem übergebenen Vektor berechnet und zurückgegeben. Werden zwei Vektoren übergeben, so wird das Vektorprodukt zwischen den beiden Vektoren dem lokalen Vektor zugewiesen, d. h. in den lokalen Attributen gespeichert.

Zum Vergleich von zwei Vektoren sind die beiden logischen Vergleichsoperatoren auf Gleichheit (==) und Ungleichheit (!=) implementiert. Einen Test auf den Nullvektor liefert die boolesche Methode `IsZero()`.

Die Berechnung des Ortsvektors zur Platzierung eines Atoms wird von der Lage der zuvor platzierten Atome bestimmt, wie es im Abschnitt 3.4 beschrieben ist. Zur Bildung des dort benötigten Richtungsvektors $\vec{\mathbf{r}}_K = (1, \vartheta, \varphi)^T$ aus den Kugelkoordinaten ϑ und φ wird die Methode `SetSphere(double dR,double dTheta,double dPhi)` verwendet. Diese wandelt die Kugelkoordinaten $r = 1$, ϑ und φ in die zugehörigen kartesischen Koordinaten um (siehe [81]), die dann in den entsprechenden Attributen `m_dX`, `m_dY` und `m_dZ` abgespeichert werden. Früher wurde hierfür die Methode `Rot(CVec3 &vRes,double daX,double daY,double daZ)` verwendet, die einen Vektor im kartesischen Koordinatensystem um die drei Raumachsen dreht. Bei der Bestimmung des Ortsvektors muss dieser aus dem lokalen Koordinatensystem des Atoms in das globale kartesische Koordinatensystem zurück transformiert werden. Die Rücktransformation erfolgt mit Hilfe der Methode `BasisT(const CVec3 &vB1,const CVec3 &vB2,const CVec3 &vB3)`. Diese kann durch die Transformation mit der Methode `Basis(const CVec3 &vB1,const CVec3 &vB2,const CVec3 &vB3)` wieder umkehrt werden. Die Methode `Basis` erzeugt aus den Basisvektoren `vB1`, `vB2` und `vB3` eine Matrix mit den Basisvektoren als Zeilenvektoren. Um einen Vektor in ein Koordinatensystem

mit den Basisvektoren `vB1`, `vB2` und `vB3` zu transformieren, wird diese Matrix mit dem zu transformierenden Vektor multipliziert:

$$\begin{pmatrix} vB1_x & vB1_y & vB1_z \\ vB2_x & vB2_y & vB2_z \\ vB3_x & vB3_y & vB3_z \end{pmatrix} \begin{pmatrix} m\_x \\ m\_y \\ m\_z \end{pmatrix} \rightarrow \begin{pmatrix} m\_x \\ m\_y \\ m\_z \end{pmatrix} \tag{9.1}$$

Die Rücktransformation erzeugt mit der Methode `BasisT` eine zur ersten Methode transponierte Matrix, die zur Transformation ebenfalls mit dem lokalen Vektor multipliziert wird. Damit beide Methoden tatsächlich Koordinatentransformationen ausführen, müssen die Basisvektoren die beiden folgenden Bedingungen erfüllen:

- Die Länge eines Basisvektors muss Eins betragen.
- Die Basisvektoren müssen orthogonal zueinander stehen.

Beide Bedingungen sind notwendig, damit Hin- und Rücktransformation zusammen die Einheitsmatrix ergeben:

$$\begin{pmatrix} vB1_x & vB1_y & vB1_z \\ vB2_x & vB2_y & vB2_z \\ vB3_x & vB3_y & vB3_z \end{pmatrix} \begin{pmatrix} vB1_x & vB2_x & vB3_x \\ vB1_y & vB2_y & vB3_y \\ vB1_z & vB2_z & vB3_z \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{9.2}$$

Die bisher beschriebenen Methoden sind als „inline“ Methoden kodiert, d. h. der Maschinencode der Methoden wird bei jedem Aufruf direkt an der auszuführenden Stelle im Programm eingefügt und nicht in einer eigenen Unterroutine gespeichert, die bei jedem Aufruf ausgeführt wird. Dies führt zu einer merklich höheren Ausführungsgeschwindigkeit des Programms, die durch einen etwas längeren Programmcode erkauft werden muss. Die höhere Ausführungsgeschwindigkeit ist insbesondere bei den Methoden dieser Klasse wichtig, da sie in den Programmen sehr häufig verwendet werden. Die „inline“ Kodierung der Methoden `GetX()`, `GetY()` und `GetZ()` eröffnet zum Beispiel die Möglichkeit, direkt auf die Attribute zuzugreifen, ohne den Nachteil einer unbeabsichtigten Änderung der Daten in Kauf nehmen zu müssen. In `CVec3` wird einzig die Methode `Serialize(CFileIO &f)` konventionell kodiert, da sie nur relativ selten beim Speichern oder Laden der Strukturdatei (DYN) bzw. der Trajektorien aufgerufen werden muss. Ihre Aufgabe ist es, die Attribute in einem festen Format in die im Parameter `f` festgelegte Datei zu speichern bzw. auszulesen.

9.1.2 Die Klasse CPeriodicBox

Eng verbunden mit der Klasse `CVec3` ist die Klasse `CPeriodicBox`. Sie ist ausschließlich zur Bestimmung von Abständen und Ortsvektoren innerhalb des periodischen Simulationsraumes bestimmt. Mit ihrer Methode `InvPBC(CVec3 *pv)`

wird der Distanzvektor `*pv` des realen Raumes (ohne periodische Randbedingungen) in einen Distanzvektor des periodischen Simulationsraumes umgewandelt. Alternativ kann dieser auch direkt mit der Methode `PBCSub(CVec3 &vBA,const CVec3 &vA,const CVec3 &vB)` aus den Ortsvektoren in `vA` und `vB` bestimmt und umgewandelt werden. Ein neu berechneter Ortsvektor `*pv` eines Atoms muss, falls dieser aus dem Simulationsraum hinauswandert, durch die Methode `PBC(CVec3 *pv)` wieder in den periodischen Simulationsraum verschoben werden. Die für diese Berechnungen notwendige Kantenlänge `dBoxl` des Simulationsraumes kann sowohl bei der Erzeugung im Konstruktor, als auch danach mit der Methode `SetBoxSize(const double dBoxl)`, festgelegt werden. Mit den Methoden `GetBoxSize()` und `GetInvBox()`, die die inverse Länge des Simulationsraumes liefert, ist es möglich diesen Wert bzw. dessen Inverse in anderen Klassen wieder zu verwenden.

Da die Länge des Simulationsraumes für alle Atome gleich ist, wird von `CPeriodicBox` nur eine Instanz benötigt. Diese wird in beiden Programmen in der globalen Variablen `pbx` abgelegt, auf die dann sämtliche Klassen direkt zugreifen können.

9.1.3 Die Klasse CAtomGroup

Die Klasse `CAtomGroup` ordnet jeder Atomsorte innerhalb einer Sorte von Molekülen eine Gruppennummer zu. Bei der Ausgabe der Trajektorie im alten Datenformat werden diese Gruppennummern dann zur Unterscheidung der verschiedenen Sorten von Atomen und Molekülen verwendet.

Eine Gruppennummer wird mit der Methode `Add(const char *szStructure,const char *szElement)` durch hinzufügen einer neuen Kombination aus Elementnamen und Strukturnamen erzeugt, diese kann dann mit der Methode `GetID(const char *szStructure,const char *szElement)`, zum Beispiel bei der Erzeugung der Trajektorie, wieder ausgelesen werden. Mit Hilfe der Methoden `GetStructure(WORD nID)` und `GetElement(WORD nID)` lassen sich die entsprechenden Struktur- oder Elementnamen anhand der Gruppennummer wieder rekonstruieren. Die Anzahl der aktuell verwalteten Gruppennummern lässt sich mit `Count()` ermitteln. Da die Daten auch in die Strukturdatei eingehen sollten, wurde die hierfür zuständige Methode `Serialize(CFileIO &f)` implementiert.

9.1.4 Die Klasse CFileIO

Bei der im folgenden beschriebenen Klasse und bei den in den beiden folgenden Abschnitten beschriebenen Klassen handelt es sich um allgemeine Klassen zur Vereinfachung der Programmierung unter C++, prinzipiell sind diese Klassen nicht auf die MD-Simulation beschränkt.

Die Klasse `CFileIO` ist für die Speicherung von Daten im binären Datenformat zuständig. Sie kapselt nicht nur die C basierte Dateiverwaltung in einer Klasse ab, sondern erlaubt auch die Verwendung der Binärdaten auf unterschiedlichsten Rechnersystemen. Unterstützt werden sowohl Rechnersysteme mit einer internen Datendarstellung im (Least Significant Byte) LSB-Format, wie Intel-PC's, als auch Systeme mit einer Darstellung im (Most Significant Byte) MSB-Format, die zum Beispiel in Mac und HP-Systemen mit Motorola Prozessoren oder PA-RISC Architektur verwendet werden.

Die Klasse `CFileIO` besitzt zwei Konstruktoren. Der Standardkonstruktor erzeugt eine Instanz mit undefinierten Attributen. In ihr muss die zu bearbeitende Datei, mit Hilfe der Methode `Open`, nachträglich geöffnet werden, wobei gleichzeitig die Attribute entsprechend gesetzt werden. Der andere Konstruktor erzeugt eine Instanz, die eine als Parameter übergebene Datei im ebenfalls übergebenen Modus öffnet.

Die Methode `Open` besitzt, wie der zweite Konstruktor, zwei Parameter. Der Erste gibt den Namen der zu öffnenden Datei an. Wird hier ein Null-Zeiger übergeben, so wird eine temporäre Datei erzeugt, die nach dem Schließen der Datei automatisch gelöscht wird. Der zweite Parameter bestimmt den Modus in dem die Datei geöffnet wird, verwendet werden hierbei die in C für `fopen` üblichen Angaben, wie „wb“, „rb“ oder „w+b“.

Geschlossen wird eine Datei durch Aufruf der Methode `Close()`. Diese wird auch automatisch vom Destruktor aufgerufen, wodurch die Datei bei Zerstörung des Objektes, z. B. am Ende einer Routine, automatisch geschlossen wird. Zur Steuerung des Datenzugriffs gibt es die Methoden `GetLength()`, `IsEOF()`, `Flush()`, `GetPos()`, `SetPos`, `Rewind()`,`SetReading()`, `SetStoring()` und `IsStoring()`, deren Funktionalität sich aus dem jeweiligen Namen ergibt. Hervorzuheben ist hier die Methode `SetPos` mit der die aktuelle Position innerhalb der Datei verändert werden kann. Ihr werden bis zu zwei Parameter übergeben, der erste Parameter enthält die Position bzw. Verschiebung und der zweite optionale Parameter die Positionierungsart:

- Die Angabe `SEEK_SET` interpretiert die Angabe im ersten Parameter als Position vom Dateianfang. Dies ist auch der Standardwert, der ohne Angabe dieses Parameters verwendet wird.

- Bei der Angabe von `SEEK_END` wird der im ersten Parameter angegebene Wert als Position relativ zum Dateiende betrachtet.

- Der im ersten Parameter angegebene Wert wird als Position relativ zur aktuellen Position interpretiert, wenn im zweiten Parameter `SEEK_CUR` angegeben wird.

Die Methode `SetPos` liefert als Rückantwort einen Wahrheitswert über die Gültigkeit der neuen Position.

Wie oben bereits erwähnt, kann die Klasse `CFileIO` zwischen binären Daten im MSB- und LSB-Format unterscheiden. Mit Hilfe der Methode `SetSwap(BOOL bSwap)` wird hierzu festgelegt, ob die einzelnen Bytes eines größeren Zahlwertes im Datenstrom vertauscht werden müssen oder nicht. D. h. ob die zu speichernden oder gespeicherten Daten bei einer Maschine mit interner LSB-Darstellung im MSB-Format gespeichert / ausgelesen bzw. bei einer internen MSB-Darstellung in das LSB-Format konvertiert werden müssen. Um automatisch das Darstellungsformat der gespeicherten Daten feststellen zu können, wurden die beiden folgenden Methoden implementiert. Die Methode `WriteSwap()` schreibt auf Rechnern mit LSB-Format, d. h. für Rechnern, die der Konvention von Intel folgen, die Zeichenfolge „IM“ und auf Rechnern mit MSB-Format, die der Konvention von Motorola folgen, „MI“ in die Datei. Ausgelesen und interpretiert wird diese Zeichenfolge in der Lesephase mit der Methode `ReadSwap()`.

Für die primitiven Zahlen-Datentypen gibt es jeweils die Methoden `Write` bzw. `Read` zum Schreiben bzw. Lesen der Daten. Text, der natürlich nicht vertauscht werden darf, wird mit der Methode `WriteStr(const char *pszTxt)` gespeichert und kann mit den Methoden `ReadStr(char *pszTxt, size_t nMaxSize)` und `ReadStr()` wieder ausgelesen werden. Letztere gibt hierzu einen Zeiger auf einen Speicherbereich mit dem eingelesenen Text zurück, dieser Speicher muss später mit der Funktion `free` wieder freigegeben werden. Unspezifische Daten, die keine Unterscheidung zwischen Rechnersystemen erfordern, lassen sich direkt mit der Methode `WriteTxt(const void *pData, size_t nSize)` speichern und mit der Methode `ReadTxt(void *pData, size_t nSize)` wieder auslesen. Im Gegensatz dazu, wird das Datenformat des Rechners bei Verwendung der Methoden `WriteSW(const void *pData, size_t nSize, size_t nCnt=1)` und `ReadSW(void *pData, size_t nSize, size_t nCnt=1)` berücksichtigt.

Zur Behandlung von Fehlern während des Datenaustausches gibt es vier Methoden. Wird die Version des Datenformats in der die Daten gespeichert wurden nicht unterstützt, so wird die Methode `OnVerError()` aufgerufen. Bei Problemen beim Schreiben oder Lesen der Daten werden die `OnWError()` bzw. `OnRError()` aufgerufen. Die Methode `OnFError()` wird bei Problemen, die beim Schließen der Datei auftreten, aufgerufen. Alle vier Methode lösen bei ihrem Aufruf ein Ausnahmeereignis aus, welches wiederum die Programmausführung mit einer entsprechenden Fehlermeldung beendet.

Die Instanzen dieser Klasse werden insbesondere verwendet, um die Strukturdatei zu erzeugen und zu schreiben bzw. um ihre Daten wieder einzulesen. Auch werden die Trajektorien der Atome und die Konfigurationen in gewissen Abständen mit Hilfe dieser Klasse abgespeichert.

9.1.5 Die Klasse CExceptionMsg

Tritt während der Strukturerzeugung oder in der Simulation ein Fehler auf, so wird, wie gerade beschrieben, im Programm eine Außnahmebehandlung (Excepti-

on) ausgelöst. Unterschieden werden hierbei zwei Fälle. Wird bei der Anforderung von Speicher festgestellt, dass für die Anforderung nicht genügend Speicher zur Verfügung steht, wird die Prozedur `ThrowMemEx()` aufgerufen. Diese ruft wiederum die Prozedur `ThrowException` mit dem festen Text „Out of Memory“ auf. Diese löst dann eine Ausnahme mit der Klasse `CExceptionMsg` und dem ihr übergebenen Text aus. In allen anderen Fällen wird die Prozedur `ThrowException` direkt aufgerufen. Die Prozedur `ThrowException` wird mit einer variablen Anzahl von Parametern, entsprechend der Prozedur `printf` aufgerufen. Mit den übergebenen Parametern wird ein Text erzeugt, der dann zur Erzeugung einer Ausnahme durch die Klasse `CExceptionMsg` verwendet wird.

Die Klasse `CExceptionMsg` besteht aus drei Teilen:

1. Der Konstruktor wird bei der Erzeugung einer Ausnahme aufgerufen, er enthält als einzigen Parameter ein Textfeld mit der Fehlermeldung.
2. Die geschützte Variable `m_szMsg` nimmt diese Fehlermeldung auf.
3. Die Fehlermeldung kann durch die Methode `Message()` ausgegeben werden.

Nachdem in `ThrowException` mit `throw CExceptionMsg(szMsg)` eine Ausnahme ausgelöst wurde, wird diese bis zum Hauptprogramm weitergereicht, um dort abgefangen zu werden. Vor Beendigung des Programms wird in der Fehlerbehandlung mit der Methode `Message()` die zugehörige Fehlermeldung ausgeben.

9.1.6 Dynamische Speicherverwaltung

Die dynamische Speicherverwaltung ist in C und C++ aufgrund zweier softwaretechnischer Aspekte als problematisch einzustufen. Zum Einen ist der Zugriff auf Speicherplätze außerhalb des reservierten Speicherbereichs möglich, und zum Anderen erfolgt die Freigabe des reservierten Speichers nach Ende der Nutzung nicht automatisch. Beide Probleme werden durch Nutzung der Template-Klassen `CMem`, `CMemIter`, `CPtrBase` und deren Nachfahren gelöst.

Die Klasse `CPtrBase` bildet die Basisklasse zur Verwaltung von dynamischen Instanzen von Objekten. Sie stellt einerseits sicher, dass bei der Erzeugung der Instanz eines Objektes genügend Speicher zur Verfügung steht, und sorgt andererseits für eine korrekte Löschung der Instanz, d.h. vor der Löschung der Instanz oder der, die Instanz enthaltenen Variablen wird der zugehörige Destruktor aufgerufen. Wird nur eine Instanz eines Objektes benötigt, die im dynamischen Speicher erzeugt und verwaltet werden soll, so ist zu deren Kapselung die von `CPtrBase` abgeleitete Klasse `CPtr` vorgesehen. Einfache Felder von Objekten werden mit der ebenfalls von `CPtrBase` abgeleiteten Klasse `CArrayPtr` verwaltet. Die Nummerierung des von `CArrayPtr` erzeugten Feldes beginnt immer bei Element Null. Sind hingegen bei der Deklaration die Ober- und Untergrenze des Feldes bekannt, wie zum Beispiel in der Berechnung des reziproken Anteils der Ewaldmethode, wird das Feld in einer Instanz der Klasse `CArrayRangePtr` erzeugt und

verwaltet, mit ihr ist es dann möglich das Feld mit einer beliebigen Feldnummer zu beginnen. Bei zweidimensionalen Feldern ersetzt man `CArrayRangePtr` durch die Klasse `CMatrixRangePtr`. Da all diese Klassen auf der Klasse `CPtrBase` beruhen, haben sie alle dessen Eigenschaften geerbt. Dies bedeutet zum Beispiel, dass in allen Instanzen bei zu wenig Speicher automatisch ein Ausnahmeereignis ausgelöst wird. Hervorzuheben ist außerdem, dass der reservierte Speicher, bei Zerstörung der die Instanz enthaltenen Variablen, automatisch freigegeben wird, und dass bei Zugriffen auf einzelne Objekte des Feldes im Debug-Modus eine Prüfung auf Einhaltung der Feldgrenzen durchgeführt wird.

Die oben beschriebenen Klassen `CArrayPtr` und `CArrayRangePtr` dienen nur zur Verwaltung statischer Felder von Objekten. Für die Verwaltung dynamischer Felder wird hingegen die Template-Klasse `CMem` verwendet. In `CMem` werden aber nur Feldelemente von einfachen Variablen und Zeigern erzeugt. Im Gegensatz zu den abgeleiteten Klassen von `CPtrBase` ist es nicht möglich Felder von Objekten zu verwalten, da bei der hier verwendeten Variante der Speicherreservierung die Konstruktoren der Objektinstanzen nicht aufgerufen werden. In `CMem` wird die Dynamik der Feldgrößen über die Methode `Resize` gesteuert. Die Einhaltung der hierin festgelegten Feldgrenzen wird, wie bei der anderen Klassenfamilie, im Debug-Modus bei jedem Zugriff überprüft.

Der direkte Zugriff auf ein Feldelement über den in den oben beschriebenen Klassen implementierten Operator `[]` ist aufwendig, da bei ihm die Position des Feldelements im Speicher bei jedem Zugriff neu berechnet werden muss. Besser ist häufig ein iterativer Zugriff, der allerdings einen sequentiellen Ablauf der Auswertung der Feldelemente benötigt. Der iterative Zugriff auf eine Instanz der Klasse `CMem` erfolgt über die Template-Klasse `CMemIter`. Mit Hilfe der Methode `First()` gelangt man an den Anfang des Feldes, von dort kann man mit der Methode `Next()` zum nächsten Feldelement gelangen. Nützlich ist die Klasse zum Beispiel bei den Listen von Zeigern auf die wechselwirkenden Atome und deren Wechselwirkungsparametern. Aber auch bei der Verwaltung der Liste der Atome eines Moleküls oder innerhalb der Verwaltung der Moleküle beschleunigt `CMemIter` deren Auswertung.

9.1.7 Die Klasse CParseScriptline

Um effizient auf die Skripte der Kraft-, Struktur- und Simulations-Datenbank zugreifen zu können verwenden die Programme eine Instanz der Klasse `CParseScriptline`. Diese teilt eine an die Methode `Pharse(const char *pchLine)` übergebene Zeile eines Skripts in durch spezielle Zeichen abgetrennte Abschnitte ein, die in die nachfolgende Auswertung als Parameter eingehen. So erfolgt eine einfache Aufspaltung einer Zeile in zwei Parameter bei einem Leerzeichen, einem Komma oder einem Gleichheitszeichen. Taucht in einer Zeile ein Semikolon auf, so wird der nachfolgende Text als Kommentar gewertet. Zeilen, die Klammern enthalten werden speziell behandelt, das äußerste Klammerpaar führt

wie ein Leerzeichen zu einer Aufspaltung der Zeile in zwei Abschnitte. Weitere Klammerpaare, so wie der Text vor und innerhalb der Klammerpaare werden in diesem Durchlauf als Einheit angesehen. Somit ist es möglich, ganze Anweisungen als Parameter zu übergeben und diese erst in der weiteren Verarbeitung, wiederum durch `CParseScriptline` weiter aufgespalten zu können. Am Beispiel „add 1 Bisacrylamid(6,7,8)“ lässt sich dies verdeutlichen. Im ersten Durchlauf wird die Zeile in die Parameter „add“, „1“ und „Bisacrylamid(6,7,8)“ aufgeteilt, erst in der Weiterverarbeitung wird der Parameter „Bisacrylamid(6,7,8)“ ebenfalls von `CParseScriptline` in weitere Parameter aufgeteilt.

Die nachfolgenden Methoden werden von `CParseScriptline` zur einfacheren Auswertung des Skripts zur Verfügung gestellt. Eine einfache Rückgabe eines einzelnen Parameters erfolgt über die Methode `Get(BYTE nParam)`. Liegt `nParam` außerhalb der Feldgrenzen, wird eine Ausnahmebehandlung ausgelöst, d. h. ein Fehler zurückgegeben. Abgeschaltet wird diese Überprüfung durch Aufruf der Methode `SetSoft()` und wieder eingeschaltet mit `ResetSoft()`. Ebenfalls implementiert sind die Methoden `GetInt(BYTE nParam)` und `GetDouble(BYTE nParam)`, die den Parametertext automatisch in numerische Werte entsprechenden Typs umwandeln. Zur Steuerung der Auswertung mittels Fallunterscheidungen sind die Methoden `IsParam(BYTE nParam)`, `IsSection()`, `IsComment()`, `IsEmpty()`, `IsEmptyComment()` und `HasParenthesis()` implementiert. Hervorzuheben sind hier die Methoden `IsEqual(const char *achText)` und `IsEqual(const char *achText,BYTE nParam)`, die einen Vergleich des nullten Parameters bzw. des Parameters `nParam` mit dem angegebenen Text ohne Unterscheidung zwischen Groß- und Kleinbuchstaben durchführen.

9.2 Die Basisklassen

Neben den grundlegenden Klassen aus dem letzten Abschnitt werden nun die Basisklassen beschrieben, die speziell für die Erzeugung der Strukturdatei mit „StructC“ und zur Simulation mit „dynC“ benötigt werden.

Die Basisklassen stellen ein Grundgerüst, der bereits am Anfang diese Kapitels beschriebenen und der Natur nachempfundenen Hierarchie, von Klassen bereit. Die Klasse `CAtom` bezieht die Eigenschaften seines Atoms aus der Beschreibung des Elements, die für jedes verwendete Element in einer Instanz der Klasse `CElement` erzeugt wurde, so wie eigenen Attributen zur Lage und Geschwindigkeit des Atoms. `CAtom` selbst ist Bestandteil der Klasse `CMolecule`. Eine Liste aller Moleküle und aller Elementbeschreibungen wird wiederum in der Klasse `CMoleculeList` verwaltet, die zudem die einzigste Instanz von `CForceField` enthält, mit der sämtliche Wechselwirkungsparameter bestimmt werden.

9.2.1 CElement

Von jedem Element gibt es im allgemeinen mehrere Atome. Damit die charakteristischen Daten eines Elements nicht in jedem Atom separat gespeichert werden müssen, wodurch eine unnötige Redundanz der Daten erzeugt wird, enthält jedes Atom lediglich eine Referenz auf die Instanz der Klasse `CElement` seines Elements. Diese enthält alle für das Element relevanten Daten und kapselt sie gegen Änderungen ab.

