(Ihre Lösung ist bis zum Mittwoch *09.01 2013 einzureichen. Abgabe per E-Mail an andre@lusi.uni-sb.de*)

Molekulardynamik-Simulation

Präsenzübungen/Einführung:

Das Programm *mdbasic* simuliert ein mikrokanonisches Ensemble von Soft-Core-Teilchen in zwei Dimensionen mit periodischen Randbedingungen. Die Simulationsparameter werden