Die Daten werden aus `CElement` über Methoden ausgelesen. Die Methode `GetName()` gibt den Namen des Elements und `GetMass()` die Masse bzw. `GetInvMass()` die inverse Masse zurück, die insbesondere bei der Bestimmung der Beschleunigungen Verwendung findet. Im verwendeten Atommodell sind die Lennard-Jones-Parameter charakteristisch für jedes Element, ein Atom wechselwirkt immer mit den selben Parametern. Einzig bei der Wechselwirkung mit dem dritten Nachbarn eines Atoms werden bei bestimmten Elementen andere Parameter verwendet. Der erste Term der Lennard-Jones-Parameter ist $\sqrt{A_{ii}}$ abrufbar mit den Methoden `GetC12()` und `Get3C12()` für den dritten Nachbarn. Der zweite Term $\sqrt{B_{ii}}$ wird durch die Resultate der Methoden `GetC6()` und `Get3C6()` bestimmt.

Von jedem verwendeten Element wird nur eine Instanz erzeugt, diese wird Teil der Klasse `CMoleculeList`, auf die dann, aus den Klassen `CAtom` und `CStructMol` Referenzen, d. h. Zeiger verweisen, wie es auch Abbildung 9.1 Seite 121 zeigt. Die Instanzen der Elemente werden im Programm „StructC“ durch die Methode `Load` der Klasse `CStructMol` wie folgt erzeugt. Mit den aus der Kraft-Datenbank gewonnenen Daten ruft die Methode `Load` die Methode `GetElement` der Klasse `CMoleculeList` auf, die diese Aufgabe an die Methode `AddElement` weiter gibt, falls das Element noch nicht erfasst wurde. `AddElement` ruft wiederum die Methode `CreateElement` der Klasse `CForceField` auf und nimmt das Element in die Klasse `CMoleculeList` auf. Die Methode `CreateElement` erstellt nun endgültig eine Instanz der Klasse `CElement`. Im Gegensatz dazu erzeugt das Programm „dynC“ die Instanzen von `CElement` in Abhängigkeit von den verwendeten Atomen bei der in der Methode `Serialize`. Hierbei liest es die Daten der Atome aus der Strukturdatei mit Hilfe der Methode `Serialize` der Klasse `CAtom` ein und ruft dort die Methode `GetElement` der Klasse `CMoleculeList` auf.

9.2.2 CAtom

Jede Instanz der Klasse `CAtom` enthält die individuellen Eigenschaften eines *einzelnen* Atoms, unter anderem den Orts- und den Geschwindigkeitsvektor. Zur Beschreibung der Art des Atoms wird jedem Konstruktor eine Referenz auf die zugehörige Instanz der Klasse `CElement` übergeben. Diese legt das Atom bzw. im Falle der Kohlenwasserstoffe CH, CH_2 und CH_3 die atomare Einheit fest, so kann zum Beispiel der Namen des Atoms über die Referenz mit `GetName()` ausgele-

sen werden. Der Konstruktor speichert die Referenz zur späteren Verwendung im Attribut `m_pElement`, diese kann mit der Methode `GetElement()` wieder ausgelesen werden. Zur Einordnung in eine Atomgruppe benötigt der Konstruktor zusätzlich eine durch die Klasse `CAtomGroup` eindeutig festgelegte Nummer. Dem Konstruktor kann optional eine Partialladung für das Atom übergeben werden. Die beiden letzten Parameter werden intern in den Attributen `m_nAtomGroupID` und `m_dCharge` gespeichert und können mit den Methoden `GetAtomGroup()` und `GetCharge()` abgefragt werden. Desweiteren wird im Konstruktor die Geschwindigkeit und die auf das Atom wirkende Kraft auf Null gesetzt, so wie der Ortsvektor als undefiniert festgelegt. Die Attribute für Geschwindigkeit und Kraft werden, obwohl sie zur Konstruktion des Simulationssystems nicht benötigt werden, bereits hier deklariert, da sie Teil der in der Strukturdatei (DYN) gespeicherten Daten sind.

Der Im Konstruktor als undefiniert bestimmte Ortsvektor wird im Attribut `m_vPosition` durch Aufruf der Methode `SetPos(const CVec3 &vPos)` festgelegt. Im DEBUG-Modus überprüft diese Methode zusätzlich, ob die angegebene Position innerhalb des Simulationsraumes liegt. Ausgelesen wird der Ortsvektor mit der Methode `GetPos()`, die im DEBUG-Modus ebenfalls die Gültigkeit des Wertes mit Hilfe der Methode `IsValid()` überprüft.

Um jedes Atom eindeutig zu identifizieren wird ihm im Attribut (`m_nAtomCounter`) eine Identifizierungsnummer zugewiesen. Vor Erzeugung der Simulation wird der interne Atomzähler mit `ResetCounter()` auf Eins gesetzt. Die Zuteilung einer Nummer erfolgt im Konstruktor, der die Methode `IncCounter()` aufruft. Die eindeutige Identifizierungsnummer wird durch die Methode `GetCounter()` ausgelesen. Vorwiegend verwendet wird die Nummer bei der Auflistung der Bindungsverhältnisse in der Datei „DynC_Dump.txt“ und zur Anzeige von Atom spezifischen Fehlermeldungen. Sie wird auch für die Erzeugung einer Protein-Datenbank (PDB) benötigt.

Wie oben bereits erwähnt gehen die Daten von `CAtom` ebenfalls in die Strukturdatei (DYN) ein. Zum speichern und auslesen der Daten aus der Strukturdatei werden diese binär mit der Methode `Serialize(CFileIO &f, WORD nVer, CSimSystem &sys)` in die Datei geschrieben bzw. aus ihr gelesen. Die Methode überprüft hierzu zuerst, ob die in `nVer` übergebene Versionsnummer mit den unterstützten Versionen der Strukturdatei übereinstimmt. Zur Speicherung der Daten werden diese in einfache Datentypen wie „long“ oder „float“ umgewandelt und dann in die Datei geschrieben. Das Auslesen der Daten kehrt diesen Prozess um. Die Daten werden hierbei zuerst in Variable einfacher Datentypen geladen und dann den zugehörigen Attributen der Instanzen von `CAtom` zugewiesen. Zur Identifizierung der hier verwendeten Instanzen der Klasse `CElement` werden deren zugeordnete Namen verwendet. Beim einlesen der Daten wird für die Zuordnung bzw. Erzeugung einer Instanz der Klasse `CElement` auf die, als Parameter in der Variablen `sys` übergebene Instanz der Klasse `CSimSystem` und deren Methode `GetElement(const char *szName)` zurückgegriffen. Zur Behandlung des Orts-

und Geschwindigkeitsvektors eines Atoms wird jeweils die Methode `Serialize` der Klasse `CVec3` aufgerufen. Der Kraftvektor wird nur temporär benötigt und deshalb nicht abgespeichert, während des Ladevorgangs wird er auf Null gesetzt.

9.2.3 CMolecule

Die Atome in einer Simulation setzen sich zu Molekülen zusammen, wobei die Atome eines Moleküls eine Einheit bilden, die sich durch ihre Anordnung und durch eine direkte Wechselwirkung der Atome des Moleküls von den anderen Atomen unterscheiden. Im Simulationssystem wird ein Molekül in Form eine Instanz der Klasse `CMolecule` oder deren Nachfahren nachgebildet.

Bei ihrer Konstruktion erhält jedes Molekül einen Namen, dieser wird dem Konstruktor als Zeichenkette übergeben. Der Konstruktor erzeugt von dieser Zeichenkette eine Kopie, die dann im Attribut `m_pchName` abgelegt wird. Der Name des Moleküls bleibt somit für die Dauer der Existenz der Instanz erhalten. Erst im Destruktor wird die Kopie, zusammen mit den Atomen des Moleküls aus dem Speicher entfernt. Der in `m_pchName` abgelegte Molekülname kann mit der Methode `GetName()` wieder abgerufen werden. Der Molekülname wird zum Beispiel bei der Ausgabe der Report-Datei oder der Protein-Datenbank (PDB) verwendet.

Wichtige Bestandteile von `CMolecule` sind, die in `m_mpAtom` gespeicherte Liste der Atome des Moleküls und die Liste der benachbarten, d. h. direkt verbundenen Atome. Anfangs sind beide Listen leer, erst wenn mit der Methode `AddAtom(CAtom *pAt)` Atome im Molekül eingebunden werden, füllen sich die Listen. In den Listen werden nur Zeiger auf die Instanzen der Atomen des Moleküls eingetragen, da die hier verwendeten Attribute durch die Template-Klasse `CMem` gekapselt werden müssen. Durch die Kapselung besitzen sie zwar eine flexible Feldgröße, können aber keine strukturierten Klassen verwalten. Die Anzahl der Atome in einem Molekül wird von der Methode `CountAtoms()` zurückgegeben, diese greift hierzu auf die durch die Kapselung leicht bestimmbare Länge der Liste zu. Die Masse des Moleküls, die zum Beispiel benötigt wird bei der Berechnung der Dichte des Gesamtsystems und bei der Ausgabe der Dichte in die Report-Datei, lässt durch die Methode `GetMass()` bestimmen.

Der Zugriff auf die Daten der Atome eines Moleküls wird durch spezielle Methoden ermöglicht. Die Methode `GetAtom(short nAtom)` liefert den allgemeinsten Zugriff auf das gewünschte Atom, allerdings muss hierzu die laufende Nummer des Atoms `nAtom` in der Atomliste bekannt sein. Für die beiden am häufigsten verwendeten Eigenschaften gibt es zusätzlich zwei spezielle Methoden, die Methode `GetName(short nAtom)` (nicht zu verwechseln mit `GetName()`) liefert den in `CElement` gespeicherten Namen des Atoms und die Methode `GetPos(short nAtom)` den Ortsvektor des Atoms. Die in den Listen gespeicherten Zeiger auf Instanzen von Atomen eines Moleküls müssen für die Methoden der Klasse `CCreateMol`, die zur Konstruktion der Simulationsstruktur verwendet werden, durch die laufenden Nummern der Atome ersetzt werden. Zu diesem Zweck wur-

de die Methode `GetNo(const CAtom *pAt)` programmiert.

Ein Molekül ist aber mehr als die bloße Summe seiner Atome, in ihm bilden die Nachbarschaftsbeziehungen den zweiten, das Molekül charakterisierenden, Teil. In der Klasse `CMolecule` werden die direkten Nachbarn in einer Liste aufgeführt, die im Attribut `m_mnNeighbour` gespeichert wird. In ihr wird für jedes Atom Speicher für `MAXNB`=8 direkte Nachbarn reserviert. Ein einfacher Zugriff auf die Daten erfolgt über die Methode `GetNeighbour(short nAtom, BYTE nNb)`, die, bei Angabe der Atomnummer und der Nummer des Eintrags, die Nummer des Nachbaratoms zurück liefert. Ist kein Nachbaratom an der angegebenen Stelle eingetragen, so wird der Standardwert `NONB` zurückgegeben.

Die Speicherung der Atomdaten in der Protein-Datenbank (PDB-Datei) erfolgt für jedes Molekül separat, deshalb besitzt `CMolecule` eine spezielle Methode zur Speicherung eines einzelnen Moleküls in der Datenbank. Die Methode `WritePDB(FILE *fOut, int cnMolecule)` speichert zuerst die Atome des Moleküls mit ihren Koordinaten unter der in `cnMolecule` festgelegten Molekülnummer. Diese werden dann durch eine Liste der direkten Bindungen ergänzt. Zusammen mit den Daten der anderen Moleküle bilden sie die Protein-Datenbank.

Zum Datenaustausch mit den anderen Programmen wird die Strukturdatei (DYN) verwendet. Die Aufbereitung und Speicherung der Daten erfolgt, wie in den Klassen `CElement` und `CAtom`, in einer Methode mit dem Namen `Serialize(CFileIO &f, CSimSystem &sys, DWORD nVer)`. Ausgetauscht werden zunächst Molekülname und Anzahl der Atome. Mit Hilfe der Methode `AddDummyAtom()` werden dann leere Instanzen der Klasse `CAtom` erzeugt, die später, durch einen Aufruf der `Serialize`-Methode dieser Instanzen, mit Daten gefüllt werden. Bei der Erzeugung dieser Instanzen wurden gleichzeitig auch die Listen der direkten Nachbarn erzeugt, diese werden ebenfalls in die Strukturgdatei geschrieben bzw. aus ihr ausgelesen und in die Listen eingetragen. Abschließend wird die `Serialize`-Methode einer jeden Instanz der Klasse `CAtom` aufgerufen, wodurch, je nach Zustand der Instanz `f` der Klasse `CFileIO`, die Daten der Atome entweder abgespeichert oder aus der Strukturdatei (DYN) ausgelesen werden.

9.2.4 CSimSystem

Mit den bisher betrachteten Klassen ist es nicht möglich ein Simulationssystem aufzubauen oder zu simulieren, hierzu muss ein Satz der im vorherigen Abschnitt definierten Moleküle zu einem Simulationssystem zusammengefasst werden. Außerdem modellieren die bisher bearbeiten Klassen nur ausgewählte Aspekte eines Simulationssystems.

Die Komponenten eines Systems werden in der Klasse `CSimSystem` zusammengefasst. Ein wesentlicher Bestandteil dieser Klasse ist die Liste sämtlicher Moleküle, gespeichert im Attribut `m_mpMolecule`. Die Moleküle (`CMolecule`) enthalten Atome (`CAtom`), die wiederum jeweils eine Referenz auf ein definiertes Element (`CElement`) besitzen. Die Instanzen der Elemente selbst müssen in einer

eigenen Liste, hier im Attribut `m_mpElement`, gespeichert werden. Zur Unterscheidung gleicher Elemente in unterschiedlichen Molekülen, werden den Atomen entsprechende Identifikationsnummern zugewiesen, die sie einer bestimmten Atomgruppe zuordnen. Die Verwaltung dieser Nummern erfolgt in der Klasse `CAtomGroup`, auf die mit der Methode `GetAtomGroup()` zugegriffen wird. Für den strukturellen Aufbau der Moleküle und für die Verknüpfung der Atome sind die Kraftfelder und ihre Parameter verantwortlich. Sie werden in der im Attribut `m_pFF` gespeicherten Instanz der Klasse `CForceField` definiert. Der Zugriff auf dieses Attribut erfolgt über die Methode `GetForce()`, die das Attribut vor ungewollten oder verdeckten Veränderungen schützt.

Die im Folgenden beschriebenen Attribute werden nur in einem der beiden Programme verwendet, dennoch müssen sie hier, und nicht in einem der Nachfahren `CSimSystemStruct` oder `CSimSystemSim`, deklariert werden, da sie in der Strukturdatei mit abzuspeichern sind. Sollen zwei Teilstränge eines Polymers verbunden werden, so gibt es für den Abstand beider Teilstücke einen Toleranzbereich der durch das Attribut `m_dCrwTol` festgelegt wird. Dieser Wert wird nur im Programm „StructC“ mit der Methode `GetCrwTol()` zur Konstruktion des Netzwerkgerüstes ausgelesen. Die Strukturdatei enthält diesen Abstand zur Dokumentation. Der aktuelle Zeitpunkt der Simulation wird in der Variablen `m_nStep`, die den Zeitschritt enthält, und in der Variablen `m_dStep` gespeichert, die den Zeitpunkt in ps enthält. Die aktuelle Schrittweite, die die Zeit zwischen zwei Zeitschritten angibt, wird in `m_dTimeStep` gespeichert. Diese drei Attribute sind für die Fortsetzung einer Simulation wichtig, sie müssen deshalb in jeder Strukturdatei mit abgespeichert werden. Gesetzt werden diese Parameter mit der Methode `SetSimData(DWORD nStep, double dStep, double dTimeStep)`.

Die Verwaltung der Moleküle erfolgt durch die drei Methoden `AddMolecule(CMolecule *pMolecule)`, `GetMolecule(DWORD nMolecule)` und `CountMolecule()`. Mit ihnen werden Moleküle in das System aufgenommen, ausgewählt oder ihre Anzahl bestimmt. Ähnliche Methoden existieren auch für die Klasse `CElement`, jedoch mit dem Unterschied, dass die geschützte Methode `AddElement(const char *szName)` automatisch aufgerufen wird, falls das Element mit `GetElement(const char *szName)` nicht gefunden wird. Auf ein bestimmtes Element der Liste kann mit der Methode `GetElement(WORD nElement)` zugegriffen werden.

Zur Visualisierung und Auswertung der Simulationsstruktur werden drei Methoden zur Verfügung gestellt:

1. Zur Visualisierung der Struktur werden die Positionen und direkten Bindungen der Atome mit der Methode `WritePDB(FILE *fOut)` im „Protein Database“-Format in die in `fOut` angegebene Datei geschrieben. Die gespeicherten Strukturdaten können mit den Programmen „Rasmol“ oder „RasTop“ räumlich dargestellt werden.

2. Für die statistische Auswertung der Häufigkeiten einer Atomgruppe wird

die, von der Klasse CSimulation erzeugte, Report-Datei mit Hilfe der Methode `ReportGroupStat(FILE * fOut)` um entsprechende Einträge ergänzt.

3. Die Strukturdatei (DYN), die die charakteristischen Daten einer Simulation enthält, wird von der Methode `Serialize(CFileIO &f)` erzeugt. Sie dient sowohl dem Austausch der Daten zwischen „StructC“ und „dynC“, als auch der Speicherung der Daten eines Phasenraumpunktes zu bestimmten Zeitpunkten.

In der Methode `Serialize(CFileIO &f)` werden als erstes die Verwaltungsdaten zur Identifizierung der Datei als Strukturdatei und die Version des Dateiformats gespeichert. Anschließend werden die einfachen Attribute, die über keine eigene `Serialize`-Methode verfügen, wie die Anzahl der Moleküle und die Länge des Simulationsraumes, gespeichert. Zur Speicherung und später auch zum auslesen der Instanzen der Klassen CForceField, CAtomGroup und der Instanzen aller Moleküle werden die Methoden `Serialize(CFileIO &f)` dieser Instanzen aufgerufen.

Ausgelesen werden die Daten dann später wieder mit der selben Methode, so ist gesichert, dass die Daten in der selben Reihenfolge wie sie geschrieben wurden wieder eingelesen werden. Doch bevor die Daten eingelesen werden können, müssen sämtliche Instanzen der Klassen CMolecule, CElement und CForceField gelöscht werden. Nach dem Einlesen des ersten Teils der Daten wird in `m_pFF` eine neue Instanz der Klasse CForceField erzeugt und es werden ebenfalls die Instanzen der Moleküle erzeugt, deren Attribute später mit einer eigenen `Serialize`-Methode eingelesen werden. Die Instanzen der Moleküle werden durch Aufruf der virtuellen Methode `AddNewMolecule()` erzeugt, die von den Nachfahren von CSimSystem zu Anpassung überschrieben werden muss. In den Nachfahren erzeugt ein Aufruf der Methode `AddNewMolecule()` in Abhängigkeit vom Nachfahren entweder ein leeres Molekül der Klasse CMoleculeStruct oder der Klasse CMoleculeSim, so dass immer mit der korrekten Klasse für die Moleküle gearbeitet wird.

9.2.5 CForceField

Die Eigenschaften eines Elements, hierzu zählen insbesondere dessen Wechselwirkungen, werden durch für jedes Element charakteristische Parameter bestimmt. Diese Parameter werden in Instanzen der Klasse CForceField verwaltet. In CForceField wird ein neues Element durch Aufruf der Methode `AddElement` erzeugt, hierbei erwartet die Methode als Übergabeparameter, neben dem Namen des Elements und seiner Masse, dessen Partialladung und Lennard-Jones-Parameter. Die übergebenen Daten werden in einem Feld mit einer speziellen Datenstruktur (TElement) gespeichert. Mit Hilfe dieser Daten kann dann mit der

Methode CreateElement, wie im Abschnitt 9.2.1 beschrieben, ein neues Element für die Instanz der Klasse CMoleculeList erzeugt werden.

Die intramolekularen Wechselwirkungen werden ebenfalls durch Methoden (Add12, Add13, Add14 und Add14i) in speziellen Datenstrukturen gespeichert, diese enthalten neben den Namen der wechselwirkenden Elemente die für die jeweilige Wechselwirkung wichtigen Zusatzparameter wie Federkonstante dK, Gleichgewichtsabstand dB0, Gleichgewichtswinkel dAngle oder Periodizität nPeriod. Die Übergabe der Wechselwirkungsparameter an die Methoden anderer Klassen erfolgt mit Hilfe der Methoden Get12, Get13, Get14 und Get14i, die die Daten in Form der Klassenobjekte TParam12, TParam13, TParam14 und TParam14i zurückgeben. Abbildung 9.1 auf Seite 121 zeigt den genauen Aufbau dieser Objekte.

Zur Übernahme und Speicherung vollständiger Parametersätze und ganzer Strukturen gibt es drei zusätzliche Methoden. Die Methode AddForce übernimmt die Parameter anderer Instanzen der Klasse CForceField in die eigene Instanz. Um die Daten einer Kraft-Datenbank, bei der es sich um eine Textdatei mit fester Struktur handelt, die im Abschnitt 2.3 beschrieben wurde, einzulesen und zu verarbeiten, wird ein Handle auf die Datei an die Methode Load übergeben. Die Methode Load sucht nach den Abschnitten „Element“, „Bond“, „Angle“, „Dihedral“ und „Improper“ in der Kraft-Datenbank und liest die zugehörigen Parameter ein. Mit Hilfe der Methoden AddElement, Add12, Add13, Add14 und Add14i werden aus den Parametern neue Feldelemente erzeugt. Zum speichern und laden einer Instanz von CForceField in einer Strukturdatei, wird die Methode Serialize verwendet. Diese ist Teil des Abspeicher- und Ladevorgangs der Klasse CMoleculeList, die auch alle anderen Instanzen der Klasse CForceField verwaltet.

9.3 Die Klassen zum Strukturaufbau mit „StructC“

Mit dem Programm „StructC“ werden die Startstrukturen der für „DynC“ benötigten Simulationssysteme erzeugt. Zur Erzeugung eines Simulationssystems wird eine spezielle, als Textdatei gespeicherte Struktur-Datenbank benötigt. Diese Datenbank enthält Informationen über den Aufbau des Simulationsraumes und über die zu simulierenden Moleküle. Diese Informationen können alternativ auch über einen Satz von Datenbanken verteilt sein, wenn diese in der ersten Datenbank angegeben werden. Mit Hilfe einer Instanz der in Abschnitt 9.3.2 beschriebenen Klasse CSimSystemStruct, die die Struktur-Datenbank zur Erzeugung des Simulationsraumes auswertet, wird das Simulationssystem erzeugt. Die Erzeugung der Moleküle erfolgt hierbei in einer eigens hierfür programmierten Klasse CCreateMol, die ausführlich in Abschnitt 9.3.4 beschrieben wird. In den folgenden Unterabschnitten werden die weiteren zum Strukturaufbau benötigten

Klassen und Methoden beschrieben.

9.3.1 CMoleculeStruct

Für die Generierung eines Moleküls muss die allgemeine Klasse `CMolecule` durch zusätzliche Methoden ergänzt werden, zu diesem Zweck wird die von `CMolecule` abgeleitete Klasse `CMoleculeStruct` generiert.

Nach Erzeugung eines Atoms ist dieses a priori vollkommen ungebunden. Die Atome mit denen es direkt benachbart ist und eine fest Bindung eingeht werden mit der Methode `SetNeighbour(short nAt1, short nAt2)` festgelegt. Diese Methode trägt in die Liste `m_mnNeighbour`, die Atome an den Positionen `nAt1` und `nAt2` der Atomliste `m_mpAtom` des Moleküls, das jeweils andere Atom als benachbart ein. Verwendet wird diese Liste später bei der Positionierung der Atome und während der Simulation bei der Festlegung der intramolekularen Wechselwirkungen.

Bei der Positionierung der Atome ist es wichtig einen gewissen Mindestabstand zwischen den Atomen einzuhalten. Zur Überprüfung und Einhaltung dieses Mindestabstandes wird unter anderem die Methode `IsFreeInMolecule(const CAtom *at, const CForceField *pFF)` benötigt. Sie vergleicht den Abstand des Atoms `at` zu allen anderen Atomen des selben Moleküls mit $^{10}/_{25}$ des entsprechenden Lennard-Jones-Abstandes und legt damit fest, wann der Mindestabstand zwischen dem Atom `at` und den anderen Atomen des Moleküls eingehalten wird.

In einigen Fällen lässt sich, auf Grund der bisher aufgebauten Molekülstruktur, das neue Atom nicht in die vorhandene Struktur einpassen, dann müssen Teile des bereits fertiggestellten Moleküls entfernt und mit anderen Bindungswinkeln neu aufgebaut werden. Der sukzessive Rückbau des Moleküls erfolgt hierbei mit Hilfe der Methode `DelLastAtom()`, die das zuletzt eingefügte Atom wieder entfernt.

9.3.2 CSimSystemStruct

Ähnlich wie `CMolecule` muss auch die Klasse `CSimSystem` durch spezielle Methoden ergänzt werden. Zusätzlich müssen vorhandene Methoden, die mit der Klasse `CMolecule` arbeiten, durch Methoden ersetzt werden, die mit der von ihr abgeleiteten Klasse `CMoleculeStruct` arbeiten.

Um `CMoleculeStruct` verwenden zu können, werden die beiden Methoden `AddMolecule(CMoleculeStruct *pMolecule)` und `GetMolecule(unsigned int nMolecule)` neu definiert, wobei letztere einen Zeiger auf die gewählte Instanz der Klasse `CMoleculeStruct` zurück gibt. Implementiert werden beide Methoden durch ein type-casting von `CMoleculeStruct` in `CMolecule`, so wie einem anschließenden Aufruf der Methoden `AddMolecule(CMolecule`

`*pMolecule)` bzw. `GetMolecule(unsigned int nMolecule)` der Klasse des Vorfahren `CSimSystem`.

Ein Mindestabstand muss nicht nur zwischen den Atomen eines Moleküls, sondern auch zu allen anderen Atomen, eingehalten werden. Aus diesem Grund wird eine Methode `IsFreePos(const CAtom *at)` definiert, die für das Atom `at` die Einhaltung eines Mindestabstandes zu allen Atomen aller Moleküle überprüft, wobei sich der Mindestabstand wieder aus dem Lennard-Jones-Radius ergibt. Die Überprüfung des Abstandes ist hierbei einfach durch einen sukzessiven Aufruf der Methode `IsFreeInMolecule` für jede Instanz von `CMoleculeStruct` zu erreichen, da in dieser Klasse sämtliche Moleküle gespeichert werden.

Für die Erstellung eines Polymersystems sind zusätzlich die beiden Methoden `GetCrwTol()` und `Density()` implementiert. Die erste Methode gibt den Inhalt des bereits in `CSimSystem` deklarierten Attributs `m_dCrwTol` wieder, das den im Abschnitt 3.4 beschriebenen Toleranzfaktor für den Abstand zweier zu verknüpfender Teilketten eines Polymers enthält. Die zweite Methode bestimmt aus den Massen der einzelnen Moleküle und der Größe des Simulationsraumes mit Hilfe eines Umrechnungsfaktors die Dichte des Systems in $\mathrm{g/cm^3}$.

Die Randparameter der Startstruktur werden im Abschnitt „Structure“ der Struktur-Datenbank (siehe auch Abschnitt 3.3) festgelegt. Aufteilen lassen sie sich in einen Satz Parameter für die Bestimmung der Startstruktur und einen Satz Parameter zu ihrer Speicherung. Mit der Methode `LoadIn(FILE *fIn)` wird der Parametersatz zur Bestimmung der Struktur eingelesen. Die Auswertung erfolgt mit Unterstützung eine Instanz der Klasse `CParseScriptline`, wobei folgende Parameter ausgewertet werden:

BoxSize: Festlegung der Kantenlänge des Simulationsraumes, die im globalen Objekt `pbx` abgespeichert wird.

Step: Bestimmt die Schrittweite der späteren Simulation. Dieser Wert wird in der Regel durch Parameter der Simulations-Datenbank überschrieben.

Debug: Legt den Grad der zusätzlichen Ausgaben im DEBUG-Modus fest. Je größer dieser Wert ist, desto detaillierter wird der Bildungsprozess beschrieben.

CrwTol: Bestimmt die in Abschnitt 3.4 beschriebene Toleranz für den Abstand zweier Teilketten eines Polymers bei deren Verknüpfung.

Include: Aus den hier angegebenen Datenbanken werden die Parameter der Kraftfelder mit Hilfe der Methode `Load(FILE *fIn)` der Klasse `CForceField` ausgelesen.

Der Parametersatz zur Speicherung der Startstruktur wird von der Methode `LoadOut(FILE *fIn)` verarbeitet. Wichtig sind hier zwei Einträge, von denen der Eine („out pdb“) die angegebene Datei erzeugt und die Strukturdaten im

PDB-Format mit Hilfe der Methode `WritePDB(FILE *fOut)` der Klasse des Vorfahrens abspeichert. Der andere Eintrag („out dyn“) erzeugt eine Datei, die die Struktur-Datenbank (DYN) enthält. Gespeichert werden die Daten der Struktur-Datenbank mit Hilfe einer hierfür erzeugten Instanz der Klasse `CFileIO` und der Methode `Serialize(CFileIO &f)` des Vorfahren.

9.3.3 CStructMol

Zur Verwaltung der Molekül- und Monomerstrukturen besitzt „StructC“ die Klasse `CStructMol`. Jedes in der Struktur-Datenbank beschriebene Molekül erzeugt eine neue Instanz dieser Klasse, in der die Struktur des Moleküls in einer zur Weiterverarbeitung besser geeigneten Form abgelegt wird.

In ihrem Konstruktor wird zunächst das Attribut `m_pchName`, das den Namen des in dieser Instanz gespeicherten Moleküls enthält, auf den Null-Zeiger gesetzt. Dies geschieht, um einerseits zu verdeutlichen, dass diese Instanz noch unbenutzt ist, und andererseits, um bereits hier einen definierten Wert für die Freigabe des Speichers für den Fall des Aufrufs des Destruktors zu besitzen. Ausgelesen wird der Name des Moleküls mit der Methode `GetName()`.

Die Molekülstruktur wird mit der Methode `Load(CSimSystem *pSys,FILE *fIn,int nMolecule)` aus der Struktur-Datenbank eingelesen, diese verwendet hierbei eine Reihe von Methoden der Klasse `CStructMol`. Ein Atom bzw. eine Atomgruppe wird in den Aufbau der Struktur aufgenommen, wenn deren Parameter durch die Methode `AddAtom(const CElement *pElement,double dCharge)` in die aktuelle Instanz eingefügt wird. Die in `CStructMol` vorhandenen Listen werden hierfür um jeweils einen Eintrag ergänzt. In diese neuen Felder von `m_mpElement` und `m_mdCharge` werden die Daten, entsprechend den Einträgen der Parameter von `AddAtom`, eingetragen. In die neuen Felder der beiden anderen zu `CStructMol` gehörenden Listen `m_mbBackbone` und `m_mZMat` werden zunächst Standardwerte eingetragen. Damit später auf genau die Daten wieder zugegriffen werden kann, gib die Methode `AddAtom` die Position des neuen Atoms in den Listen zurück. Anhand dieser Position (`nAtom`) können die eben gespeicherte Ladung und der Typ des Elements mit den Methoden `GetCharge(WORD nAtom)` und `GetElement(WORD nAtom)` wieder abgefragt werden.

Die Relationen zwischen den Atomen eines Moleküls werden in die Struktur-Datenbank in einer vorgegebenen Form eingetragen, die einer Z-Matrix ähnelt. In „StructC“ werden diese Daten in der Datenstruktur `TZMat` gespeichert, dessen Aufbau u. a. auf Seite 121 in Abbildung 9.1 gezeigt wird. Diese Datenstruktur enthält u. a. Speicherplatz für die Referenznummern des ersten bis dritten Nachbaratoms (`nBase`,`nNb2` und `nNb3`), so wie die Wechselwirkungsparameter (`dR12`,`dAng123`,`dAng1234` und `nPeriod`) für die Wechselwirkung mit den Nachbaratomen und eine Liste (`anBond`) der Atome, mit denen das betreffende Atom weitere Bindungen eingeht.

Die `TZMat`-Datenstruktur eines Atoms wird mit Hilfe einer der beiden Metho-

den `SetZMat(double dR12, short nBase, double dAng123, short nNb2, short nPeriod, short nNb3)` und `SetZMat(double dR12, short nBase, double dAng123, short nNb2, double dAng1234, short nNb3)` mit Daten gefüllt, sie speichern die als Parameter übergebenen Daten in die Feldelemente des letzten Atoms. Beide Methoden unterscheiden sich in der Behandlung der Torsionswechselwirkung. In der einen Methode wird der Torsionswinkel direkt angegeben, wohingegen im anderen Fall entweder die ausgewählte Periode den Winkel bestimmt oder durch Angabe spezieller Werte eine Berechnung in der Klasse `CCreateMol` erfolgt, die im nächsten Abschnitt beschrieben wird. Ebenfalls wird hier berechnet, ob das neue Atom Teil des Polymergerüstes eines Polymers ist. Die in `TZMat` gespeicherten Parameter werden bei bekannter Position in den Listen mit den entsprechenden Methoden (`GetBase(WORD nAtom)`, `GetNb2(WORD nAtom)`, `GetNb3(WORD nAtom)`, `GetR(WORD nAtom)`, `GetAng123(WORD nAtom)`, `GetAng1234(WORD nAtom)`, `GetAngPeriod(WORD nAtom)` und `IsBackbone(WORD nAtom)`) wieder ausgelesen.

In `SetZMat` werden die Einträge der Liste `anBond` der `TZMat`-Datenstruktur nicht berücksichtigt, diese werden von einer speziell für die Bindungen bestimmten Methode verwendet. Die Methode `SetBond(char nNo,short nAtom)` trägt in das durch `nNo` bestimmte Feldelement des letzten Atoms aus `anBond` eine Referenz auf das im Parameter `nAtom` festgelegte Atom ein. Diese kann später mit der Methode `GetBond(WORD nAtom,char nNo)` wieder ausgelesen werden.

Mit Hilfe der eben beschriebenen Methoden ist es der Methode `Load(CSimSystem *pSys,FILE *fIn,int nMolecule)` möglich, die Strukturdaten eines Moleküls aus der Struktur-Datenbank auszulesen und in den entsprechenden Attributen zu speichern. Die Methode sucht hierfür in der Struktur-Datenbank in Abhängigkeit vom Parameter `nMolecule` nach einem Abschnitt „Molecule“„Molecule1“„Molecule2“ usw., der die gesuchte Molekülstruktur enthält. In jeder Zeile eines Abschnitts mit Molekülstruktur kann entweder ein Eintrag für ein neues Atom stehen oder mit der Anweisung **Name=** der Name des Moleküls. Enthält eine Zeile die Daten eines neuen Atoms, so sind diese wie im Abschnitt 3.3 beschrieben angeordnet. Zur Auswertung wird die Zeile in Spalten aufgeteilt. So enthält die erste oder zweite Spalte den Namen des Atoms, der mit Hilfe der Methode `GetElement` der Klasse `CSimSystem` eine Referenz auf die entsprechende Instanz der Klasse `CElement` bestimmt, wobei diese Instanz unter Umständen zuvor von `GetElement` neu erzeugt werden musste. Ist anstatt eines Atomnamens ein Stern (*) eingetragen, handelt es sich um einen Platzhalter für ein Atom eines anderen Moleküls, genauer eines anderen Monomers, mit dem dieses eine Bindung eingehen wird. Für den Platzhalter wird die entsprechende Referenz auf den Null-Zeiger gesetzt und der Zähler `m_cnDummyAtom` um eins erhöht. Aus der Referenz auf die Instanz und aus der Ladung, die in der nächsten Spalte angegeben wird, erzeugt die Methode `AddAtom` ein neues Atom. Die nächsten Spalten bestimmen die Nachbaratome, so wie optional deren Wechselwirkungen, hierzu werden deren Inhalte in numerische Werte umgewandelt.

Damit auch die Parameter des Torsionswinkels als numerische Werte dargestellt werden können, wird bei der Periodizität ein Stern in einen Wert von -1 und ein „M“ in eine -2 umgewandelt. Diese Werte gehen dann in die Methode `SetZMat` ein. Besitzt ein Atom außer der Bindung zu seinem „Basisatom“ weitere Bindungen, so werden diese als letztes angegeben und mit der Methode `SetBond` wie oben beschrieben im Attribut `anBond` gespeichert. Falls während der Verarbeitung keine Fehler aufgetreten sind, quittiert `Load` dies mit einem logischen Wahr im Rückgabewert.

Für die Erzeugung eines Polymers in `CCreateMol` werden zwei weitere Methoden benötigt. Die Methode `CountDummy()` liefert die Anzahl der Platzhalter-Atome. Aus der Gesamtanzahl der Einträge und der Anzahl der Platzhalter-Atome lässt sich die Anzahl der „wirklichen“ Atome bestimmen, diesen Wert gibt die zweite Methode `CountAtoms()` zurück.

9.3.4 CCreateMol

Nachdem in den vorhergehenden Abschnitten die Klassen behandelt wurden, die den Rahmen für die Strukturerzeugung bilden, wird in diesem Abschnitt die Erzeugung der Startstruktur beschrieben. `CCreateMol` bildet somit den Kern des Programms „StructC“.

Zur Erzeugung der Startstruktur wird im Hauptprogramm eine Instanz von `CCreateMol` generiert, hierzu muss eine Referenz auf die zuvor erzeugte Instanz des zugehörigen Simulationssystems, d. h. der Klasse `CSimSystemStruct` übergeben werden. Im Konstruktor wird diese Referenz für die weitere Verwendung in den Methoden der Klasse im Attribut `m_pSimSystem` gespeichert. Die Klasse `CCreateMol` enthält außerdem eine Referenz auf die Instanz des zu erzeugenden Moleküls, diese wird anfangs auf den Null-Zeiger gesetzt, da das Molekül erst später in der Methode `Create` erzeugt wird. In der Initialisierung wird dem Attribut `m_szPolymerName` der Standardname „POLY“ (für Polymer) zugewiesen.

Die zur Konstruktion notwendigen Informationen werden aus der Struktur-Datei mit Hilfe der in der Klasse implementierten Methode `Load(FILE *fIn)` eingelesen. Diese ruft als erstes die Methode `LoadStruct(FILE *fIn)` auf, die Instanzen für alle in der Struktur-Datenbank aufgeführten Moleküle erzeugt. Hierbei werden die Daten der erweiterten Z-Matrizen mit Hilfe der Methode `Load` der Klasse `CStructMol` eingelesen und in diese im Attribut `m_aStructMol` gespeicherten Instanzen eingetragen.

Die Randparameter der Startstruktur sind im Abschnitt „Structure“ gespeichert, aus dem filtert die `Load`-Methode den Eintrag „Seed“, der den Startwert für den Zufallsgenerator festlegt. Die auch in diesem Abschnitt mit „Include“ eingebundenen anderen Datenbanken werden nur von der Methode `LoadStruct(FILE *fIn)` durchsucht, da `Load` per Definition nur die Abschnitte „Structure“ und „Create“ der Hauptdatenbank berücksichtigt.

Die Startstruktur wird erzeugt durch Einträge im Abschnitt „Create“, dieser

wird aus Gründen der Effizienz zuerst bis auf Kommentare und Leerzeilen eingelesen und zeilenweise in ein Feld vom Datentyp `CMem<CParseScriptline*>` abgelegt. Die Auswertung der Zeilen erfolgt in einer der für die Erzeugung der Struktur wichtigsten Methoden, in `ScanLines(const CMem<CParseScriptline*> &mppsLine,WORD nLine)`, deren Quellcode Abbildung 9.5 zeigt. In den Zeilen 3 und 4 des Quellcodes werden Vorarbeiten zur Auswertung der ausgewählten Textzeile `nLine` durchgeführt. Dort wird zur Vereinfachung der Programmierung eine Referenz auf die auszuwertende Textzeile in der Variablen `psCur` erstellt, in Zeile 4 wird dann überprüft, ob diese Textzeile leer ist. Die zu bearbeitende Textzeile wird zur Kontrolle auf dem Bildschirm ausgegeben. In Zeile 7 wird getestet, ob `psCur` das Schlüsselwort „Density“ enthält, in diesem Fall werden die dann benötigten Werte und Atome im Kode bis zur Zeile 23 bestimmt und erzeugt. Hierbei erfolgt die Bestimmung der gewünschten Dichte und der Molekülsorte, mit deren Molekülen der Simulationsraum aufgefüllt werden soll, in den Zeilen 9 bis 11 durch Verwendung der Methode `FindStruct`. Erzeugt werden die benötigten Moleküle in der Schleife zwischen Zeile 13 und 20, hier sind nur einfache als Lösungsmittel dienende Moleküle zulässig. Erzeugt wird ein Molekül, in dem zuerst mit der Methode `Create` (Zeile 15) eine Struktur generiert wird. Dessen Atome werden im Simulationsraum entsprechend der Struktur durch die Methode `CreateAtoms` (Zeile 16) platziert, und schließlich mit der Methode `Finish()` (Zeile 18) in die Liste der Moleküle in der Instanz von `CSimSystemStruct` aufgenommen. Zur Kontrolle wird die Anzahl der Erzeugten Moleküle vor dem Verlassen der Methode ausgegeben.

Enthält die Variable `psCur` das Schlüsselwort „Polymer“, wird in den Zeilen 24 bis 30 dem Attribut `m_szPolymerName` der Name des neuen Polymers zugewiesen. Zur Vervollständigung der Auswertung wird in Zeile 29 die nächste Textzeile aus `mLine` durch Rekursion ausgewertet.

In `psCur` sollte, nach dem alle anderen Fälle abgearbeitet sind, nur noch der Name eines Moleküls stehen. Die Struktur wird in Zeile 31 wieder mit Hilfe der Methode `FindStruct` in der Strukturliste gesucht und ausgewählt. Enthält die Anweisungszeile zusätzlich Parameter in einem geklammerten Ausdruck (wird in Zeile 34 überprüft), handelt es sich bei dem Molekül um ein Monomer. Dieses bildet den Anfang eines Polymers mit Bindungen zu weiteren Monomeren.

Die Bildung von Polymeren erfolgt in den Zeilen 34 bis 45. Da sich ein Polymer in der Bildungsphase so stark verschlingen kann, dass es nicht zu Ende konstruiert werden kann, besitzt diese Methode in den Zeilen 37 bis 42 eine Schleife, die es erlaubt das Polymer in bis zu `POLY_TRY=10000` Versuchen zu konstruieren. Die Konstruktion erfolgt in der für die Erzeugung von Polymeren zuständigen Methode `ScanLinesPoly`, der in Zeile 41 die Anweisungszeilen `mppLine`, die Startzeile `nLineTmp` und das bereits identifizierte Monomer `nStruct` übergeben wird.

Nach Generierung des Polymers wird mit der nächsten noch nicht ausgewerteten Anweisungszeile fortgefahren (Zeile 43, 58). Handelt es sich bei der erkannten Struktur um ein einfaches Molekül, so bestimmt ein weiterer optionale Parameter

```
BOOL CCreateMol::ScanLines(const CMem<CParseScriptline*> &mppsLine, WORD nLine)
{
  const CParseScriptline &psCur=*mppsLine[nLine];
  if(psCur.IsEmpty()) return TRUE;
  printf("Scan:%s\n",psCur.GetLine());
  char nStruct;
  if(psCur.IsEqual("Density"))
  {
    double dDensity=psCur.GetDouble(1);
    nStruct=FindStruct(psCur.Get(2));
    if(nStruct<0) ThrowException("Molecule not found (%s)",psCur.Get(2));
    WORD cnMolecules=0;
    while(m_pSimSystem->Density()<dDensity)
    {
      Create(m_aStructMol[nStruct].GetName());
      if(!CreateAtoms(nStruct,NULL))
        ThrowException("PosAtom impossible");//can't place molecule within 8 atempts
      Finish();
      ++cnMolecules;
    }//while
    printf("%d %s created.\n",cnMolecules,m_aStructMol[nStruct].GetName());
    return TRUE;
  }//if
  if(psCur.IsEqual("Polymer"))
  {
    free(m_pchPolymerName);
    m_pchPolymerName=strdup(psCur.Get(1));
    ++nLine;
    return ScanLines(mppsLine,nLine);
  }//if
  nStruct=FindStruct(psCur.Get(0));
  if(nStruct<0) ThrowException("Molecule not found (%s)",psCur.GetLine());
  //create
  if(psCur.HasParenthesis())
  {
    WORD nLineTmp;
    for(WORD nRetry=0;nRetry<POLY_TRY;++nRetry)
    {
      TRACE1(10,"Try:%d\n",nRetry);
      nLineTmp=nLine;
      if(ScanLinesPoly(mppsLine,nLineTmp,nStruct)) break;
    }//for
    nLine=nLineTmp;
    if(nRetry>=POLY_TRY) ThrowException("ScanLinePoly error");
  } else {//simple molecule
    WORD nCnt=1;
    if(psCur.IsParam(1)) nCnt=(WORD)psCur.GetInt(1);
    if(nCnt<=0) nCnt=1;
    for(WORD j=0;j<nCnt;++j)
    {
      Create(m_aStructMol[nStruct].GetName());
      if(!CreateAtoms(nStruct,NULL))
        ThrowException("PosAtom impossible");//can't place molecule within 8 atempts
      Finish();
    }//for
    ++nLine;
  }//else
  return ScanLines(mppsLine,nLine);
}
```

Abbildung 9.5: Quellcode der Methode `ScanLines`.

```
#define MAXBASE   100
BOOL CCreateMol::ScanLinesPoly(const CMem<CParseScriptline*> &mppsLine, WORD &nLine,char nBStruct)
{
  const CParseScriptline &psCur=*mppsLine[nLine];
  if(psCur.IsEmpty()) return TRUE;
  Create(m_pchPolymerName);
  printf("ScanPoly Creates :%s\nStart:%s\n",m_pchPolymerName,psCur.GetLine());
  if(!CreateAtoms(nBStruct,NULL)) return FALSE;
  CAtom *aatBase[MAXBASE];
  {for(char j=0;j<MAXBASE;++j) aatBase[j]=NULL;}
  for(char cnParam=0;psCur.IsParam(cnParam+1);++cnParam)
  {
    int nFunUnit=psCur.GetInt(cnParam+1);
    if(nFunUnit<0 || nFunUnit>MAXBASE-1) ThrowException("CCreateMol: range error");
    if(cnParam>=m_aStructMol[nBStruct].CountDummy())
      ThrowException("CCreateMol: too many parameters");
    aatBase[nFunUnit]=GetBaseAtom(nBStruct,cnParam);
  }//for
  //next lines
```

Abbildung 9.6: Quellcode der Methode `ScanLinesPoly` (Teil 1).

die Anzahl der von ihr zu erzeugenden Moleküle. Diese Anzahl wird in Zeile 47 bestimmt und in `nCnt` gespeichert. Zu beachten ist, dass mindestens ein Molekül erzeugt wird, auch wenn kein Wert (`nCnt`=0) angegeben wurde (Zeile 48). Die Erzeugung der Moleküle in den Zeilen 51 bis 54 erfolgt, analog zu der oben für „Density“ beschriebenen Erzeugung einfacher Moleküle, durch Aufruf der drei Methoden `Create`, `CreateAtoms` und `Finish()`. `ScanLines` wird durch einen rekursiven Aufruf mit der nächsten Anwendungszeile als Parameter beendet.

Die Erzeugung der Polymerstruktur wird von der oben erwähnten Methode `ScanLinesPoly(const CMem<CParseScriptline*> &mppsLine, WORD &nLine,char nBStruct)` übernommen, ihr wird eine Liste der Anweisungszeilen, die aktuelle Anweisungszeile und eine Referenznummer auf die Struktur des ersten Monomers übergeben. Da diese Methode die hier sehr wichtigen Polymerstrukturen erzeugt, ist deren Quellcode, der sich in zwei abgeschlossene Teile aufteilen lässt, in den Abbildungen 9.6, 9.7 und 9.8 abgedruckt.

Der erste Teil in Abbildung 9.6 beschreibt die Erzeugung des Moleküls und die Positionierung der Atome des ersten Monomers. In den Zeilen 4 und 5 wird wie in `ScanLines` die Startposition der aktuellen Anweisungszeile bestimmt. Unter dem in `m_szPolymerName` gespeicherten Polymernamen wird in Zeile 6 ein Molekülrahmen für das Polymer erzeugt. Die ersten Atome des Polymers werden in Zeile 8 aus dem bereits in `ScanLines` bestimmten ersten Monomer gebildet.

Die funktionellen Einheiten der Monomere bilden das Bindeglied zwischen den Monomeren. Um bei der Polymerbildung einfacher auf diese funktionellen Einheiten zugreifen zu können, werden Referenzen auf die zugehörigen Atome im Feld `aatBase[MAXBASE]` gespeichert. Später werden ihnen im Konstruktionsskript Nummern zugeteilt, die der Position der Referenzen der zugehörigen Atome im Feld `aatBase` entsprechen. Von den in `aatBase` vorhandenen Feldelementen

wird nur ein Teil benötigt. Zur einfacheren Handhabung werden deshalb in Zeile 10 zunächst alle Feldelemente als unbenutzt deklariert, in dem ihnen der Null-Zeiger zugewiesen wird. In den Zeilen 11 bis 18 werden dann für dieses Monomer die funktionellen Einheiten bestimmt (Zeile 13) und die Referenzen auf die zugehörigen Atome in die entsprechenden Feldelemente von `aatBase` eingetragen. Hierzu wird in den Zeilen 15 und 16 zunächst überprüft, ob für den gewählten Parameter ein Platzhalter im Monomer existiert, für den dann in Zeile 17 mit der Methode `GetBaseAtom` die Referenz auf das Atom bestimmt wird.

Im zweiten Teil der Methode `ScanLinesPoly` geht es um den Aufbau des restlichen Polymers aus Monomeren, dieser musste auf Grund seiner Länge in die Abbildungen 9.7 und 9.8 aufgespalten werden. Der Teil in Abbildung 9.7 befasst sich hauptsächlich mit dem Einfügen einer festen Anzahl von Monomeren mit zufälliger Orientierung. Wie in Zeile 20 zu sehen ist, ist der Rest der Methode bis Zeile 135 in eine Schleife eingefasst, in der die restlichen Anweisungszeilen ausgewertet werden. Zur weiteren Auswertung wird in Zeile 22 die aktuelle Anweisungszeile ausgewählt. Enthält diese keine Einträge, ist die Konstruktion des Polymers beendet und die Schleife kann verlassen werden (Zeile 23).

Steht in der aktuellen Anweisungszeile das Schlüsselwort „add“, so werden Monomere an die selektierte funktionelle Einheit angefügt (Zeile 25 bis 57). In diesem Fall enthält die Anweisungszeile im ersten Parameter die funktionelle Einheit, die in Zeile 27 bestimmt wird. Von dem Kommando „add“ existieren zwei Varianten, eine mit zwei und eine mit drei Parametern. Hat „add“ drei Parameter, so wird ein einfaches bifunktionales Monomer mehrfach angefügt. Die Struktur des Monomers, d. h. der Name des Monomers in der Struktur-Datenbank, wird hierfür in Zeile 30 aus dem zweiten Parameter bestimmt. Der dritte Parameter bestimmt die Anzahl der einzufügenden Monomere. Diese werden in den Zeilen 33 bis 37 erzeugt und verteilt, hierbei werden dessen Referenz auf das Atom mit der funktionalen Einheit in Zeile 36 bei jeder Erweiterung neu bestimmt.

Hat „add“ nur zwei Parameter, enthält der Zweite ein multifunktionales Monomer inklusive einer Liste der, den funktionalen Einheiten zugeordneten, Nummern. Die Auswertung dieses Parameters erfolgt in einer eigenen Instanz der Klasse `CParseScriptline`, d. h. dieser Parameter wird in eine Liste von Unterparametern aufgespalten. Aus dem ersten Eintrag wird in Zeile 43, wie im ersten Fall, mit Hilfe der Methode `FindStruct` die Struktur des anzufügenden Monomers bestimmt. Diese wird verwendet, um in der nächsten Zeile die benötigten Atome zu erzeugen und sie damit an das bestehende Polymer anzufügen. Das neue Monomer belegt die funktionale Einheit an die es angefügt wurde, deshalb wird diese Einheit in Zeile 45 frei gegeben. Aus den weiteren Parametern werden in den Zeilen 46 bis 54, analog zu den Zeilen 11 bis 18 aus Abbildung 9.6, den Atomen die neuen funktionalen Einheiten zugeordnet.

Der zweite in `ScanLinesPoly` implementierte Befehl („crw“, Zeile 59ff) verbindet zwei funktionale Einheiten durch eine Kette bifunktionaler Monomere. Bezogen auf das P-NIPAM werden hiermit die Polymerketten zwischen den

```
  //next lines
  for(;;)
  {
    const CParseScriptline &psCur=*mppsLine[++nLine];
    if(psCur.IsEmpty()) break;
    printf("ScanPoly:%s\n",psCur.GetLine());
    if(psCur.IsEqual("add"))
    {
      int nBase=psCur.GetInt(1);
      if(psCur.IsParam(3))//add 1 Monomer=5
      {
        char nStruct=FindStruct(psCur.Get(2));
        WORD nCnt=(WORD)psCur.GetInt(3);
        if(nCnt<=0) nCnt=1;
        for(WORD j=0;j<nCnt;++j)
        {
          if(!CreateAtoms(nStruct,aatBase[nBase])) return FALSE;
          aatBase[nBase]=GetBaseAtom(nStruct,0);
        }//for
      }else{//add 1 Monomer(2,3)
        CParseScriptline ps;
        ps.Parse(psCur.Get(2));
        if(!ps.HasParenthesis())
          ThrowException("Input error: no '()' (%s)",psCur.GetLine());
        char nStruct=FindStruct(ps.Get(0));
        if(!CreateAtoms(nStruct,aatBase[nBase])) return false;
        aatBase[nBase]=NULL;
        for(char cnParam=0;ps.IsParam(cnParam+1);++cnParam)
        {
          int nFunUnit=ps.GetInt(cnParam+1);
          if(nFunUnit<0 || nFunUnit>MAXBASE-1)
            ThrowException("CCreateMol: range error");
          if(cnParam>=m_aStructMol[nBStruct].CountDummy())
            ThrowException("CCreateMol: too many parameters");
          aatBase[nFunUnit]=GetBaseAtom(nBStruct,cnParam);
        }//for
      }//else
      continue;//next line
    }//if

    if(psCur.IsEqual("crw"))
    {
      float dProb=(float)psCur.GetDouble(1);
      int nBase1=psCur.GetInt(2);
      int nBase2=psCur.GetInt(3);
      static const char *szValueError=
        "CCreateMol(crw error): Parameter %d Value (%s) not found.";
      if(aatBase[nBase1]==NULL) ThrowException(szValueError,2,psCur.GetLine());
      if(aatBase[nBase2]==NULL) ThrowException(szValueError,3,psCur.GetLine());
      char nStruct=FindStruct(psCur.Get(4));
      //nach den regulaeren atomen kommen die dummy (*) atome
      int nBaseAtom=m_aStructMol[nStruct].GetBase(m_aStructMol[nStruct].CountAtoms());
      if(nBaseAtom<0)
        ThrowException("CCreateMol: no Base for %s",m_aStructMol[nStruct].GetName());
```

Abbildung 9.7: Quellcode der Methode `ScanLinesPoly` (Teil 2).

Bisacrylamid-Quervernetzer-Monomeren gebildet. Der erste Parameter, der in Zeile 61 ausgewertet wird, enthält die Wahrscheinlichkeit für den gerichteten „random walk“ auf das Ziel, die andere funktionale Einheit, zuzusteuern. D. h. es legt fest, mit welcher Wahrscheinlichkeit die beiden Enden aufeinander zuwandern bzw. mit welcher Wahrscheinlichkeit sie in eine zufällige Richtung wachsen. Somit führt ein kleiner Wert zu langen Polymerketten, wohingegen ein gerichtetes Wachstum, ausgedrückt durch eine hohe Wahrscheinlichkeit, zu kurzen Polymerketten führt. Die Parameter zwei und drei, bestimmt in den Zeilen 62 und 63, legen jeweils eine der beiden funktionalen Einheiten fest, zwischen denen die Polymerkette konstruiert wird. Hierzu wird in den Zeilen 64 bis 67 überprüft, ob die angegebenen funktionalen Einheiten vorhanden sind, bevor in Zeile 68 aus dem vierten Parameter der Name der Struktur des bifunktionalen Monomers ausgelesen wird. Als nächstes wird in den Zeilen 70 bis 72 aus der eben bestimmten Struktur das Atom gesucht, mit dem der Aufbau des Monomers begonnen werden kann.

Nach diesen Vorarbeiten erfolgt die Konstruktion der Polymerkette mit dem in Abbildung 9.8 gezeigten Quellcode. Für deren Konstruktion werden unter Umständen mehrere Anläufe benötigt, da sich die Teile des Polymers sterisch so behindern können, dass das Polymer in der angefangenen Form nicht fertiggestellt werden kann. Damit die eingefügten Atome für einen weiteren Versuch wieder entfernt werden können, wird in Zeile 74 eine Referenz auf das letzte vor der Erweiterung bereits bestehende Atom in der Variablen `patLastOK` gespeichert. Für die einzelnen Durchläufe wird die Kettenkonstruktion in den Zeilen 76 bis 130 von einer Schleife umschlossen. Innerhalb dieser Schleife werden zuerst die letzten Atome der beiden offenen Enden bestimmt und in `patBase1` und `patBase2` gespeichert. Zu Beginn der Konstruktion sind dies die Atome der beiden funktionellen Einheiten (Zeile 20 / 21). Zunächst werden alle Atome aus einem eventuell vorangegangenen Schleifendurchlauf in den Zeilen 82 bis 85 mit Hilfe der durch `patLastOK` festgelegten Grenze gelöscht. Zur Konstruktion der Kette müssen noch einige Variablen initialisiert werden. Die Variable `nGrowNow` legt fest an welchem Ende (`patBase1` oder `patBase2`) das nächste Monomer angefügt werden soll, begonnen wird hier stets mit `patBase1`. Außerdem werden in den Zeilen 88 und 89 für die Abbruchbedingungen, die Variable für die Toleranz des Abstands `dTol` und der Zähler der Fehlversuche `nStop` initialisiert.

Innerhalb der Schleife zwischen Zeile 91 und 117 wird zuerst mit Hilfe der zuvor initialisierten Variablen `nGrowNow` entschieden an welchem der beiden Enden ein neues Monomer angefügt werden soll. Anschließend wird versucht das Monomer an dem ausgewählten Ende anzufügen. Gelingt dies nicht, wird der Zähler `nStop` erhöht, dann wird versucht ein Monomer am anderen Ende anzubringen. Nach drei Fehlversuchen bricht die Konstruktion der Kette ab, es beginnt ein neuer Versuch das Monomer an das bestehende Polymer anzuhängen. Im ersten Schleifendurchlauf wird in Zeile 98 bis 103 die Länge eines Monomers der Variablen `dTargetDist` zugewiesen. Sie wird bestimmt, indem der Abstand zwischen

```
      //try new construction
      CAtom *patLastOK=GetMolecule()->GetAtom(GetMolecule()->CountAtoms()-1);
      double dTargetDist=-1;
      for(char nTry=CHAIN_TRY;nTry>0;--nTry)
      {
        CAtom *patBase1,*patBase2;
        patBase1=aatBase[nBase1];
        patBase2=aatBase[nBase2];
        //remove atoms from last try
        while(patLastOK!=GetMolecule()->GetAtom(GetMolecule()->CountAtoms()-1))
        {
          GetMolecule()->DelLastAtom();
        }//while
        char nGrowNow=1;//select ending 1 or 2
        CVec3 vDist;
        char nStop=0;
        double dTol=m_pSimSystem->GetCrwTol();
        if(dTol<0) dTol=0;
        do {
          if(nGrowNow==1)
          {
            CVec3 v(patBase2->GetPos());
            if(CreateAtoms(nStruct,patBase1,&v,dProb))
            {
              int nAtomNo=GetMolecule()->CountAtoms()-m_aStructMol[nStruct].CountAtoms()+nBaseAtom;
              if(dTargetDist<0)
              {
                CVec3 v;
                pbx.PBCSub(v,patBase1->GetPos(),GetMolecule()->GetAtom(nAtomNo-nBaseAtom)->GetPos());
                dTargetDist=v.Abs();
              }//if
              patBase1=GetMolecule()->GetAtom(nAtomNo);
            } else ++nStop;
            nGrowNow=2;
          } else {
            CVec3 v(patBase1->GetPos());
            if(CreateAtoms(nStruct,patBase2,&v,dProb))
            {
              int nAtomNo=GetMolecule()->CountAtoms()-m_aStructMol[nStruct].CountAtoms()+nBaseAtom;
              patBase2=GetMolecule()->GetAtom(nAtomNo);
            } else ++nStop;
            nGrowNow=1;
          }//else
          pbx.PBCSub(vDist,patBase1->GetPos(),patBase2->GetPos());
        } while (fabs((vDist.Abs()-dTargetDist)/dTargetDist)>dTol && nStop<=3);
        if(nStop<=3)
        {
          //connect chains
          GetMolecule()->SetNeighbour(GetMolecule()->GetNo(patBase1),GetMolecule()->GetNo(patBase2));
          aatBase[nBase1]=aatBase[nBase2]=NULL;
          {//output of chain distance
            CVec3 vDist;
            pbx.PBCSub(vDist,patBase1->GetPos(),patBase2->GetPos());
            printf("crw Dist:%f\n",vDist.Abs());
          }//{
          break;
        }//if
      }//for
      if(nTry<=0) return FALSE;
      continue;
    }//if
    break;//no poly command, end of sequence
  }//for
  Finish();
  return TRUE;
}
```

Abbildung 9.8: Quellcode der Methode `ScanLinesPoly` (Teil 3).

dem ursprünglichen Atom mit der funktionalen Einheit und dem Atom das künftig die funktionale Einheit berechnet wird. Die Verschiebung der funktionalen Einheit auf das Atom des neuem Monomers erfolgt in Zeile 77 bzw. 111, indem die Nummer dieses Atoms in der Atomliste bestimmt wird und in Zeile 104 bzw. 112 dessen Referenz in `patBase1` bzw. `patBase2` gespeichert wird.

Zur Überprüfung des Abstandes der beiden formalen Einheiten wird der Abstand der zugehörigen Atome in Zeile 116 ermittelt. Die Schleife des Bildungsprozesses wird beendet, wenn es nicht möglich war innerhalb von drei Versuchen ein Atom zu platzieren, oder wenn der oben bestimmte Abstand innerhalb des durch `dTol` festgelegten Toleranzbereiches liegt.

Die Bildung der Polymerkette wird in Zeile 121 durch Verknüpfung der Atome am Ende der Polymerkette und dem Löschen der formalen Einheiten aus der Liste erfolgreich beendet. Am Ende der Konstruktion, wenn das gesamte Polymer im Simulationsraum platziert ist, wird es mit Hilfe der Methode `Finish()` in Zeile 136 in die Liste der Moleküle aus der Instanz der Klasse `CSimSystemStruct` aufgenommen.

In den Methoden `ScanLines` und `ScanLinesPoly` wurden einige spezielle Methoden verwendet, die nun beschrieben werden sollen. Die Methode `FindStruct(const char *pName)` vergleicht, ohne Berücksichtigung der Groß- / Kleinschreibung, in einer Schleife den Suchtext `pName` mit den Namen der in `m_aStructMol` gespeicherten Molekülstrukturen, bis Suchtext und Name der Molekülstruktur identisch sind, um dann dessen Position in der Liste zurückzugeben. Die nur von `ScanLinesPoly` verwendete Methode `GetBaseAtom(char nStruct, char nDummy)` bestimmt aus der Struktur und den Parametern des Moleküls das Atom, an dem die durch `nDummy` festgelegte formale Einheit sitzt. Hierzu wird die Nummer `nBase` des Atoms in der Instanz der Klasse `CStructMol` bestimmt. Aus dieser wird dann die Nummer des entsprechenden Atoms im Molekül berechnet, mit der die zurückgegebene Referenz auf das Atom bestimmt werden kann.

Für die Erstellung und Fertigstellung des Moleküls sind zwei spezielle Methoden vorgesehen. Die Methode `Create(const char pName)` erzeugt ein neues Molekül unter dem in `pName` angegebenen Namen. Sie löscht, falls vorhanden, das alte in `m_pMolecule` gespeicherte Molekül und erzeugt dort eine neu Instanz der von `CMolecule` abgeleiteten Klasse `CMoleculeStruct`. Handelt es sich dabei um die Erstellung des ersten Moleküls, so wird zusätzlich der momentane Wert des Zufallszahlengenerators ausgegeben, dies erleichtert die Reproduzierbarkeit der zu erzeugenden Startstruktur. Mit der Methode `GetMolecule()` kann die Referenz auf das zu erzeugende Molekül jeder Zeit abgerufen werden. Sind sämtliche Atome des Moleküls durch `CreateAtoms` erzeugt und im Simulationsraum verteilt, so muss das Molekül in die Liste der Moleküle der Instanz `m_pSimSystem` der Klasse `CSimSystem` aufgenommen werden. Dies erledigt die Methode `Finish()`, die außerdem das Molekül aus der Strukturerzeugung entfernt, in dem es das Attribut `m_pMolecule` auf den Null-Zeiger setzt.

Die ebenfalls in `ScanLines` und `ScanLinesPoly` verwendete Metho-

de `CreateAtoms(char nStructID,CAtom *patBase, const CVec3 *vTarget, float dProb)` muss, um die Atome der Struktur `nStructID` korrekt im Simulationsraum zu erstellen, drei Teilaufgaben erfüllen. Als erstes muss sie neue Instanzen der Klasse `CAtom` erzeugen und in die Atomliste der Moleküls aufnehmen, wobei die Instanz der Klasse `CElement` und die Partialladung des Atoms durch die Daten in der Strukturliste bestimmt werden. Die Zuordnung zu einer bestimmten Atomgruppe erfolgt mit der Methode `Add` der Klasse `CAtomGroup`. Anschließend muss sie die neu erstellten Atome untereinander und bei Polymeren auch mit den bereits erzeugten Atomen verbinden. Die Klasse `CMolecule` besitzt hierzu Methoden und Attribute, die direkt benachbarte Atome verknüpft, d. h. die Atome verknüpft, die untereinander eine direkte Bindung eingehen. Mit `GetBase(i)` wird aus der Molekül- / Monomerstruktur für ein Atom `i` der Anknüpfungspunkt des Atoms an das Molekül bestimmt. Einen Eintrag in der Nachbarschaftsliste erhalten beide Atome durch einen Aufruf der Methode `SetNeighbour` mit den Nummern beider Atome als Parameter. Einen Sonderfall bildet hierbei die Anknüpfung des ersten Atoms des Monomers an das Atom mit der Referenz `patBase` eines bestehenden Polymerstranges. Dieses Atom wird in der Struktur durch einen Rückgabewert von -1 bei der Methode `GetBase` gekennzeichnet. Ebenfalls in die Nachbarschaftsliste gehen die in der Z-Matrix explizit angegebenen (z. B. in Abbildung 3.4 in Zeile 9) direkten Nachbarn ein. Mit Hilfe der Nachbarschaftsliste und bei Berücksichtigung der intramolekularen Wechselwirkungen können die Atome durch die Methode `PosAtoms` im Simulationsraum verteilt werden. Ist es, z. B. durch sterische Hinderung nicht möglich alle Atome unter Berücksichtigung der intramolekularen Wechselwirkungen und den Parametern der Z-Matrix, die in den Strukturdaten verborgen sind, im Simulationsraum zu verteilen, werden alle neu erzeugten Atome von `PosAtoms` wieder gelöscht und ein entsprechender Statuswert an die aufrufende Methode zurückgegeben. Um dennoch eine Struktur aufzubauen, versuchen es `ScanLines` und `ScanLinesPoly` bis zu 20 mal ein Monomer anzufügen bzw. ein Molekül zu erzeugen.

Die Atome werden im Simulationsraum in der Reihenfolge ihres auftreten in der Z-Matrix platziert. Wird ein Monomer an ein bestehendes Teilpolymer angefügt, so sind bei der Platzierung der ersten Atome dieses Monomers die benachbarten Atome des Teilpolymers und deren intramolekulare Wechselwirkungen zu berücksichtigen. Die Freiheitsgrade für die Platzierung eines Atoms nehmen mit jedem weiter zu berücksichtigendem Atom zu, bis sie ab drei Atomen soweit eingeschränkt sind, dass nur noch der Torsionswinkel in engen Grenzen frei wählbar ist. Die Z-Matrix bestimmt im Allgemeinen den Torsionswinkel bzw. eine Periode im Torsionswinkel. Dieser wird durch Angabe eines Winkels fest vorgegeben, durch den gerichteten Aufbau im Polymer bestimmt oder zufällig vom Programm ausgewählt. Für die Platzierung der Atome wird in Abhängigkeit von diesen drei Wahlmöglichkeiten die entsprechende `PosAtomZMat4`-Methode aufgerufen. Bei den ersten Atomen eines Moleküls werden die nicht vorhandenen Nachbaratome durch Referenzen auf den Null-Zeiger ersetzt, hierdurch wird dann auch

die Berechnung der dadurch fehlenden intramolekularen Wechselwirkungen übersprungen. Der Rückgabewert der Methoden `PosAtomZMat4` legt fest, ob das Atom platziert werden konnte. Ist dies nicht der Fall, werden die Atome des Moleküls durch einen erneuten Aufruf von `PosAtoms` neu verteilt.

Die Positionierung eines Atoms erfolgt immer über den Aufruf einer der `PosAtomZMat4`-Methoden. Diese ruft, bei weniger als drei bereits erzeugten Atomen, die entsprechende spezialisierte Methode auf. Die Positionierung des ersten Atoms erfolgt problemlos über die Methode `PosAtomZMat1(CAtom *pat1)`. Hierzu wird ein Punkt im Simulationsraum durch Erzeugung eines Zufallsverktors bestimmt. Die Platzierung erfolgt daraufhin an diesem Punkt durch Aufruf der Methode `PosAtomZMat1(CAtom *pat1, CVec3 vPos)`. Ist das Atom dort nicht platzierbar, wird dies bis zu zehn mal an unterschiedlichen Punkten wiederholt.

In der Methode `PosAtomZMat1(CAtom *pat1, CVec3 vPos)` wird das Atom zunächst temporär an die in `vPos` angegebene Stelle platziert, um zu überprüfen, ob das Atom zu bereits vorhandenen Atomen einen genügenden „Sicherheitsabstand“ besitzt. Dessen Betrag wird durch die Lenard-Jones Parameter festgelegt. Zunächst wird durch Aufruf der Methode `IsFreeInMolecule` die Einhaltung des Mindestabstandes zu den Atomen des eigenen Moleküls überprüft. Ist dieser Test erfolgreich verlaufen, wird mit der Methode `IsFreePos` der Klasse `CSimSystem` der Abstand zu den Atomen der anderen Moleküle getestet. Das Ergebnis dieses Tests wird dann an die aufrufende Methode zurückgegeben.

Geht man in der Aufrufkette einen Schritt zurück, gelangt man zur Methode `PosAtomZMat2(CAtom *pat1, double dR12, const CAtom *pat2)`. Diese bestimmt im Allgemeinen die Lage des zweiten Atoms. Nur falls der Parameter `pat2` eine Referenz auf den Null-Zeiger enthält, wird die oben beschriebene Methode `PosAtomZMat1(CAtom *pat1)` aufgerufen. Zur Festlegung der Position des zweiten Atoms wird in einer Schleife ein zufälliger Richtungsvektor erzeugt, der als letzter Parameter in den Aufruf der Methode `PosAtomZMat2(CAtom *pat1, double dR12, const CAtom &at2, const CVec3 &vAxe12)` eingeht. Die Schleife sorgt dafür, dass bei einem fehlgeschlagenen Versuch, bis zu neun weitere Versuche unternommen werden, um das zweite Atom zu platzieren, bevor die Methode abbricht und einen Fehlerstatus zurück an die aufrufende Methode gibt. Die Methode `PosAtomZMat2(CAtom *pat1, double dR12, const CAtom &at2, const CVec3 &vAxe12)` bestimmt die Lage des Atoms `at2` aus dem Richtungsvektor `vAxe12` und dem Atomabstand `dR12`, bei negativen Abstand, wird die in den Kraftfeld-Parametern stehende Bindungslänge verwendet. Die eigentliche Platzierung erfolgt nach Berechnung der Lage wieder durch die Methode `PosAtomZMat1(CAtom *pat1, CVec3 vPos)`.

Die Methode `PosAtomZMat3(CAtom *pat1, double dR12, const CAtom *pat2, double dAng123, const CAtom *pat3, double dAng)` nutzt zur Platzierung des dritten Atoms eine spezielle Eigenschaft der Methode `PosAtomZMat4(CAtom *pat1, double dR12, const CAtom *pat2, double`

dAng123, const CAtom *pat3, double dAng1234,const CAtom *pat4).
Falls das vierte übergebene Atom mit dem zweiten und dritten Atom keine Ebene bildet, wird das erste Atom in einem zufälligen Drehwinkel um die Achse vom dritten zum zweiten Atom platziert. In PosAtomZMat3 wird hierzu die Methode PosAtomZMat4 mit den Parametern von PosAtomZMat3 aufgerufen. Der in PosAtomZMat3 fehlende Parameter pat4 wird, um die gerade beschriebene Eigenschaft auszunutzen, mit einer Referenz auf das selbe Atom wie Parameter pat2 belegt.

Die Verteilung der Atome eines Moleküls wird von der Methode PosAtoms durchgeführt, hierzu verwendet sie, wie oben bereits erwähnt, die Methoden PosAtomZMat4, da diese durch Aufruf der anderen Methoden die Atome entsprechend den intramolekularen Wechselwirkungen im Simulationsraum verteilen können. Ab dem dritten zu platzierenden Atom greifen drei der Methoden PosAtomZMat4 nach einigen Umrechnungen auf die vierte Methode PosAtomZMat4(CAtom *pat1, double dR12, const CAtom &at2, double dAng123, const CVec3 &v3, double dAng1234, const CVec3 &v4) zu. Diese erzeugt zuerst ein, wie bereits in Abschnitt 3.4 beschriebenes, neues lokales Basissystem. Die Z-Achse des Systems wird hierbei aus dem Vektor von v3 zur Position von Atom at2 gebildet, dieser spannt mit dem Vektor von v3 nach v4 eine Ebene auf, auf der die Y-Achse senkrecht steht, d. h. das Vektorprodukt der Vektoren der Ebene erzeugen einen Vektor entlang der lokalen Y-Achse. Wird durch die lokale Z-Achse und dem Vektor von v3 nach v4 keine Ebene aufgespannt, so wird eine zufällige senkrecht zur Z-Achse stehende Y-Achse erzeugt. Das Vektorprodukt von Z-Achse und Y-Achse liefert dann die X-Achse. Die so erzeugten Achsenvektoren stehen senkrecht zueinander, nach einer Normierung bilden sie die Basisvektoren eines lokalen kartesischen Koordinatensystems im zu erzeugenden Molekül. Aus den Winkeln dAng123 und dAng1234 wird durch Aufruf der Methode SetSphere ein, auch im Abschnitt 3.4 beschriebener, Richtungsvektor erzeugt, der aus dem lokalen kartesischen Koordinatensystem mit der Methode BasisT in das globale, ebenfalls kartesische Koordinatensystem transformiert wird. Dieser Richtungsvektor wird zusammen mit der Referenz auf das zu platzierende Atom, dem direkt bindenden Atom und dem Atomabstand an die Methode PosAtomZMat2 übergeben, die das Atom dann, wie oben beschrieben, über die Methode PosAtomZMat1 platziert.

Ist ein Torsionswinkel explizit vorgegeben, bietet sich zur Platzierung des neuen Atoms die Methode PosAtomZMat4(CAtom *pat1, double dR12, const CAtom *pat2, double dAng123, const CAtom *pat3, double dAng1234, const CAtom *pat4) an. Diese unterscheidet sich von der Vorhergehenden durch die Möglichkeit, den Bindungswinkel dAng123 durch Übergabe eines negativen Wertes auch aus den Kraftfeldparametern bestimmen zu lassen.

Ist anstelle des Torsionswinkels die Periode angegeben, d. h. angegeben in welchem Energieminimum der Torsionswechselwirkung sich der Winkel befindet, wird die Methode PosAtomZMat4(CAtom *pat1, double dR12, const CAtom

`*pat2, double dAng123, const CAtom *pat3, const CAtom *pat4, short nAngPer)` verwendet, ihr wird als letzter Parameter die gewünschte Periode übergeben. Hierbei wird zuerst wieder überprüft, ob das vierte übergebene Atom den Null-Zeiger enthält und die Ortsbestimmung dann mit `PosAtomZMat3` erfolgt. Dann werden die Wechselwirkungsparameter der Torsion für die gewählten Atome aus den Kraftfeldparametern extrahiert. Aus den Wechselwirkungsparametern (m, ϕ_0) und der vorgegebenen Periode `nAngPer` lässt sich der Torsionswinkel ϕ wie folgt berechnen:

$$\phi = \frac{\pi + 2\pi \texttt{ nAngPer} + \phi_0}{m}$$

Aus dem nun bekannten Torsionswinkel wird die Lage des Atoms mit Hilfe der zuvor besprochenen Methode bestimmt. Diese einfache Lösung gilt nur, falls der Wert im Parameter `nAngPer` nicht negativ ist. Negative Werte kennzeichnen Spezialfälle, in denen `nAngPer` spezielle Steuercodes enthält, die in der Datenbank durch Sonderzeichen gekennzeichnet werden. In der `Load`-Methode von `CStructMol` werden diese in entsprechende negative Zahlen umgewandelt (siehe auch Abschnitt 9.3.3, Seite 143). Dem Steuercode -1 entspricht in der Datenbank einem Stern „*“ in der Spalte der Periode. Dieser Eintrag sorgt für eine zufällige Auswahl eines der m Energieminima der Torsionswechselwirkung. Wobei die Periode zufällig gewählt wird und der Ortsvektor des Atoms mit Hilfe der Methode `PosAtomZMat4(CAtom *pat1, double dR12, const CAtom *pat2, double dAng123, const CAtom *pat3, double dAng1234,const CAtom *pat4)` bestimmt wird. Ist der vom neuen Atom beanspruchte Raumbereich bereits durch ein anderes Atom belegt, versucht die Methode durch wiederholte, zufällige Wahl eines Energieminimums einen freien Raumbereich zu finden. Der zweite implementierte Steuercode -2, in der Datenbank durch ein „M“ repräsentiert, wählt zufällig eines der Minima, die zu einer minimalen Wechselwirkung zwischen den intermolekularen Wechselwirkungen (van der Waals und Coulomb) führen. Hierfür werden die intermolekularen Wechselwirkungsenergien auf das neue Atom für alle m Energieminima der Torsionswechselwirkung mit der Methode `Energy(const CAtom *pat)` berechnet, so wie deren minimalstes Minimum ausgewählt. Um nicht kleinste Schwankungen der intermolekularen Wechselwirkung als Unterscheidungskriterium einzubeziehen, werden alle Energien bis zu einem Prozent Abweichung vom Minimalwert als gleichwertig angesehen. In der Methode werden deshalb in einer zweiten Schleife die berechneten Wechselwirkungsenergien mit der Minimalenergie verglichen, wobei eine der erlaubten Perioden über eine zufällige Auswahl selektiert wird. Aus der so gewonnenen Periode wird der Torsionswinkel bestimmt, mit dem die `PosAtomZMat4`-Methode für feste Torsionswinkel dann wieder den Ortsvektor des Atoms berechnet.

Bei der oben verwendeten Methode `Energy(const CAtom *pat)` wird für das Atom `pat` eine einfache Näherung der Wechselwirkungsenergie hergeleitet. In die

Berechnung gehen nur die von anderen Atomen auf `pat` ausgeübten Coulomb-Wechselwirkungen im Vakuum und die van der Waals-Wechselwirkungen ein. Nicht berücksichtigt werden die in den Abschnitten 2.2.1 und 2.2.3 beschriebenen Ausschlusslisten der intermolekularen Wechselwirkungen, jedoch werden die periodischen Randbedingungen berücksichtigt. Berechnet wird zunächst die Wechselwirkungsenergie des Atoms mit den bereits verteilten Atomen des selben Moleküls, zu der dann die Energie der Wechselwirkungen mit den Atomen der anderen Moleküle hinzu kommt.

Mit den bis her eingeführten `PosAtomZMat4`-Methoden ist es nur schwer möglich zwei Teilketten eines Polymers über einen zufälligen Kettenverlauf zu verbinden, hierfür wurde eine weitere Methode geschrieben. Die Methode `PosAtomZMat4(CAtom *pat1, double dR12, const CAtom *pat2, double dAng123, const CAtom *pat3, const CAtom *pat4, const CVec3 &vTarget, float dProb)` wird in `PosAtoms` aufgerufen, wenn ein Polymer durch einen nur teilweise zufälligen „directed random walk“ vergrößert werden soll. Der letzte Parameter `dProb` gibt hier die Wahrscheinlichkeit an, dass das Atom `pat1` nicht in eine zufällig Richtung, sondern in Richtung des Zielpunktes `vTarget`, an das Teilpolymer angebaut wird. Mit Hilfe eines Zufallszahlengenerators wird die Methode entweder durch den Aufruf der entsprechenden `PosAtomZMat4`-Methode beendet, oder ein Ortsvektor in Richtung des Zielpunktes berechnet. Im Fall des gerichteten Aufbaus, wird der geeignetste Ortsvektor durch testen aller m möglichen Energieminima des Torsionswinkels bestimmt. Die Ortsvektoren werden für jeden Wikel, wie oben beschrieben, durch Aufruf der entsprechenden `PosAtomZMat4`-Methode berechnet.

Nach Erstellung des Simulationssystems gibt das Programm „StructC“ eine kurze Zusammenfassung der erzeugten Struktur aus, hierfür wird die hier nicht weiter erläuterte Methode `Report()` dieser Klasse verwendet.

9.4 Die Klassen zur Simulation mit „dynC“

Die mit „StructC“ erzeugten Startsysteme befinden sich, bis auf die kurzreichweitigen Wechselwirkungen, die die primäre Struktur der Moleküle vorgeben, nicht in einem Gleichgewichtszustand. Dieser muss für die langreichweitigen Wechselwirkungen durch Simulation noch erreicht werden. Hierzu verwendet man das Programm „dynC“, es erfüllt zwei Aufgaben. Zum Einen führt es die Startsysteme durch Equilibrierung in den Gleichgewichtszustand und zum Anderen werden durch Simulation die Trajektorien erzeugt, die den Verlauf einer Simulation über die besuchten Punkte im Phasenraum protokollieren.

„dynC“ setzt sich wie „StructC“ aus Klassen zusammen, die die Eigenschaften und möglichen Aktionen der Systeme und deren Komponenten beschreiben. Die verwendeten Klassen werden zum Teil in beiden Programmen verwendet, andere Klassen werden von Klassen aus „StructC“ abgeleitet. Wieder andere Klassen

erfüllen spezielle Aufgaben, die nur zur Simulation benötigt werden. Im folgenden werden die letzten beiden Arten beschrieben, wobei zuerst die von gemeinsamen Vorfahren abgeleiteten Klassen beschrieben werden.

9.4.1 CAtomSim

Mit der Klasse `CAtom` lassen sich zwar die Strukturen der Atome in Simulationsstrukturen verwalten, aber zur Simulation der Atome sind einige Erweiterungen dieser Klasse notwendig. Die Klasse `CAtomSim` wird von der Klasse `CAtom` abgeleitet, um diese zusätzliche Funktionalität zu implementieren.

Das bereits in der ursprünglichen Klasse implementierte Attribut der Geschwindigkeit `m_vVelocityTimeStep` enthält, wegen der einfacheren Handhabbarkeit während der Simulation, die Geschwindigkeit des Atoms multipliziert mit der Schrittweite `m_dTimeStep`. Diese modifizierte „Geschwindigkeit“ wird mit der Methode `SetVTS(const CVec3 &vVTS)` durch die in `vVTS` übergebene Geschwindigkeit ersetzt. Ausgelesen wird sie mit der Methode `VTS()`. Um die Geschwindigkeit um einen festen Beitrag zu erhöhen oder zu verringern, wurde die Methode `AddVTS(const CVec3 &vVTSnew)` implementiert, die die Geschwindigkeit um `vVTSnew` erhöht. Soll die Geschwindigkeit um einen festen Faktor `dS` geändert werden, um zum Beispiel eine neue Schrittweite einzustellen, wird die Methode `ScaleVTS(double dS)` verwendet. Während der Simulation wird die Geschwindigkeit von einem Zeitschritt zum Nächsten durch ein Aufruf der Methode `UpdateVTS()` angepasst. Diese berücksichtigt die in einem Zeitintervall auf das Atom wirkende Kraft bzw. Beschleunigung, die im neuen Attribut `m_vForce` gespeichert wurde bei der Bestimmung der neuen Geschwindigkeit. Das neue Attribut wird mit der Methode `F()` ausgelesen. Um die auf ein Atom wirkende Kraft zu ändern existieren drei Methoden. Die Methode `ResetF()` wird am Anfang der Kraftberechnung eines jeden Iterationsschrittes aufgerufen, um die auf das Atom wirkenden Kräfte auf Null zu setzen. Die von den Wechselwirkungen herrührenden Kräfte wirken auf das Atom entweder additiv oder subtraktiv, entsprechend wird entweder die Methode `AddF(const CVec3 &vFnew)` oder `SubF(const CVec3 &vFnew)` aufgerufen. Die so in `m_vForce` berechnete resultierende Kraft muss zur Bestimmung der neuen Geschwindigkeit in eine Beschleunigung umgerechnet werden. Hierfür wird der in `m_vForce` gespeicherte Wert mit der Methode `RescaleF(const double &dScale)` durch die Masse des Atoms dividiert und mit dem Faktor `dScale` multipliziert. Der Faktor `dScale` enthält hierbei einen sich aus der Newtonschen Bewegungsgleichung ergebenen und von der Schrittweite abhängenden Wert.

Die intermolekularen Wechselwirkungen eines Atoms wirken nicht auf alle anderen Atome. Aufgrund der intramolekularen Wechselwirkungen zwischen Atomen sind die intermolekularen Wechselwirkungen der entsprechenden Atome verringert oder ganz ausgeschlossen. Zwischen Atomen, die eine direkte Bindung eingehen bzw. über einen festen Bindungswinkel verbunden sind, gibt es keine

intermolekularen Wechselwirkungen. Für die entsprechenden Atome in den Instanzen von CAtomSim wird deshalb eine Ausschlussliste angelegt. Diese wird im Attribut `m_apExclude` gespeichert und enthält eine Liste der Atome mit denen keine intermolekulare Wechselwirkung besteht. In die Ausschlussliste aufgenommen wird ein Atom `pAtom` durch Aufruf der Methode `AddExclude(CAtomSim *pAtom)`. Die Methode `GetExclude(BYTE cn)` liest das Atom wieder aus, wobei in `cn` die Position innerhalb der Liste angegeben werden muss. Für die Atome mit Torsionswechselwirkung gibt es häufig im Lennard-Jones-Parametersatz spezielle Wechselwirkungsparameter, deshalb werden auch diese zusätzlich in einer gesonderten Liste (`m_apThirdNb`) verzeichnet, auf die ebenfalls mit entsprechenden Methoden (Abbildung 9.2) zugegriffen werden kann.

9.4.2 CMoleculeSim

Die in CMolecule verwalteten Atome werden untereinander zu Molekülen verknüpft, hierzu werden spezielle Methoden benötigt, die unter anderem von der von CMolecule abgeleiteten Klasse CMoleculeSim bereit gestellt werden. Die Verknüpfung der Atome erfolgt durch die Methode `ConnectAtoms(const CForceField *pFF)`, die sowohl die intramolekularen Wechselwirkungen inklusive der Zwangskräfte bestimmt, als auch die Ausschlusslisten für die intermolekularen Wechselwirkungen erzeugt. In diese Ausschlusslisten gehen alle unmittelbar benachbarten Atome durch Aufruf der Methode `AddExclude` der Klasse CAtom ein, d. h. es gehen alle Atome mit einer direkten Bindung zum gewählten Atom ein. Bestimmen lassen sich die Nachbarn mit Hilfe der im Vorfahren CMolecule definierten Methode `GetNeighbour`. Ebenfalls gehen in die Ausschlusslisten die zweiten Nachbarn, die Nachbarn der Nachbarn, ein. Deren Nachbarn wiederum werden mit der Methode `AddThirdNb` in die Liste der dritten Nachbarn eingetragen.

Nachdem die Ausschlusslisten der intermolekularen Wechselwirkungen gebildet wurden, erzeugt die Methode nun über die Klasse CBondCreate die für das Molekül benötigten Instanzen der intramolekularen Wechselwirkungen und trägt diese über die Methode `AddBond` in die verkette Liste `m_pBond` ein. Eine Paarbindung wird über Instanzen der Klasse CBond12 zwischen allen durch `GetNeighbour` festgelegten, benachbarten Atomen aufgebaut. Ausgenommen sind die Atompaare, die durch eine Zwangskraft aneinander gebunden sind, und somit in einem fixen Atomabstand gehalten werden. In der Struktur-Datenbank werden diese Zwangskräfte durch eine Federkonstante K_b der Paarwechselwirkung von Null kenntlich gemacht. Die zwischen drei Atomen wirkende Wechselwirkung der Biegeschwingung wird zwischen dem betrachteten Atom und zwei seiner Nachbarn aufgebaut. Wie bei den Atompaaren ist es auch hier möglich, Zwangskräfte durch entsprechende Wahl der Federkonstanten einzufügen.

Die Festlegung der Torsionswechselwirkungen ist etwas aufwändiger, hier müssen vier unterschiedliche, benachbarte Atome gefunden werden. Ausgehend

vom gewählten Basisatom A werden zwei Nachbaratome B und X gewählt, außerdem muss vom Atom B ein von A unterschiedlicher Nachbar Y ausgesucht werden. Da in der Regel nur eine Variante der vom Basisatom A und seinen Nachbarn B, X und Y in der Kraft-Datenbank verzeichnet ist, werden die Nachbarn B, X und Y solange ausgetauscht, bis diese eine ausgewählt wird. Sind alle vier Atome ausgewählt, wird über die Biegeschwingung überprüft, ob $X-A-B$ oder $A-B-Y$ lineare Teilmoleküle bilden. Ist dies nicht der Fall, existiert zwischen den Atomen $X-A-B-Y$ ein Torsionswinkel, für den durch Aufruf der Methode `Bond14` der Klasse `CBondCreate` eine Instanz der Klasse `CBond14` erzeugt wird. Diese wird mit `AddBond` in die Liste der intramolekularen Wechselwirkungen aufgenommen.

Die Wechselwirkungen des uneigentlichen Öffnungswinkels werden als letztes in die Liste der intramolekularen Wechselwirkungen aufgenommen, hierfür müssen von dem betrachteten Atom drei unterschiedliche Nachbarn bestimmt werden. Lassen sich diese Nachbarn bestimmen, so wird, wie in den anderen Fällen auch, eine Instanz, der die Wechselwirkung bearbeitenden Klasse `CBond14i`, mit der Methode `Bond14i` der Klasse `CBondCreate` erzeugt. Diese wird, wie die Anderen mit `AddBond` in die Liste der intramolekularen Wechselwirkungen aufgenommen.

Für eine komplette Verknüpfung der Atome des Moleküls müssen noch die Zwangskräfte in einer Instanz, der von `CConstraint` abgeleiteten Klassen, verwaltet werden. Zunächst wird, in Abhängigkeit vom im Simulationsabschnitt angegebenen Parameter „Constraint," mit der Methode `Create` der Klasse `CPairConst` eine Instanz der Klasse `CRATTLEConst` oder `CSHAKEConst` erzeugt. Diese nimmt die, durch die oben beschriebene Wahl der Federkonstanten festgelegten Zwangskräfte auf. Eine Zwangskraft zwischen einem Bindungswinkel wird so umgesetzt, dass zwischen zwei der drei wechselwirkenden Atomen $B-A-C$ eine Zwangskraft eingefügt wird, d. h. die Zwangskräfte wirken zwischen den Atomen B und C. Sollen Wassermoleküle oder andere drei atomige Molekülteile mit Zwangskräften zwischen allen drei Atomen durch die speziell hierfür implementierte Klasse `CWaterConst` behandelt werden, wird dies ebenfalls im Parameter „Constraint" über den zusätzlichen Eintrag „Water" angegeben. Dann überprüft `ConnectAtoms`, ob in dem Molekül drei Zwangskräfte vorhanden sind. Bilden die Zwangskräfte außerdem ein Dreieck, werden dessen vorhandene Instanzen, der von `CPairConst` abgeleiteten Klasse, durch eine neue Instanz der Klasse `CWaterConst` mit den drei Atomen und deren drei Zwangskräften ersetzt. Hiermit ist, in dieser speziellen Aufteilung, die insbesondere auch auf das häufig als Lösungsmittel verwendete Wasser zutrifft, eine wesentlich effizientere Bestimmung der Zwangskräfte möglich.

Zur Bestimmung der Temperatur des Systems ist es notwendig die Anzahl der Freiheitsgrade im System zu kennen. Die Freiheitsgrade eines jeden Moleküls werden von der Methode `DegreeOfFreedom()` bestimmt. Hierbei besitzt jedes Atom a priori drei Freiheitsgrade, jede Zwangskraft blockiert einen dieser Frei-

heitsgrade, so dass sich die Anzahl der Freiheitsgrade im Molekül in Abhängigkeit von der Atomanzahl und der Anzahl der Zwangskräfte berechnen lässt.

In jedem Simulationsschritt müssen die auf die Atome wirkenden Kräfte neu berechnet werden, hierzu stellt CMoleculeSim unterstützende Methoden zur Verfügung. Am Anfang der Kraftberechnung wird mit der Methode `ResetF()` die zu jedem Atom gehörende und die Kraft speichernde Variable `m_vForce` auf Null gesetzt, um anschließend die Kräfte auf die Atome an deren neuen Positionen berechnen zu können. In diese Berechnungen gehen u. a. die intramolekularen Wechselwirkungen ein, die in der Methode `CalcBond()` berechnet werden. Diese werden in einer Schleife, zur Abarbeitung der in `m_pBond` gespeicherten verketteten Liste von Instanzen, berechnet. Hierzu wird dort von jeder Instanz, die stets zu einer von CBond abgeleiteten Klassen gehört, die Methode `Calculate()` aufgerufen. Der Aufbau der Liste erfolgt mit Hilfe der oben bereits beschriebenen Methode `AddBond(CBond *pNew)`.

Die Bestimmung der neuen Positionen und Geschwindigkeiten der Atome erfolgt auf Grund der Berücksichtigung der Zwangsbedingungen in zwei Teilen durch die Methoden `CalcConstraintA()` und `CalcConstraintB()` vor bzw. nach der Kraftberechnung. Beide Methoden wiederum rufen die entsprechenden Methoden der in `m_pConstraint` gespeicherten Instanz, der von CConstraint abgeleiteten Klasse, auf.

Der Datenaustausch mit der Strukturdatei (DYN) erfolgt über die bereits in CMolecule definierte Methode `Serialize`, wird jedoch ergänzt, durch einen automatischen Aufruf der oben beschriebenen Methode `ConnectAtoms`, falls die Daten eines Moleküls mit `Serialize` eingelesen werden. Die in `Serialize` von CMolecule benutzte Methode `AddDummyAtom()` zur Erzeugung der Atome muss, da jetzt Instanzen der Klasse CAtomSim und nicht Instanzen der Klasse CAtom erzeugt werden, durch eine entsprechende neue Methode ersetzt werden.

Zur Überwachung der Wechselwirkungen existieren zwei weitere Methoden. Die Methode `Dump(FILE *fOut)` ruft zum Einen die Methode `Dump(FILE *fOut)` einer jeden Instanz der Klasse CAtomSim auf und gibt zum Anderen eine Liste der verwendeten intramolekularen Wechselwirkungen und der wirkenden Zwangskräfte aus. Die zweite Methode `WriteEXL(FILE *fOut)` gibt nicht nur eine Liste der intramolekularen Wechselwirkungen und der wirkenden Zwangskräfte aus, sondern ergänzt diese durch die Liste der benachbarten Atompaare, die in den Ausschlusslisten nicht wechselwirken.

9.4.3 CSimSystemSim

Die in CSimSystemSim implementierten Methoden sind hauptsächlich durch den Umstieg von CMolecule nach CMoleculeSim begründet, insbesondere die Methoden `AddMolecule(CMoleculeSim *pMolecule)` und `GetMolecule(unsigned int nMolecule)`. Sie erfüllen die selben Aufgaben wie die Methoden der Vorfahrklasse, nur arbeitet CSimSystemSim mit einer neuen Klasse von Molekülen.

Ein Molekül wird mit der Methode `AddNewMolecule()` erzeugt, indem in `CSimSystemSim` eine Instanz der Klasse `CMoleculeSim` erzeugt wird, und nicht wie in der Vorfahrenklasse eine Instanz der Klasse `CMolecule`.

Zum Auslesen der zuvor mit `SetSimDate` gesetzten und in der Strukturdatei (DYN) gespeicherten Daten wurde die Methode `GetSimData(DWORD &nStep, double &dStep, double &dTimeStep)` implementiert. Mit ihr wird am Anfang der Simulation der aktuelle Simulationszeitpunkt festgelegt und die Schrittweite definiert. Zur Vereinfachung der Datenausgabe werden, die im vorherigen Abschnitt für `CMoleculeSim` beschriebenen Methoden `Dump(FILE *fOut)` und `WriteEXL(FILE *fOut)` ebenfalls in `CSimSystemSim` implementiert. Beide Methoden geben die Daten der eingefügten Instanzen der Klasse `CMoleculeSim`, d. h. der eingefügten Moleküle aus, indem sie zu jedem Molekül die entsprechende Methode `Dump(FILE *fOut)` bzw. `WriteEXL(FILE *fOut)` aufrufen.

9.4.4 CBondCreate

Die Erzeugung der Instanzen zur Behandlung der intramolekularen Wechselwirkungen erfolgt nicht direkt aus den in `CForcefield` gespeicherten Kraftfeldparametern, sondern über Methoden der Klasse `CBondCreate`. `CBondCreate` muss zur Erzeugung der Instanzen auf die Kraftfeldparameter zugreifen, deshalb wird dem Konstruktor eine Referenz auf die verwendete Instanz der Klasse `CForceField` übergeben. Für jede der vier intramolekularen Wechselwirkung existiert eine spezielle Methode (`Bond12(CAtomSim *pAtom1, CAtomSim *pAtom2)`, `Bond13(CAtomSim *pAtom1, CAtomSim *pAtom2, CAtomSim *pAtom3)`, `Bond14(CAtomSim *pAtom1, CAtomSim *pAtom2, CAtomSim *pAtom3, CAtomSim *pAtom4)` und `Bond14i(CAtomSim *pAtom1, CAtomSim *pAtom2, CAtomSim *pAtom3, CAtomSim *pAtom4)`) . Jede der vier Methoden erstellt eine für sie notwendige Instanz einer der Klassen `CBond12`, `CBond13`, `CBond14` oder `CBond14i` zur Implementierung der Wechselwirkung zwischen den Atomen auf die gleiche Weise. Zuerst wird versucht die Wechselwirkungsparameter mit Hilfe einer der Methoden `Get12`, `Get13`, `Get14` oder `Get14i` der Klasse `CForceField` zu bestimmen. Ist dies nicht möglich, wird eine Ausnahme ausgelöst und eine Fehlermeldung ausgegeben. Aber im Normalfall sollte mindestens eine Kombination in der Kraft-Datenbank eingetragen und durch `CForceField` eingebunden worden sein. Zur Implementierung der Wechselwirkung wird im dynamischen Hauptspeicher eine Instanz oben erwähnten Klassen erzeugt und als Ergebnis zurückgegeben.

9.4.5 CBond

In den Instanzen der Klasse `CBond` und deren Nachfahren werden die in Abschnitt 2.1 beschriebenen intramolekularen Wechselwirkungen in Routinen nachgebildet.

Wobei die Grundstruktur der abgeleiteten Klassen bereits durch CBond vorgegeben wird. Unter anderem beinhaltet CBond den Mechanismus mit dem die intramolekularen Wechselwirkungen eines Moleküls in einer einfach verketteten Liste verbunden werden. Die Verkettung der Liste wird erzeugt, in dem jede Instanz der Klasse CBond in `m_pNext` eine Referenz auf die Nächste besitzt. Um eine neue Wechselwirkung in diese Liste einzufügen wird die Methode `Add(CBond *pNew)` mit einer Referenz auf die neue Wechselwirkung aufgerufen. In ihr wird dann die Methode `Add` so lange rekursiv aufgerufen, bis `m_pNext` einen Null-Zeiger enthält, dieser wird durch eine Referenz auf die neue Instanz ersetzt. Die Methode `Next()` gibt die Referenz auf die nächste Instanz zurück, so dass man durch sukzessives aufrufen dieser Methode vom Anfang der verketteten Liste bis zum Ende gelangen kann. Ihren eigentlichen Zweck, die Bestimmung der intramolekularen Wechselwirkung, erfüllen diese Instanzen über die virtuelle Methode `Calculate()`. In der Basisklasse CBond enthält diese zwar nur einen leeren Platzhalter, der wird aber von den abgeleiteten Klassen zur Bestimmung der Potentiale und der auf Grund der Potentiale auf die Atome wirkenden Kräfte verwendet. Zur Überwachung der Molekülstrukturen gibt es die Methode `Info(char *pInfo)`, die in `pInfo` Informationen über die wechselwirkenden Atome und deren Wechselwirkungsparameter zurück gibt.

Zu jeder von CBond abgeleiteten Klasse gibt es jeweils zwei Konstruktoren, deren Parameter die für die Berechnung des Potentials und der Kraft notwendigen Daten enthalten. Die Parameter des ersten Konstruktors bestehen aus einer für die Wechselwirkung charakteristischen Datenstruktur (TParam12, TParam13, TParam14 oder TParam14i), die die Kraftfeldparameter der Wechselwirkung enthalten und deren Referenzen auf die zur Wechselwirkung gehörenden Atome zeigen. Alternativ können im zweiten Konstruktor die Kraftfeldparameter auch explizit in einzelnen Parametern angegeben werden. Die Berechnung der Wechselwirkung erfolgt wie oben erwähnt in der virtuellen Methode `Calculate()`, die hierfür auf die im Konstruktor übergebenen Kraftfeldparameter und Referenzen zurückgreift. Die Berechnung der Potentiale erfolgt nach den in Abschnitt 2.1 angegebenen Formeln ((2.2), (2.4), (2.8) und (2.9)), aus denen mit Hilfe von Gleichung (2.1) die resultierenden Kräfte bestimmt werden.

9.4.6 CConstraint

Alle implementierten Klassen zur Behandlung von Zwangskräften basieren auf der Klasse CConstraint, sie enthält einen Rahmen für die Methoden, die zur Bestimmung der Zwangskräfte bzw. deren Anwendung auf die Atome benötigt werden. Außerdem werden hier die für alle Zwangskräfte gültigen Toleranz- und Iterationsgrenzen festgelegt. Wie bereits erwähnt, werden Zwangskräfte und der mit ihnen eng verbundene Iterationsalgorithmus in zwei Teilen bestimmt. In `CalculateA()` wird der Anteil berechnet, der vor der Berechnung der auf die Atome wirkenden Kräfte benötigt wird. Die Berechnung des Anteils, der nach

Berechnung der Kräfte zu berechnen ist, erfolgt in der Methode `CalculateB()`. Zur korrekten Bestimmung der Freiheitsgrade ist es notwendig die Anzahl der Zwangskräfte in einem Molekül zu berücksichtigen, aus diesem Grund ist die Methode `CountConst()` in `CConstraint` deklariert. Ebenfalls deklariert ist die Methode `Info(WORD nConst, char *pInfo)` zur Ausgabe der internen Daten, um diese in der Testphase auf Korrektheit überprüfen zu können.

Da die Bestimmung der Zwangskräfte bzw. die Festlegung der Koordinaten der betroffenen Atome in den implementierten Verfahren ein iterativer Prozess ist, der nicht unendlich fortgeführt werden kann, wird mit der Methode `SetTol(unsigned nMaxIt, double dTol)` eine Obergrenze für die Anzahl der Iterationen festgelegt. Zur Geschwindigkeitssteigerung wird zusätzlich die Toleranz `dTol` festgelegt, innerhalb der eine Korrektur der Atomabstände auf die Zwangsbedingungen unterbleibt. Beide Werte können über die Methoden `GetTol()` und `GetMaxIt()` ausgelesen werden.

Generell lassen sich alle Zwangskräfte auf Wechselwirkungen zwischen zwei Atomen reduzieren. Für jedes Molekül gibt es deshalb, mit Hilfe der von `CConstraint` abgeleiteten Klasse `CPairConst`, die Möglichkeit alle Zwangskräfte in einer Liste mit Zwangskräften zwischen paaren von Atomen festzulegen. Die hierbei erzeugte Instanz gehört nicht zur Klasse `CPairConst` selbst, da diese nur einen nach außen einheitlichen Rahmen für die beiden von ihr abgeleiteten Klassen `CSHAKEConst` und `CRATTLEConst` bildet. Die Instanzen dieser Nachfahren werden mit der statischen Methode `Create(WORD cnConst, CMoleculeSim *pMol)` erzeugt. Sie werden von dieser, in Abhängigkeit vom mit `SetType(BYTE nType)` festgelegten und mit `GetType()` wieder lesbaren Auswahlkriterium, im dynamischen Speicher erzeugt.

Der Konstruktor `CPairConst(WORD cnConst, CMoleculeSim *pMol)`, der auch von den Konstruktoren der Nachfahren aufgerufen werden muss, speichert eine Referenz auf die im Molekül `pMol` angegebenen Atome im Attribut `m_pmpAtom`, so kann auf die Atome des Moleküls, insbesondere zur Berechnung der Zwangskräfte, mit der Methode `GetAtom(short nAtom)` schneller zugegriffen werden als über einen Zugriff auf die Atome über das Molekül. Die im letzten Absatz beschriebene Liste der Zwangskräfte wird mit Hilfe der Methoden `AddConstSq(short atA, short atB, double dDistSq)` und `AddConst(short atA, short atB, double dDist)` gebildet, in dem an das Ende der Listen `m_mnAtomA`, `m_mnAtomB` und `m_mdDistSqAB` die neuen Daten aus den übergebenen Parametern angehängt werden. Die Länge dieser Listen, d. h. die Anzahl der Zwangskräfte wird über die Größe der ersten Liste `m_mnAtomA` mit Hilfe der Methode `CountConst()` bestimmt.

Zur Festlegung der Zwangskräfte stehen drei Hilfsmethoden zur Verfügung. `GetAtomNoA(WORD nConst)` und `GetAtomNoB(WORD nConst)` lesen aus den jeweiligen Listen die Positionsnummern der Atome aus, die der Zwangskraft unterliegenden. Der von der Zwangskraft festgelegte Atomabstand wird mit der Methode `GetConst(WORD nConst)` ausgelesen. Die gewonnenen Daten werden für ei-

ne spätere Ausgabe in eine Datei durch die Methode `Info(WORD nConst, char *pInfo)` in eine lesbare Form aufbereitet.

Die Berechnung der Wechselwirkungen erfolgt in einer der beiden, von `CPairConst` abgeleiteten Klassen, `CSHAKEConst` bzw. `CRATTLEConst`. Die Berechnung der Geschwindigkeiten und Positionen der Atome unter Berücksichtigung der Zwangskräfte kann sowohl vor der Kraftberechnung in `CalculateA()`, als auch danach in `CalculateB()` erfolgen.

In der Klasse `CSHAKEConst` werden Orte und Geschwindigkeiten gleichzeitig in der Methode `CalculateB()` bestimmt. Die Methode `CalculateA()` wird nicht benötigt. Zuerst werden aus den Atomkoordinaten und den Geschwindigkeiten die neuen Koordinaten des Atoms berechnet, diese berücksichtigen aber noch nicht die Zwangskräfte, welche in einem zweiten Schritt iterativ durch Korrektur der entsprechenden Koordinaten in das System eingebaut werden. Da das SHAKE-Verfahren in den hier durchgeführten Simulationen nicht verwendet wird, wird es hier nur kurz angerissen. Ausführlich beschrieben wird es in den Artikeln von Ryckaert u. a. [41, 82]. In jedem Iterationsschritt wird für jedes, den Zwangskräften unterliegende Paar von Atomen, eine von der Abweichung zum Idealabstand abhängige Zwangskraft bestimmt. Diese ändert dann unter Berücksichtigung der Atommassen die Lage der Atome. Wirken auf ein Atom mehrere Zwangskräfte, sind zur Bestimmung der resultierenden Zwangskraft in der Regel mehrere Iterationsschritte erforderlich. Aufgrund der einfachen Zwangskräfte am Wasserstoff des Bisacrylamids, reicht hier ein Iterationsschritt zu ihrer Berechnung aus.

Dem gegenüber nutzt das RATTLE-Verfahren beide Methoden. In `CalculateA()` werden, vor der Berechnung der Kräfte, die neuen Positionen der Atome bestimmt und die Geschwindigkeit teilweise berechnet. Nach der Kraftberechnung kann dann die Geschwindigkeit der Atome endgültig in `CalculateB()` bestimmt werden. Beide Routinen werden ausführlich in Abschnitt 4.2 und in dem Artikel von Andersen [42] beschrieben.

Für starre drei atomige Moleküle wie dem hier verwendeten Wasser gibt es ein, bereits bei der Erzeugung der Moleküle erwähntes, in der Klasse `CWaterConst` implementiertes, speziell optimiertes Verfahren, das unter anderem auch von Allen und Tildesley in ihrem Buch [32] beschrieben wurde. Dem Konstruktor werden Referenzen auf die drei den Zwangsbedingungen unterliegenden Atome, so wie die zwischen ihnen liegenden Atomabstände übergeben, um diese innerhalb der Instanz in Attributen zu speichern. Damit die von `CConstraint` abgeleitete Klasse verwendet werden kann, müssen die leeren virtuellen Methoden von `CWaterConst` mit Inhalt gefüllt werden. Die Methode `CountConst()` gibt in allen von `CConstraint` abgeleiteten Klassen die Anzahl der Zwangskräfte zurück, die in diesem Fall des starren drei atomigen Moleküls immer drei beträgt. Bei `Info(WORD nConst, char *pInfo)` handelt es sich um eine weitere virtuelle Methode, mit der Informationen über die Zwangskräfte in Überwachungsdateien ausgegeben werden, und die an `CWaterConst` angepasst werden musste. Die Be-

rechnung der Zwangskräfte erfolgt nach Bestimmung der Kräfte, so dass die Methode `CalculateA()` nur einen leeren Platzhalter enthält, der somit eine einheitliche Behandlung der Zwangskräfte erlaubt. Das Herzstück bildet die Methode `CalculateB()`, in der sämtliche Berechnungen für die Zwangskräfte, so wie zur Bewegung der Atome nach dem SETTLE Algorithmus [83] von Miyamoto und Kollman erfolgen.

Da die Berechnung der Zwangskräfte des Wassers nicht zwangsläufig mit diesem Algorithmus erfolgen muss, sondern auch mit einem der beiden anderen Verfahren erfolgen kann, sind hierfür zwei weitere Methoden implementiert. Ein Aufruf von `UseWaterConst()`, der bei Angabe des Parameters „Water“ im Schlüsselwort „Constraint“ der Steuerungsdatei automatisch erfolgt, legt fest, dass die Zwangskräfte des Wassers oder anderer Moleküle mit den selben Zwangsbedingungen von `CWaterConst` zu behandeln sind. Die zweite Methode `IsWaterConst()` wird bei der Konstruktion des Moleküls verwendet, um zu entscheiden, ob die Zwangskräfte der Wassermoleküle und Anderer mit diesem Algorithmus bestimmt werden sollen oder mit einem der Anderen.

9.4.7 CLenardJones

In jeder Simulation wird in `CSimulation` nur eine Instanz der Klasse `CLenardJones` erzeugt. Diese bestimmt die in Abschnitt 2.2.1 beschriebenen van der Walls- bzw. Lenard-Jones-Wechselwirkungen der Atome untereinander in jedem Simulationsschritt neu. Die Wechselwirkungsparameter müssen aber nicht bei jeder Berechnung direkt aus den Daten der Atome der Instanz von `CSimulation` gewonnen werden. Sie werden vielmehr für eine spätere optimierte Berechnung der Wechselwirkungen, zu Beginn der Simulation einmal bestimmt und dann optimiert in lokalen Attributen abgespeichert. Hierzu werden mit Hilfe der Methode `Init(const CSimulation &sim, double dCutOffSq)` die drei in Abbildung 9.9 gezeigten Listen parallel erzeugt. Bei einer Wechselwirkung zwischen zwei Atomen, d. h. die Wechselwirkungsparameter beider Atome sind ungleich Null und das jeweils andere Atom ist weder in der Ausschlussliste (Abschnitt 2.2.1) des anderen Atoms noch dessen dritter Nachbar, werden die Wechselwirkungsparameter beider Atome mit Hilfe der Mischungsregeln aus Gleichung (2.15) bzw. (2.14) zu einem gemeinsamen Wechselwirkungsparametern verschmolzen und in den Listen `m_mdC6ij` und `m_mdC12ij` abgespeichert. Für die eindeutige Identifizierung dieser Wechselwirkungen wird eine Referenz des zweiten Atoms in einem vom ersten Atom gebildeten Abschnitt `m_mpAtCalc` gespeichert. Die in `m_mpAtCalc` gespeicherte Liste besteht aus Abschnitten variabler Länge, die durch leere Elemente (Null-Zeiger) voneinander getrennt sind. Das erste Element eines Abschnitts enthält immer die Referenz auf das Atom, mit dem die weiteren Atome dieses Abschnittes wechselwirken. Die Behandlung der dritten Nachbarn erfolgt analog.

In der Simulationsphase werden die oben erzeugten Listen von der Methode

m_mpAtCalc	m_mdC6ij	m_mdC12ij
at1		
at2	$C6_{12}$	$C12_{12}$
at2	$C6_{12}$	$C12_{12}$
⋮	⋮	⋮
at2	$C6_{12}$	$C12_{12}$
NULL		
at1		
at2	$C6_{12}$	$C12_{12}$
⋮	⋮	⋮
at2	$C6_{12}$	$C12_{12}$
NULL		
	⋮	
at1		
at2	$C6_{12}$	$C12_{12}$
⋮	⋮	⋮
at2	$C6_{12}$	$C12_{12}$
NULL		
NULL	0	0

Abbildung 9.9: Aufbau der Listen für die Berechnung der Wechselwirkungen der Atome im Lenard-Jones-Potential.

`Calculate()` verwendet, um die vom Lenard-Jones-Potential verursachten Kräfte auf die Atome zu bestimmen. Hierzu werden die Abschnitte der Liste `m_mpAtCalc` nacheinander abgearbeitet. Jeder Abschnitt enthält alle die mit dem ersten Atom des Abschnitts wechselwirkenden Atome. Der Rechenaufwand reduziert sich hierdurch um die Bestimmung der Wechselwirkungskoeffizienten C_6 und C_{12}. Desweiteren werden nur die Paare berücksichtigt, die untereinander wechselwirken.

In der Debug-Phase können, zur Verifizierung der in der Instanz von `CLenardJones` erzeugten Attribute, der cutoff-Radius aus `m_dCutOffSqr` und die Liste der Atome `m_mpAtCalc` durch Aufruf der Methode `Dump(FILE *fOut)` in eine mit `fOut` festgelegte Datei gespeichert werden.

9.4.8 CEwaldSum

Die Behandlung der elektrostatischen Wechselwirkungen erfolgt in einer Instanz der Klasse `CEwaldSum`. Die Berechnung der elektrostatischen Wechselwirkungen über das Ewald-Verfahren (vgl. Abschnitt 2.2.3) lässt sich, wie beim Lenard-Jones-Potential, aufteilen in einen Initialisierungsschritt, der vor der eigentlichen Simulation ausgeführt wird und den Berechnungsschritten während der Simula-

tion, in denen die Wechselwirkungen bestimmt werden.

Die Initialisierung der Klassenattribute erfolgt in der Methode `Init(const CSimulation &sim, double dCutOffSq, double dKappa, double dDiel, WORD nKmax)`, in der die Attribute mit Hilfe der Randparameter und der zu simulierenden Atome gesetzt werden. Als Erstes werden die beim Aufruf übergebenen Parameter für eine spätere Berücksichtigung in der Simulation in die lokalen Attribute für den quadratischen cutoff-Radius (`m_dCutOffSqr`), dem dielektrischen Vorfaktor $(4\pi\epsilon_0\epsilon_r)^{-1}$ (`m_dDiel`), der Verteilungsbreite (`m_dKappa`) und dem Auswertungsradius im reziproken Raum (`m_nKmax`$^{*2\pi}/_L$, mit der Kantenlänge L des Simulationsraums) gespeichert. Um die Initialisierung zu beschleunigen wird dann eine ausreichend große Anzahl leerer Felder für die Listen der Attribute `m_mpAtWW`, `m_mdDQijWW`, `m_mpAtExclude`, `m_mdDQijExclude` und `m_mpAtList` generiert. Die Listen `m_mpAtWW` und `m_mpAtExclude` werden wie die Liste für die Lenard-Jones-Wechselwirkung in Abschnitte aufgeteilt, wobei wieder für jedes Atom ein Abschnitt mit Referenzen auf die wechselwirkenden Atome vorhanden ist, deren Ende durch eine leere Referenz gekennzeichnet ist. Gleiches gilt für die Parameterlisten `m_mdDQijWW` und `m_mdDQijExclude`, sie besitzen den selben Aufbau wie die Listen `m_mdC6ij` und `m_mdC12ij` der Lenard-Jones-Wechselwirkung. Abgeschlossen wird dieser erste Initialisierungsabschnitt mit der Kennzeichnung des Endes der Listen durch eine zusätzliche leere Referenz und der Freigabe der ungenutzten Felder .

Vom ersten Teil der Initialisierung wird automatisch der zweite Teil aufgerufen. Dieser befindet sich in der privaten Methode `InitKVector()`, die die Stützpunkte $\vec{\mathbf{k}}$ im reziproken Gitter festlegt. Generell ist die Lage der Stützpunkte $\vec{\mathbf{k}}$ nur von der Größe des Simulationsraumes abhängig. Da die hier erstellten Simulationen, wegen des über die periodischen Randbedingungen hinausgehenden Polymergerüstes, bei festen Volumina V, d. h. im mikrokanonischen bzw. kanonischen Ensemble ablaufen, können die Stützpunkte $\vec{\mathbf{k}}$ bereits bei der Initialisierung der Simulation festgelegt werden. Zu jedem Stützpunkt $\vec{\mathbf{k}}$ gibt es einen nur von der Konstanten κ und dem Stützpunkt $\vec{\mathbf{k}}$ abhängigen Faktor K_v (vgl. Gleichung 2.29):

$$K_v = \frac{2\pi}{\vec{\mathbf{k}}^2} \exp\left(\frac{-1}{4\kappa^2}\vec{\mathbf{k}}^2\right)$$

Dieser Faktor K_v wird in der Methode `InitKVector()` berechnet und anhand einer fortlaufenden Nummerierung der Stützpunkte in den Feldelementen des Attributs `m_mdKvec`, für eine beschleunigte Berechnung des reziproken Anteils der Ewald-Methode, gespeichert.

Da „dynC“ auch die einfache Berechnung der Coulomb-Wechselwirkungen unterstützt, wurde die Methode `CoulombCalculate()` in die Klasse `CEwaldSum` integriert. Zur Berechnung der Wechselwirkungen werden ebenfalls die für die Ewald-Methode berechneten Attribute verwendet, insbesondere entspricht die Liste der wechselwirkenden Teilchenpaare der im Attribut `m_mpAtWW` gespeicherten

Liste. Die Abarbeitung dieser Liste entspricht dem Vorgehen der weiter unten beschriebenen Methode `CalcReal()`. Beide Methoden lesen die Gesamtladung q_{ij} der wechselwirkenden Teilchen i und j, inkusive des dielektrischen Vorfaktors $(4\pi\epsilon_0\epsilon_r)^{-1}$, der ebenfalls zu durchlaufenden Liste des Attributs `m_mdDQijWW` aus. Grundlage der Berechnung bildet Gleichung (2.22). Für die Bestimmung der Energie muss somit nur noch der Teilchenabstand `dRij` unter Berücksichtigung der periodischen Randbedingungen berechnet werden. Ist der Abstand kleiner als der cutoff-Radius ergibt sich die Wechselwirkungsenergie `dEij` aus:

$$\texttt{dEij} = \frac{\texttt{m\_mdDQijWW}}{\texttt{dRij}}$$

Die daraus resultierende Kraft auf die Teilchen lässt sich aus der Ableitung der Energieformel (vgl. Gleichung 2.1) bestimmen und führt zu:

$$\texttt{vFi} = \texttt{vRij}\frac{\texttt{dEij}}{\texttt{dRij}^2}$$

Die Berechnung nach der Ewald-Methode lässt sich in drei logische Teile aufspalten (Gleichung (2.31)), dem Realanteil (Gleichung (2.28)), dem reziproken Anteil (Gleichung (2.29)) und dem Selbstterm (Glecihung (2.30)). In `CEwaldSum` existiert für jedes dieser Teile eine eigene Methode, so dass zur Berechnung der Wechselwirkungen nach dem Verfahren von Ewald, die Methode `Calculate()` nur diese drei als privat deklarierten Methoden `CalcReal()`, `CalcK()` und `CalcSelf()` aufrufen und unter Berücksichtigung von Gleichung (2.31) die zurückgegebenen Teilenergien zu einer Coulomb-Energie summieren muss.

Die private Methode `CalcReal()` berechnet den durch Gleichung (2.28) vorgegebenen Realteil der Ewald-Summe. Wie aus dieser Gleichung abzulesen ist, benötigt man zur Lösung dieser Aufgabe die konjugierte Fehlerfunktion $\mathrm{erfc}(x)$. Die bereits in Gleichung (2.26) eingeführte Funktion wird hier durch die Methode `Erfc(double dKR)` angenähert, diese verwendet die Approximation von Abramowitz [84] , die für alle $x \geq 0$ mit einem Approximationsfehler $\epsilon(x) < 1.5\ 10^{-7}$ behaftet ist:

$$\begin{aligned}\mathrm{erfc}(x) \;=\; & \left(a_1\left(\frac{1}{1+px}\right) + a_2\left(\frac{1}{1+px}\right)^2 + a_3\left(\frac{1}{1+px}\right)^3\right. \\ & \left. + a_4\left(\frac{1}{1+px}\right)^4 + a_5\left(\frac{1}{1+px}\right)^5\right)e^{-x^2} + \epsilon(x) \qquad (9.3)\end{aligned}$$

mit: $a_1 = 0,254829592 \quad a_2 = -0,284496736 \quad a_3 = 1,421413741$
$a_4 = -1,453152027 \quad a_5 = 1,061405429 \quad p = 0,3275911$

Zur Durchführung der Summationen werden, wie bereits bei der klassischen Coulomb-Berechnung erwähnt, die Felder der Attribute `m_mpAtWW` und

`m_mdDQijWW` verwendet. Hierzu wird in einer äußersten Schleife dem lokalen Zeiger `pAtI` die Referenz auf das erste Teilchen i aus der Liste `m_mpAtWW` zugewiesen. Die Referenzen auf die Teilchen j mit denen dieses Teilchen wechselwirkt werden in einer Subschleife ebenfalls aus der Liste entnommen.

Entscheidend für die Berücksichtigung der Wechselwirkung zwischen i und j ist deren Abstand. Ist dieser größer als der cutoff-Radius, wird die Wechselwirkung nicht mehr berücksichtigt, und der Abstand des nächsten Teilchenpaars überprüft. Aus Gründen der Optimierung wird nicht der Abstand direkt mit dem cutoff-Radius verglichen, sondern, der quadratische Teilchenabstand, unter Berücksichtigung der periodischen Randbedingungen, mit dem quadratischen cutoff-Radius. Ebenfalls zur Optimierung werden vor der Berechnung des Potentials (Gleichung (2.28)) und der Kräfte einige Zwischenrechnungen durchgeführt und für eine spätere Verwendung in lokale Variablen gespeichert. Die berechneten Kräfte werden dann über die Methoden `AddF` und `SubF` der Klasse `CAtomSim` zu den bereits vorhandenen Kräften der Atome addiert bzw. subtrahiert, im Gegensatz dazu wird die berechnete potentielle Energie an die aufrufende Methode zurückgegeben.

Die langreichweitigen, über den cutoff-Radius und den Simulationsraum hinausgehenden Coulomb-Wechselwirkungen, werden im reziproken Teil der Ewald-Methode berechnet. Dieser Teil der Ewald-Methode wird durch die Gleichung (2.29) beschrieben und ist in der Methode `CalcK()` implementiert. Sie enthält den kompliziertesten Teil der Berechnung und ist nach der Methode `CalcR()` die aufwendigste Methode. Sie benötigt innerhalb der Coulomb-Berechnung etwa 40% der Rechenzeit und dient der Bestimmung der potentiellen Energie, so wie der daraus resultierenden Kräfte.

Die Berechnung des reziproken Anteils lässt sich in drei Phasen aufteilen. In der ersten Phase wird der zweite Exponentialterm aus Gleichung (2.29) in die einzelnen Raumanteile aufgespalten:

$$\begin{aligned} \exp\left(-i\vec{\mathbf{kr}}_j\right) &= \exp\left(-ik_x r_{jx}\right) * \exp\left(-ik_y r_{jy}\right) * \exp\left(-ik_z r_{jz}\right) && (9.4) \\ &= \texttt{cxEikX.Get(}j\texttt{,}k_x\texttt{)} * \texttt{cxEikY.Get(}j\texttt{,}k_y\texttt{)} * \\ &\quad \texttt{cxEikZ.Get(}j\texttt{,}k_z\texttt{)} && (9.5) \end{aligned}$$

Der Rechenaufwand hierfür lässt sich entscheidend verringern, wenn für jede Raumrichtung Listen (`cxEikX`, `cxEikY` und `cxEikZ`) mit den berechneten Exponentialtermen für die festgelegten Raumpunkte des reziproken Gitters und für die Teilchen angelegt werden. Der Exponentialterm muss damit nicht für jedes Teilchen j und jeden Punkt im reziproken Raum $\vec{\mathbf{k}}$ bestimmt werden, sondern kann aus der Kombination der in den Listen eingetragenen Werte für die drei Raumrichtungen bestimmt werden.

Die zweite Phase bildet den Hauptteil der Berechnungen. Hier wird, nach der Zusammenfassung mehrerer Konstanten zu einer Konstanten und der Initialisierung einiger Laufvariablen, die Summe über die reziproken Raumpunkte $\vec{\mathbf{k}}$ in

separate Summen, für jede Raumrichtung aufgespalten. Die zweite Summe wird über die wechselwirkenden Teilchen aus Gleichung (2.29) mit Hilfe des in Gleichung (9.5) gezeigten Verfahrens bestimmt. Das Betragsquadrat dieses komplexen Ergebnisses bildet, zusammen mit dem im Initialisierungsprozess bestimmten ersten Exponentialterm, im Attribut `m_mdKvec`, die Energie, dessen Summe wiederum die Gesamtenergie `dE` dieses Teils der Wechselwirkung bildet. Neben den Energien der Teilchen werden auch die von diesen Energien auf die Teilchen wirkenden Kräfte nach Gleichung (2.1) bestimmt. In der abschließenden dritten Phase wird die in der zweiten Phase bestimmte Gesamtenergie `dE` mit Hilfe des Vorfaktors `m_dDiel` in die vom Programm verwendete Einheit $^{\text{kcal}}/_{\text{mol}}$ transformiert.

Der letzte zu berechnende Anteil der Ewald-Methode enthält den Selbstterm, dieser korrigiert einen durch die Selbstwechselwirkung verursachten Fehler. Bestimmt wird er in der Methode `CalcSelf()`. Die Selbstwechselwirkung entsteht im reziproken Anteil der Ewald-Methode durch die periodische Fortsetzung des Simulationsraumes. Diese erzeugt für jedes Molekül eine Wechselwirkung seiner Atome mit den Atomen der Kopien des selben Moleküls in den periodischen Fortsetzungen. Wie in den beiden anderen Teilen werden auch hier die wechselwirkenden Teilchen und deren Wechselwirkungsstärke (Ladungen) aus einer in `Init(const CSimulation &sim, double dCutOffSq, double dKappa, double dDiel, WORD nKmax)` erzeugten Liste entnommen. Die im Attribut `m_mpAtExclude` gespeicherte Liste besitzt, für jedes Teilchen mit Selbstwechselwirkung, einen Abschnitt mit einer Unterliste der Teilchen mit dem dieses wechselwirkt. Ergänzt werden die Listen in `m_mpAtExclude` durch eine weitere Liste im Attribut `m_mdDQijExclude`, diese enthält die Wechselwirkungsstärke der Selbstwechselwirkung und die durch die Wechselwirkung des ersten Teilchens mit einem Teilchen aus der Unterliste verursachten Paarwechselwirkung. Die Liste in `m_mpAtExclude` wird mit Hilfe zweier verschachtelter Schleifen abgearbeitet. In der äußeren Schleife werden die Teilchen aus der Liste ausgewählt, die wiederum mit den Teilchen aus den Unterlisten in Wechselwirkung stehen. In einer inneren Schleife werden aus den Wechselwirkungen der Teilchen mit dem Teilchen der äußeren Schleife, deren Kräfte und Energien (gespeichert in `dE`) berechnet. Beide Schleifen finden sich in Gleichung (2.30) in Form einer Doppelsumme wieder. Wie der Gleichung zu entnehmen ist, werden neben der Ladung ($q_i q_j$), die in `m_mdDQijExclude` gespeichert wurde, auch der Abstand r_{ij}, bestimmt durch die Länge des Vektors `vRij`, so wie die Fehlerfunktion $\text{erf}(\kappa r_{ij})$ zur Bestimmung der Wechselwirkungsenergie benötigt. Zwischen der Fehlerfunktion $\text{erf}(x)$ und der konjugierten Fehlerfunktion $\text{erfc}(x)$ existiert die einfache Relation:

$$\text{erf}(x) \;=\; 1 - \text{erfc}(x) \tag{9.6}$$

mit der sich die Fehlerfunktion $\text{erf}(x)$ ebenfalls durch die Methode `Erfc(double dKR)` beschreiben lässt. Zu Beginn der oben beschriebenen äußeren Schleife erfolgt die Berechnung der dritten Summe aus Gleichung (2.30). Unter Verwendung der

zusätzlichen Variablen `dE1` werden hierzu die Selbstwechselwirkungsterme (q_i^2) der Teilche i, die ebenfalls bei der Initialisierung in `m_mdDQijExclude` gespeichert wurden, summiert und am Ende mit dem in Gleichung (2.30) vorgegebenen Faktor $\kappa/\sqrt{\pi}$ multipliziert. `CalcSelf()` gibt die Energie der Selbstwechselwirkung, bestehend aus der oben bestimmten Energie und der in der inneren Schleife berechneten Energie `dE` zurück.

9.4.9 Complex

Im Ewald-Summen-Verfahren werden zur Berechnung der Coulomb-Wechselwirkungen komplexe Zahlen benötigt, dies wird durch Implementation der Klasse `Complex` erreicht. Intern wird eine komplexe Zahl als Summe aus Real- und Imaginärteil betrachtet, deren Werte in den Attributen `re` und `im` gespeichert werden. In den Anwendungsroutinen können beide Werte über die Methoden `real()` und `imag()` wieder ausgelesen werden. Zur Erzeugung komplexer Zahlen stehen drei Konstruktoren zur Verfügung. Der erste Konstruktor `Complex()` erzeugt eine undefinierte komplexe Zahl. Dieser Konstruktor wird in erster Linie zur Reservierung von Speicherbereich verwendet, insbesondere bei der Erzeugung von Feldern komplexer Zahlen, wie bei den Feldern `cxEik`, `cxEikX`, `cxEikY` und `cxEikZ` in der Methode `CalcK()` der Klasse `CEwaldSum`. Der zweite Konstruktor `Complex(const Complex& y)` übernimmt den Wert der übergebenen komplexen Zahl `y`, so dass dieser wie eine Kopierfunktion für komplexe Zahlen wirkt. Dem dritten Konstruktor `Complex(double r, double i=0)` wird entweder bei Übergabe eines Wertes dieser als reeller Wert zugewiesen, oder bei Übergabe zweier Werte, der Erste als Realanteil und der Zweite als imaginärer Anteil interpretiert. Ein Konstruktor für komplexe Zahlen in der Exponentialschreibweise existiert nicht, da dieser wie der letztgenannte Konstruktor zwei Parameter besäße. Statt dessen wird zur Speicherung / Umwandlung einer komplexen Zahl aus der Exponentialschreibweise die Methode `polar(double r, double t)` verwendet, die den Radius `r` und die Phase `t` des komplexen Wertes $\mathtt{r}e^{i\mathtt{t}}$ nach der Eulerschen Gleichung konvertiert:

$$\begin{aligned} \mathtt{re} &= \mathtt{r}\cos(\mathtt{t}) \\ \mathtt{im} &= \mathtt{r}\sin(\mathtt{t}) \end{aligned}$$

Um die Klasse generell einsetzen zu können, ist auch der umgekehrte Fall implementiert, den Radius `r` und die Phase `t` lassen sich mit Hilfe der Methoden `rad()` und `phase()` bestimmen.

Zur Bestimmung der konjugiert komplexen Zahl sind zwei Methoden implementiert. `conj()`, die den konjugiert komplexen Wert des in der Instanz gespeicherten Wertes zurück gibt, und `conj(Complex &cx)`, bei dem die komplexe Zahl in der Instanz den konjugiert komplexen Wert von `cx` annimmt. Für die Ewald-Methode wird, wegen höheren Ausführungsgeschwindigkeit, nur die zweite Methode verwendet. Um mit diesen Zahlen rechnen zu können, sind ebenfalls

Zuweisungs- und Rechenoperatoren implementiert, diese folgen den allgemeinen Rechenregeln für komplexe Zahlen und werden deshalb hier nicht weiter behandelt.

9.4.10 CSimulation

Die in diesem Abschnitt beschriebene Klasse `CSimulation` führt die in der Simulationsdatei festgelegten MD-Simulationen durch. Die Initialisierung der lokalen Attribute mit Start- und Standardwerten erfolgt wie üblich im Konstruktor (`CSimulation()`). Hervorzuheben sind hier die Festlegung auf eine Schrittweite von 1 fs, ein cutoff-Radius von 10 Å, so wie eine Beschränkung der Anzahl der Iterationen auf 3000 pro Simulationsschritt und eine Toleranz von 10^{-5} als Abbruchkriterium bei den Zwangskräften. Zur späteren Bestimmung der Coulomb-Wechselwirkungen mit dem Ewald-Verfahren wird außerdem der Vorfaktor `m_Diel` auf einen Wert von 332.0636 für das Vakuum festgelegt, so wie κ und k_{max} die Werte 6 bzw. 5 zugewiesen. Auf der anderen Seite wird der während Initialisierung und Simulation in Anspruch genommene Speicher im Destruktor (`~CSimulation()`) wieder frei gegeben.

Der Ablauf und die Randbedingungen einer MD-Simulation werden, wie erwähnt in der Simulationsdatei festgelegt. Die Simulationsdatei wird mit Hilfe der Methode `Load(const char *szFile)` bzw. `Load(FILE *fIn)` eingelesen und analysiert. Sie liest die angegebene Datei zeilenweise ein und wertet diese mit Hilfe der Klasse `CParseScriptline` aus. Hierbei wird nach dem Abschnitt „[Simulation]“ gesucht, da dieser sämtliche Informationen über die Simulation enthält. Die einzelnen Zeilen dieses Abschnittes werden auf bestimmte Schlüsselwörter durchsucht, die in Abschnitt 4.4 bereits beschrieben wurden. Den Schlüsselwörtern sind lokale Attribute der Klasse `CSimulation` zugeordnet, dessen Werte in Abhängigkeit von den zusätzlich angegebenen Parametern geändert werden. Zum Beispiel wird dem lokalen Attribut `m_dLJCutOff` in den Zeilen „LJCutOff(10)“, „LJCutOff 10“ oder „LJCutOff=10“ ein Radius von 10 Å zugewiesen. Ein anderes Schlüsselwort ist das Wort „Step“, mit dem die anfängliche Schrittweite des zu simulierenden Systems festgelegt wird. Da die Schrittweite im Laufe der Simulation an mehreren Punkten im Programm geändert wird, und jede Änderung aufwändig ist, wurde hierfür eine eigene Methode `SetTimeStep(double dStep)` konstruiert. Um die Schrittweite zu ändern, wird zuerst in jedem Molekül die mit der Schrittweite skalierte Geschwindigkeit seiner Teilchen auf die neue Schrittweite umskaliert. Anschließend wird die neue Schrittweite bzw. dessen Quadrat den Attributen `m_dTimeStep` und `m_dTimeStepSqr` zugewiesen.

Die im Abschnitt „[Simulation]“ der Simulationsdatei festgelegte Simulation wird durch Aufruf der Methode `Go()` gestartet, die wiederum zur Initialisierung der eigenen Attribute und der Attribute der langreichweitigen Wechselwirkungen die Methode `Init()` aufruft. Ist der Status der Ablaufverfolgung auf einer Stufe (gTracelevel) größer als Zehn, so wird die Datei „DynC_Dump.txt“ er-

stellt, in die, durch Aufruf der Methode `Dump`, die wichtigsten Laufzeitparameter des simulierten Systems und der langreichweitigen Wechselwirkungen geschrieben werden. Danach werden in Abhängigkeit von den Vorgaben der Simulationsdatei die Dateien für die Trajektorien und Konfiguration erzeugt bzw. geöffnet und die Randparameter der Simulation für eine Berücksichtigung in der Auswertung gespeichert. In der Regel werden während der Simulation in festen Abständen Zustandsdaten wie Temperatur oder Energien in eine weitere Datei, der Reportdatei geschrieben. Desweiteren werden Daten wie Größe des Simulationsraums, Dichte und Solltemperatur durch Aufruf der Methode `ReportInit` im Klartext an die ausgewählte Datei angehängt. Bevor mit der eigentlichen Simulation begonnen werden kann, müssen noch die Variablen für die Zählung der Simulationsschritte zwischen den Abspeicherungen der Trajektorien und der weiteren Zustandsdaten initialisiert werden.

Die Simulation selbst wird aufgeteilt in einen Programmabschnitt zur Simulation, d. h. Produktion der Simulationsdaten und einen Abschnitt für die Equilibrierung des Systems. Die Produktionsphase wird solange durchlaufen, bis der letzte durch `m_nSimuEnd` definierte Simulationsschritt ausgeführt wurde. Jeder Produktionsschritt wird mit der Bestimmung des neuen Phasenraumpunktes aus dem alten Punkt eingeleitet. Der neue Punkt im Phasenraum wird in drei Schritten mit der Methode `Step()` bestimmt. Diese ruft für diese drei Schritte die drei Methoden `MoveA()`, `CalculateF()` und `MoveB()` auf. Der neue Phasenraumpunkt und dessen Zustandsdaten müssen in festgelegten Abständen in die Report-Datei, die Trajektorien-Datei, die Strukturdatei (DYN) oder die PDB-Datei gespeichert werden. Bei einer Simulation im NVT-System wird abschließend die Temperatur des Systems in Richtung Gleichgewichtstemperatur korrigiert, dies erfolgt mit einer der beiden folgenden Methoden. Entweder durch die sehr radikale `Thermalize()` Methode nach Woodcock, die einfach die Temperatur auf die Gleichgewichtstemperatur setzt, oder die Methode `ThermalizeBer()`, die zur Anpassung der Temperatur das Berendsen-Verfahren verwendet (vgl. Abschnitt 4.3). Die Equilibrierung läuft nach dem selben Prinzip ab, zur Simulation gibt es aber drei wichtige Unterschiede:

- Die Equilibrierung wird frühestens bei Erreichen der Gleichgewichtstemperatur beendet.
- Die Schrittweite ändert sich in Abhängigkeit von der Temperatur.
- Die Korrektur der Temperatur erfolgt nur mit der Methode `Thermalize()`.

Wie oben bereits erwähnt, ruft die Methode `Go()` als Erstes die Methode `Init()` auf. `Init()` bestimmt als erstes den Vorfaktor `m_dTScale`, mit dem man aus der kinetischen Energie direkt die Temperatur des Systems bestimmen kann. Die Hauptbestandteile des Vorfaktors `m_dTScale` sind zum Einen der Boltzmannfaktor k_B =`cdBoltz` und zum Anderen die Anzahl der Freiheitsgrade

(nDegOfFreedom) des Systems, die sich aus der Summation der Freiheitsgrade der einzelnen Moleküle (CMoleculeSim::DegreeOfFreedom()) ergeben. Da das System in der Simulationsbox fixiert ist, verringert sich diese Anzahl der Freiheitsgrade im Gesamtsystem für jede Raumrichtung um eins.

Für die langreichweitigen Wechselwirkungen spielt das Molekül und dessen Struktur keine Rolle, entscheidend sind einzig die Atome des Moleküls. Für eine schnelle Berechnung der Coulomb- und van der Waals-Wechselwirkungen ist somit ein schneller Zugriff auf die Eigenschaften (Attribute) der Atome essentiell. Aber auch innerhalb der Klasse CSimulation ist es wünschenswert, schnell auf einzelne Atome zuzugreifen, um zum Beispiel eine Trajektorie in eine Datei zu schreiben oder das System auf eine vorgegebene Temperatur zu bringen und zu halten. Hierfür wurden die Methoden GetAtomList() und CountAtoms() implementiert, die einen schnellen Zugriff auf die Atome erlauben, die in der Referenzliste m_mpAtList gespeichert sind. Gefüllt wird die Referenzliste in der Methode Init() nach Bestimmung des Freiheitsgrades des Systems, nur dort ist der langsame Zugriff auf die Atome über die Methoden der Klasse CMolecule notwendig. Mit Hilfe dieser Referenzliste können dann auch die Initialisierungsmethoden Init der Lenard-Jones-Wechselwirkung CLenardJones und der Coulomb-Wechselwirkung CEwaldSum ausgeführt werden. Vervollständigt wird die Initialisierungssequenz durch die Berechnung der auf die Teilchen wirkenden Kräfte mit der Methode CalculateF() bzw. durch Festsetzung der Kräfte auf Null am Simulationsbeginn, da diese nicht abgespeichert werden.

Bevor in CalculateF() die Energien und Kräfte bestimmt und summiert werden können, müssen das Attribut m_dEbond zur Summation der Bindungsenergie und die Attribute der Atome zur Speicherung der Kräfte auf Null gesetzt werden. Für die Kräfte erfolgt dies in einer Schleife über alle Moleküle durch Aufruf der Methode ResetF() für jedes Molekül. Im zweiten Teil der selben Schleife werden bereits die Energien und Kräfte der intermolekularen Wechselwirkungen eines jeden Moleküls durch Aufruf der Methode CalcBond() berechnet. Im Anschluss werden die intramolekularen Wechselwirkungen durch Aufruf der Methode Calculate() aus den Instanzen m_LJ und m_dECoulomb der Klassen CLenardJones und CEwaldSum bestimmt. Sollen die Coulomb-Wechselwirkungen anstatt mit dem Ewaldverfahren mit der einfachen abstandsabhängigen Berechnung der Coulomb-Wechselwirkungen bestimmt werden, muss anstatt der Methode Calculate() die Methode CoulombCalculate() der Klasse CEwaldSum aufgerufen werden.

Wie bereits bei der Beschreibung der Methode Step() gezeigt wurde, wird die Methode CalculateF() zur Berechnung der Energien und Kräfte umschlossen von den beiden Methoden MoveA() und MoveB(), die für die Integration des Systems zuständig sind. Beide Methoden sorgen für die Bewegung der Moleküle, indem sie die Methoden CalcConstraintA() bzw. CalcConstraintB() eines jeden Moleküls aufrufen. Wobei die Methode MoveB() einen Simulationsschritt abschließt, so dass , nachdem die Moleküle ihre neuen Positionen / Geschwindigkei-

ten bekommen haben, die Zähler `m_nStep` und `m_dStep` für den Simualtionsschritt erhöht werden können.

In der Methode `CalculateF()` wurden nur die potentiellen Energien der Teilchen bestimmt, zur Bestimmung der Gesamtenergie muss in `MoveB()` außerdem die kinetische Energie mit Hilfe der Methode `Ekin()` bestimmt werden.

In `Ekin()` wird die kinetische Energie mit Hilfe der in `Init()` definierten Liste `m_mpAtList` bestimmt, hierbei wird die Masse eines Atoms über das dem Atom zugeordnete Element (`GetElement().GetMass()`) bestimmen und das Quadrat der Geschwindigkeit aus der Geschwindigkeits-Schrittweiten-Variablen (`VTS()`) berechnet. Die Summe der Produkte beider Werte über alle Atome wird dem Attribut `m_dEkin` zugewiesen. Dieser Wert in `m_dEkin` wird durch die Multiplikation mit Vorfaktoren und Division durch das Quadrat der Schrittweite in die kinetische Energie umgewandelt, wobei das Quadrat der Schrittweite über die Variable VTS() in das Attribut `m_dEkin` gelangt. Die Temperatur `m_dT` des Systems wird ebenfalls in `Ekin()` bestimmt. Diese ergibt sich aus der Multiplikation der in `m_dEkin` bestimmten kinetische Energie mit dem bereits in `Init()` bestimmten Skalierungsfaktor `m_dTScale`.

Während der Thermalisierungsphase müssen die Geschwindigkeiten der Atome nach jedem Simulationsschritt so geändert werden, dass die Temperatur (`m_dT`) nur gering von dem im Skript festgelegten Soll-Wert (`m_dTref`) abweicht. Hierfür wurden die Methoden `Thermalize()` und `ThermalizeBer()` implementiert, sie folgen dem in Abschnitt 4.3 beschriebenen Verfahren von Woodcock [4] bzw. dem Berendsen Thermostaten [43]. Beide Verfahren ändern die Geschwindigkeiten aller Atome um einen festen Skalierungsfaktor (`dSC`). Dieser Faktor bestimmt sich im ersten Fall aus Gleichung 4.44:

$$\texttt{dSC} = \text{sqrt}(\texttt{m\_dTref}/\texttt{m\_dT})$$

und führt unter Umständen zu einer sehr radikalen Änderung der Teilchengeschwindigkeiten. Im zweiten Fall wird die Stärke der Geschwindigkeitsänderung nicht nur von der Temperaturdifferenz zwischen Ist-Temperatur `m_dT` und Soll-Temperatur `m_dTref` bestimmt, sondern, wie Gleichung 4.45 zeigt, zusätzlich von einem weiteren Faktor, der im Skript festgelegten Relaxationszeit `m_dTauBer`:

$$\texttt{dSC} = \text{sqrt}(1.0 + \texttt{m\_dTimeStep}/\texttt{m\_dTauBer} * ((\texttt{m\_dTref}/\texttt{m\_dT}) - 1))$$

Die potentielle Energie und die kinetische Energie, die mit den in den letzten Abschnitten beschriebenen Methoden berechnet wurden, werden normalerweise zur weiteren Auswertung in eine Protokolldatei geschrieben. Der Name der Protokolldatei und die Schreibintervalle werden im Skript nach Angabe des Schlüsselwortes „Report“ festgelegt. Geschrieben wird die Protokolldatei mit der Methode `Report(FILE * fOut)`. Sie schreibt den Zeitschritt, die Temperatur, die kinetische Energie, die van der Waals-Wechselwirkungsenergie, die Coulomb-Wechselwirkungsenergie, die intramolekularen Bindungsenergien sowie die gesamte potentielle Energie und die Gesamtenergie in die angegebene Datei.

9.4.11 CTrajectory

Um die Phasenraumpunkte einer Simulation als Trajektorie abzuspeichern sind zwei unterschiedliche Verfahren implementiert. Das hier beschriebene Verfahren ist, im Gegensatz zum im nächsten Kapitel beschriebenen, das effektivere und neuere Verfahren, allerdings können das Analyseprogramm „SiDAn“ und das Visualisierungsprogramm „TrajPlay“ noch nicht auf die hiermit erzeugten Daten zugreifen.

Das Erstellen und Lesen einer Trajektorie lässt sich in zwei Phasen aufteilen. In der Initialisierungsphase werden zunächst durch Aufruf der Methode `AddVec(CVec3 *pVec)` Referenzen auf die Geschwindigkeits- und Ortsvektoren aller Teilchen im lokalen Attribut `m_mpVec` gespeichert. Hierzu wird die Liste `m_mpVec` bei jedem Aufruf von `AddVec(CVec3 *pVec)` um ein Element erweitert, dem dann die Referenz auf den Vektor zugewiesen wird.

Zum Schreiben von Trajektorien wird in der Initialisierung die Methode `InitWrite(CFileIO *pfIO, double dBoxSize, double dLJCutOff, double dCCutOff, double dKappa, double dDiel, WORD nKmax, WORD nFreqTraj)` aufgerufen. Diese enthält neben einer Referenz auf die geöffnete Datei, die in einer Instanz der Klasse `CFileIO` gespeichert ist, eine Liste von Parametern, die das zu simulierende System charakterisieren. Sie wird zuerst in `m_pfIO` eine lokale Kopie der Referenz auf die Datei erzeugen, damit diese Datei später nicht bei jedem Schreibvorgang angegeben werden muss. Im Kopf der Trajektorien-Datei wird eine eindeutige Kennung („TRTD“) und eine Versionsangabe gespeichert, beide Werte werden hierfür in den lokalen Variablen `nID` und `nVer` abgelegt. Soll eine bestehende Trajektorien-Datei fortgesetzt werden, erkennt die Methode dies, da die Länge der geöffneten Datei größer als Null Byte ist. Dann wird die Initialisirungsroutine `InitRead` aufgerufen, um den gespeicherten Kopf der Datei mit den als Parameter übergebenen Soll-Daten zu vergleichen. Stimmt eines der Werte nicht mit den vorgegebenen Parametern überein, wird eine Ausnahmebehandlung ausgelöst, die das Programm nach einer Fehlermeldung abbricht. Wird mit `InitWrite` hingegen eine neue Trajejektoriendatei erzeugt, muss ein Dateikopf geschrieben werden. Dieser enthält durch Aufruf der Methode `WriteSwap()` als erstes eine Kennung über das verwendete Abspeicherungsformat (LSB oder MSB), die ergänzt wird um die Kennung, die Version, die Anzahl der verwendeten Vektoren und die an `InitWrite` übergebenen Parameter.

Die Methode `InitRead` liest den Dateikopf einer Trajektorien-Datei aus. Hierzu wird zuerst, wieder in `m_pfIO` eine lokale Kopie der Referenz auf die Datei erzeugt und die eindeutige Kennung, so wie die Version in den lokalen Variablen `nID` und `nVer` abgelegt. Der Aufruf der Methode `ReadSwap()`, dem Gegenstück zu `WriteSwap()`, bestimmt für die Klasse `CFileIO` das verwendete Abspeicherungsformat (LSB oder MSB). Es legt fest, ob die eingelesen Daten vor der Weiterverarbeitung „binär gedreht“ werden müssen. Die mit `InitWrite` gespeicherten Daten werden anschließend ausgelesen, wobei eine falscher Kennung, Versionsnummer

oder Größe der Liste der Vektoren eine Ausnahmebehandlung ausgelöst, die das Programm wiederum nach einer Fehlermeldung abbricht.

In der zweiten Phase werden die eigentlichen Daten der Trajektorie transferiert. Gespeichert werden die Daten durch Aufruf der Methode `Write`, die neben den in der Liste gespeicherten Vektoren auch den Simulationszeitpunkt, die Temperatur und die Energien speichert. Diese mit `Write` gespeicherten Daten werden mit der Methode `Read` zur Analyse wieder ausgelesen.

9.4.12 CMolDynIO

Das zweite Verfahren wurde in der Klasse `CMolDynIO` implementiert, es erzeugt und verarbeitet Konfigurations- und Trajektoriendateien, die kompatibel mit dem alten Fortran 77 Code sind. Den Aufbau der Konfigurationsdatei gibt die Struktur `TMolConf` aus den Abbildungen 9.2 und 9.3 auf den Seiten 122 und 123 wieder. In der Konfigurationsdatei werden globale Parameter wie Systemgröße (`boxl`), Anzahl von Atomen (`ncont, nconth, nwatom, nacmol`), Temperatur (`tempe`), Zeitschritt (`nstp`) und Energien (`enbc, enbch, eptot, eptoth, ekin, ekins, epts`), aber auch Atomspezifische Eigenschaften wie Teilchenart (`nat`), Position (`xm, ym, zm`) und Geschwindigkeit (`vx, vy, vz`), so wie Listen über intramolekulare Bindungen (`li12, li13, li14, li14i`) gespeichert.

Die Konfigurationsstruktur wird nach der Initialisierung des Systems, direkt vor dem Beginn der Simulationsphase, durch Aufruf der Methode `SaveConfig(LPCTSTR sFile)` gespeichert. Vor der Speicherung muss die Anzahl der Teilchen aus den einzeln vorgegebenen Komponenten berechnet und in `m_cnAtom` gespeichert werden. Darüber hinaus muss die Länge der Listen für die Ortskoordinaten `m_mfPosX, m_mfPosY, m_mfPosZ` und Geschwindigkeit `m_mfVX, m_mfVY, m_mfVZ` der Teilchen auf die vorhandene Anzahl Teilchen angepasst und deren Koordinaten aus den Listen `m_Conf.xm`, `m_Conf.ym` und `m_Conf.zm` in `m_mfPosX`, `m_mfPosY`, `m_mfPosZ` eingetragen werden. Die eigentliche Speicherung der Konfigurationsstruktur `m_Conf` übernimmt die Methode `WriteSW` der Klasse `CFileIO`, die wiederum automatisch jede Variable im richtigen Format in die Datei speichert. Zugriff auf die Konfigurationsstruktur `m_Conf` und somit Zugriff auf deren Elemente bekommt man mit Hilfe der Methode `Config()`, die eine Referenz auf `m_Conf` zurück liefert.

Eine gespeicherte Konfigurationsstruktur `m_Conf` kann mit der Methode `LoadConfig(LPCTSTR sFile)` wieder aus einer Datei ausgelesen werden. Die Methode `ReadSW` füllt die Struktur `m_Conf` mit den gespeicherten Werten. Anhand der Feldvariablen `m_Conf.Kmax` wird dann ermittelt, ob die Daten in diesem Format abgespeichert wurden, oder die Daten noch einmal unter Vertauschung der Reihenfolge der Bytes eingelesen werden müssen. Wie in der Methode `SaveConfig(LPCTSTR sFile)` wird auch hier die Anzahl der Teilchen in `m_cnAtom` bestimmt und die Länge der Listen für die Ortskoordinaten und Geschwindigkeiten der Teilchen neu festgelegt. Abschließend wird die Lage der Teilchen aus

den gespeicherten Werten in `m_Conf.xm`, `m_Conf.ym` und `m_Conf.zm` wieder in die Listen `m_mfPosX`, `m_mfPosY` und `m_mfPosZ` zurück gespeichert.

Die Trajektorie einer Simulation wird in dieser Version in einer eigenen Datei gespeichert. Die Methode `Open` dieser Klasse liest über die Methode `LoadConfig(LPCTSTR sFile)` die benötigten Daten aus der Konfigurationsdatei und öffnet bzw. erzeugt, in Abhängigkeit vom Parameter `bReadOnly`, die Datei für die Trajektorie entweder nur zum Lesen oder sowohl zum Lesen als auch zum Schreiben. Die Größe der Datei lässt auf die Anzahl `m_cnRecord` der gespeicherten Phasenraumpunkte wie folgt schließen: Da jede Variable vom Typ „float“ vier Bytes benötigt, ergibt sich bei sechs Phasenraumkoordinaten eine Belegung von 24 Byte pro Teilchen, hinzu kommen 15 weitere Variablen vom Typ „long“ mit ebenfalls je vier Bytes. Des weiteren wird in dieser Methode die Schrittweite zwischen den Phasenraumpunkten bestimmt, hierzu wird der Zeitpunkt des zweiten gespeicherten Phasenraumpunktes vom Zeitpunkt des ersten Phasenraumpunktes abgezogen.

Zur Speicherung eines Phasenraumpunktes wurde die Methode `SaveData(DWORD nTimePos, long nClock)` implementiert. Sie speichert einen Phasenraumpunkt zum Zeitschritt `nClock` auf der Position `nTimePos` in die Datei. Befindet sich die Position am Ende der Datei, so wird ein neuer Zeitpunkt an das Ende angehängt, wobei sich natürlich auch die Anzahl der Phasenraumpunkte `m_cnRecord` erhöht. Nachdem in der Datei auf die durch `nTimePos` festgelegte Position gewechselt wurde, werden der Zeitschritt `nClock`, so wie Koordinaten und Geschwindigkeiten aller Teilchen in die Datei geschrieben. Zu beachten ist hierbei, dass wegen der Kompatibilität zu Fortran 77, deren Konventionen eingehalten werden müssen, d. h. dass vor und hinter jedem zu schreibenden Block dessen Größe mit Hilfe der Hilfsvariablen `d0` bis `d6` gespeichert werden müssen. Jeder Schreibvorgang wird mit dem Aufruf der Methode `Flush()` abgeschlossen, diese erzwingt das Schreiben der Daten auf den Datenträger. Die Methode `LoadData(DWORD nTimePos)` liest einen gespeicherten Phasenraumpunkt wieder aus. Hierbei wird der Dateizeiger auf die durch `nTimePos` bestimmte Position geschoben und die Daten in der gleichen Reihenfolge wieder ausgelesen, wie sie gespeichert wurden.

Bevor ein Phasenraumpunkt gespeichert wird, müssen Ort und Geschwindigkeit jedes Teilchens in entsprechende lokale Attribute hinterlegt werden. Hierfür wurden die Methoden `SetPos(int nItem, const CVec3 &vPos)` und `SetV(int nItem, const CVec3 &vV)` in die Klasse `CMolDynIO` implementiert. Beide Methoden weisen den gewählten Listenelementen `m_mfPosX[nItem]`, `m_mfPosY[nItem]` und `m_mfPosZ[nItem]` bzw. `m_mfVX[nItem]`, `m_mfVY[nItem]` und `m_mfVZ[nItem]` die jeweiligen Feldelemente der übergebenen Vektoren `vPos` bzw. `vV` zu. Auf die Position bzw. Geschwindigkeit kann mit Hilfe der Methoden `GetPos(int nItem)` und `GetV(int nItem)` wieder zugegriffen werden. Zum Auslesen gibt es weitere Methoden, wie `CountAtoms()` für die Anzahl der Atome, `Records()` für die Anzahl der Phasenraumpunkte, `Clock()` und `ClockPS()`

für den Zeitpunkt der gespeicherten Daten oder `TimeStep()` für den Abstand zwischen den Abspeicherungen.

Kapitel 10

Ausblick

Das hier vorgestellte Simulationssystems stellt eine lauffähige Implementierung dar, die die wesentlichen, aber noch nicht alle Aspekte eines Simmulationssystems umfasst und sich weiter optimieren lässt.

Das Konstruktionsprogramm „StructC“ baut nach einem in Skripten vorgegebenen Plan das zu simulierende System auf. Jedes Molekül wird dabei zufällig im Raum platziert, so dass die Lage des ersten Teilchens immer zufällig gewählt wird. Der Rest des Moleküls wird dann mit Hilfe einer modifizierten Z-Matrix und den internen Kraftfeldparametern der Paarbindung, der Winkelbindung und der Dihedralbindung aufgebaut, wobei prinzipiell statt einer Z-Matrix auch die Verwendung eines SMILES-String implementiert werden könnte. Ergänzt werden könnte dieses Konstruktionsprogramm durch die Festlegung des Massenschwerpunkts einiger Moleküle oder durch die Einbeziehung der langreichweitigen Wechselwirkungen, d. h. durch Berücksichtigung des van der Waals- und des Coulomb-Potentials. Eine optionale Einbeziehung von Wasserstoffbrückenbindungen in den Konstruktionsprozess könnte, bei gleichzeitiger Implementierung in „dynC“, die Equilibrierungsphase weiter verkürzen. Beim Aufbau der Startstruktur bleiben die Geschwindigkeiten zur Zeit unberücksichtigt, sie werden auf Null festgelegt. Eine Maxwell-Verteilung der Geschwindigkeiten der Moleküle (nicht der Teilchen) könnte zu einer realistischeren Startstruktur führen.

An die Strukturbildung schließt sich der Simulationsprozess an, der sich in eine Equilibrierungsphase und eine Produktionsphase aufteilen lässt. In der Equilibrierungsphase wird das System durch Zufuhr oder Abfuhr von kinetischer Energie über einen Thermostaten auf die gewünschte Simulationstemperatur gebracht. In der anschließenden Produktionsphase werden die Trajektorien der Teilchen, des zu simulierenden Systems, in einer Datei abgespeichert. Der Thermostat korrigiert bei einer Simulation im kanonischen Ensemble die Temperatur nur noch gering. Jeder Simulationsschritt lässt sich grob in zwei Abschnitte aufteilen: der Kraftberechnung und dem Integrationsschritt. Die Kraftberechnung besteht zum Einen in der Berechnung der kurzreichweitigen Kräften, resultierend aus der Paarwechselwirkung, der Winkelwechselwirkung, der Torsionswechselwirkung und der

Wechselwirkung des uneigentlichen Öffnungswinkels, so wie den langreichweitigen Kräften, die sich aus den van der Waals-Wechselwirkungen und den mit dem Ewald-Kornfeld-Verfahren bestimmten Coulomb-Wechselwirkungen zusammensetzen. Der Integrationsschritt wendet die berechneten Kräfte auf die entsprechenden Teilchen an und berechnet die neuen Geschwindigkeiten und Orte der Teilchen, wobei unter Umständen Zwangsbedingungen, wie feste Bindungsabstände mit Wasserstoffatomen berücksichtigt werden müssen. In der Simulation wird ca. 90% der Rechenzeit für die Bestimmung das Ewald-Kornfeld-Verfahren verwendet, würde dieses durch ein effizienteres Verfahren wie der „Particle-Particle / Particle-Mesh“-Methode ersetzt, könnte sie Rechenzeit wesentlich verkürzt werden. Auch ein besserer Integrationsalgorithmus, wie etwa der „Gear-predictor-corrector“-Algorithmus [32] verringern die benötigte Rechenzeit. Die Implementierung eines Barostaten und die Bestimmung des Drucks, würden die Flexibilität des Simulationsprogramms erhöhen, und es erlauben auch im großkanonischen Ensemble zu simulieren. Eine weitere Modernisierung würde durch eine Parallelisierung des Programms auf PVM oder MPI Basis erreicht. Um die Benutzerfreundlichkeit zu erhöhen, könnte das Ausführungsskript für „dynC“ mehrere Simulationsabschnitte enthalten. Equilibrierung und Simulation würden so in einem Skript zusammengefasst und durch einen einzigen Programmaufruf ausgeführt werden.

Literaturverzeichnis

[1] Adler, B. J. ; Wainwright, T. E.: J. Chem. Phys. 27 (1957), S. 1208

[2] Adler, B. J. ; Wainwright, T. E.: J. Chem. Phys. 31 (1959), S. 459

[3] Rahman, A.: Phys. Rev. 136 (1964), S. A405

[4] Woodcock, L. V.: Chem. Phys. Lett. 10 (1971), S. 257

[5] Rahman, A. ; Stillinger, F. H.: J. Chem. Phys. 55 (1971), S. 3336

[6] Case, D. A. ; Pearlman, D. A. ; Caldwell, J. W. ; Cheatham III, T. E. ; Ross, W. S. ; Simmerling, C. L. ; Darden, T. A. ; Merz, K. M. ; Stanton, R. V. ; Cheng, A. L. ; Vincent, J. J. ; Crowley, M. ; Tsui, V. ; Radmer, R. J. ; Duan, Y. ; Pitera, J. ; Massova, I. ; Seibel, G. L. ; Singh, U. C. ; Weiner, P. K. ; Kollman, P. A.: *AMBER 6*. San Francisco: University of California, 1999

[7] van Gunsteren, W. F. ; Berendsen, H. J. C.: *Groningen Molecular Simulations (GROMOS) Library Manual*. Groningen, The Netherlands: GROMOS, 1987

[8] Brooks, B. R. ; Bruccoleri, R. E. ; Olafson, B. D. ; States, D. J. ; Swaminathan, S. ; Karplus, M.: Journal of Computational Chemistry 4 (1983), S. 187

[9] Allinger, N. L. ; Yuh, Y. H. ; Lii, J.-H.: J. Am. Chem. Soc. 111 (1989), S. 8551

[10] Lii, J.-H. ; Allinger, N. L.: J. Am. Chem. Soc. 111 (1989), S. 8566

[11] Lii, J.-H. ; Allinger, N. L.: J. Am. Chem. Soc. 111 (1989), S. 8576

[12] Allinger, N. L. ; Chen, K. ; Lii, J.-H.: J. Comp. Chem. 17 (1996), S. 642

[13] Nevins, N. ; Chen, K. ; Allinger, N. L.: J. Comp. Chem. 17 (1996), S. 669

[14] Nevins, N. ; Lii, J.-H. ; Allinger, N. L.: J. Comp. Chem. 17 (1996), S. 695

[15] Nevins, N. ; Allinger, N. L.: J. Comp. Chem. 17 (1996), S. 730

[16] Allinger, N. L. ; Chen, K. ; Katzenellenbogen, J. A. ; Wilson, S. R. ; Anstead, G. M.: J. Comp. Chem. 17 (1996), S. 747

[17] Weiner, S. J. ; Kollman, P. A. ; Case, D. A. ; Singh, U. C. ; Ghio, C. ; Alagona, G. ; Jr, S. P. ; Weiner, P.: J. Am. Chem. Soc. 106 (1984), S. 765

[18] Weiner, S. J. ; Kollman, P. A. ; Nguyen, D. T. ; Case, D. A.: J. Comp. Chem. 7 (1986), S. 230

[19] Gough, C. A. ; DeBolt, S. E. ; Kollman, P. A.: J. Comp. Chem. 13 (1992), S. 963

[20] Veenstra, D. L. ; Ferguson, D. M. ; Kollman, P. A.: J. Comp. Chem. 8 (1992), S. 971

[21] Pearlman, D. A. ; Case, D. A. ; Caldwell, J. W. ; Ross, W. S. ; Cheatham III, T. E. ; DeBolt, S. ; Ferguson, D. ; Seibel, G. ; Kollman, P.: Comp. Phys. Commun. 91 (1995), S. 1

[22] Rappé, A. K. ; Casewit, C. J. ; Colwell, K. S. ; Goddard III, W. A. ; Skiff, W. M.: J. Am. Chem. Soc. 114 (1992), S. 10024

[23] Halgren, Th. A.: J. Comp. Chem. 17 (1996), S. 490

[24] Halgren, Th. A.: J. Comp. Chem. 17 (1996), S. 520

[25] Halgren, Th. A.: J. Comp. Chem. 17 (1996), S. 553

[26] Halgren, Th. A. ; Nachbar, R. B.: J. Comp. Chem. 17 (1996), S. 587

[27] Halgren, Th. A.: J. Comp. Chem. 17 (1996), S. 616

[28] Jorgensen, W. L.: J. Am. Chem. Soc. 103 (1981), S. 335

[29] Kong, C. L.: J. Chem. Phys. 59 (1973), Nr. 5, S. 2464

[30] Waldman, M. ; Hagler, A. T.: J. Comp. Chem. 14 (1993), Nr. 9, S. 1077

[31] Delhommelle, J. ; Millie, P.: Molec. Phys. 99 (2001), Nr. 8, S. 619

[32] Allen, M. P. ; Tildesley, D. J.: *Computer Simulation of Liquids*. Oxford : Clarendon Press, 1987

[33] Haberlandt, R. ; und G. Peinel und K. Heinzinger, S. F.: *Molekulardynamik - Grundlagen und Anwendungen*. Braunschweig : Vieweg, 1994

[34] Hautman, J. ; Klein, M. L.: Mol. Phys. 75 (1992), S. 379

[35] Netz, P. A.: *Computerexperimente zur Struktur, Dynamik und Diffusion in Polyacrylamidgelen*. Bielefeld, Universität Bielefeld, Diss., 1997

[36] Müller-Plathe, F.: Mol. Simul. 18 (1996), S. 133

[37] Sayle, R. A. ; Milner-White, E. J.: Trends Biochem. Sci. 20 (1995), S. 374

[38] Verlet, L.: Physical Review 159 (1967), S. 98

[39] Hockney, R. W. ; Eastwood, J. W.: *Computer simulation using particles*. New York : McGraw-Hill, 1981

[40] Swope, W. C. ; Andersen, H. C. ; Berens, P. H. ; Wilson, K. R.: J. Chem. Phys. 76 (1982), S. 637

[41] Ryckaert, J. P. ; Ciccotti, G. ; Berendsen, H. J. C.: J. Comp. Phys. 23 (1977), S. 327

[42] Andersen, H. C.: J. Comp. Phys. 52 (1983), S. 24

[43] Berendsen, H. J. C. ; Postma, J. P. M. ; van Gunsteren, W. F. ; DiNola, A. ; Haak, J. R.: J. Chem. Phys. 81 (1984), S. 3684

[44] Evans, D. J.: J. Chem. Phys. 78 (1983), S. 3297

[45] Haile, J. M. ; Gupta, S.: Journal of Chemical Physics 79 (1983), S. 3067

[46] Evans, D. J. ; Morriss, G. P.: Comp. Phys. Rep. 1 (1984), S. 297

[47] Hoover, W. G.: Physical Review A 31 (1985), S. 1695

[48] Tanaka, T.: Sci. Am. 244 (1981), S. 110

[49] Hirokawa, Y. ; Tanaka, T.: J. Chem. Phys. 81 (1984), S. 6379

[50] Tanaka, T.: Phys. Rev. Lett. 40 (1978), S. 820

[51] Suzuki, A. ; Tanaka, T.: Nature 346 (1990), S. 345

[52] Bergmann, L. ; Schaefer, C. ; Raith, W. (Hrsg.): *Lehrbuch der Experimentalphysik*. Bd. 5. Berlin : de Gruyter, 1992

[53] Berendsen, H. J. C. ; Grigera, J. R. ; Straatsma, T. P.: J. Phys. Chem. 91 (1987), S. 6269

[54] Gay, J. G. ; Berne, B. J.: J. Chem. Phys. 74 (1981), S. 3316

[55] Allen, M. P. ; Brown, J. T. ; Warren, M. A.: J. Phys.: Condens. Matter 8 (1996), S. 9433

[56] Wagner, W. L. ; Bennett, L.: Molecular Physics 94 (1998), S. 571

[57] Krömer, G. ; Paschek, D. ; Geiger, A.: Ber. Bunsenges. Phys. Chem. 97 (1993), S. 1188

[58] Yakovenko, S. Y. ; Muravski, A. A. ; Krömer, G. ; Geiger, A.: Mol. Phys. 86 (1995), S. 1099

[59] Sandström, D. ; Komolkin, A. V. ; Maliniak, A.: J. Chem. Phys. 106 (1997), S. 7438

[60] Isermann, Rolf: *Identifikation dynamischer Systeme*. Bd. 1. Berlin, New York : Springer, 1988

[61] Haile, J. M.: *Molecular dynamics simulation: elementary methods*. New York : Wiley, 1992

[62] Debye, P.: *Polar Molecules*. New York : Dover Publications, 1929

[63] Nyquist, H.: Trans. AIEE 47 (1928), S. 617

[64] Shannon, C. E.: Proc. Institute of Radio Engineers 37 (1949), S. 10

[65] Luzar, A. ; Chandler, D.: Journal of Chemical Physics 98 (1993), S. 8160

[66] Tamai, Y. ; Tanaka, H. ; Nakanishi, K.: Macromolecules 29 (1996), S. 6750

[67] Oldiges, Ch. ; Tönsing, T. ; Netz, P. A. ; Eimer, W.: Ber. Bunsenges. Phys. Chem. 102 (1998), S. 1620

[68] Oldiges, Ch. ; Tönsing, T.: Phys. Chem. Chem. Phys. 2 (2000), S. 5630

[69] Oldiges, Ch. ; Wittler, K. ; Tönsing, T. ; Alijah, A.: J. Phys. Chem. A 106 (2002), S. 7147

[70] Tönsing, T. ; Oldiges, Ch.: Phys. Chem. Chem. Phys. 3 (2001), S. 5542

[71] Oldiges, Ch. ; Tönsing, T.: Phys. Chem. Chem. Phys. 4 (2002), S. 1628

[72] Bronstein, I. N. ; Semendjajew, K. A. ; Grosche, G. (Hrsg.) ; Ziegler, V. (Hrsg.) ; Ziegler, D. (Hrsg.): *Taschenbuch der Mathematik*. 24. Thun und Frankfurt/Main : Harri Deutsch, 1989

[73] Filon, L. N. G.: Proceedings of the Royal Society of Edinburgh A 49 (1928), S. 38

[74] Berendsen, H. J. C. ; Postma, J. P. M. ; van Gunsteren, W. F. ; Hermans, J.: *Interaction Models for Water in Relation to Protein Hydration.* D. Reidel Publishing Company, 1981, S. 331

[75] Jorgensen, W. L. ; Chandrasekhar, J. ; Madura, J. ; Impey, R. W. ; Klein, M. L.: J. Chem. Phys. 79 (1983), S. 926

[76] Stillinger, F. H. ; Rahman, A.: J. Chem. Phys. 60 (1974), S. 1545

[77] Mahoney, M. W. ; Jorgensen, W. L.: J. Chem. Phys. 112 (2000), S. 8910

[78] Tanaka, N. ; Matsukawa, S. ; Kurosu, H. ; Ando, I.: Polymer 39 (1998), S. 4703

[79] *CD-ROM: msdn Library.* Redmond, Unterschleißheim : Microsoft, 2000

[80] *Internet: msdn Library.* http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvsgen/html/HungaNotat.asp

[81] Stöcker, H. (Hrsg.): *Taschenbuch mathematischer Formeln und moderner Verfahren.* 4. Thun und Frankfurt/Main : Harri Deutsch, 1999

[82] Ciccotti, G. ; Ryckaert, J. P.: Comput. Phys. Rep. 4 (1986), S. 345

[83] Miyamoto, S. ; Kollman, P. A.: J. Comp. Chem. 13 (1992), S. 952

[84] Abramowitz, M. (Hrsg.): *Handbook of mathematical functions.* 9th. New York : Dover Publications, 1970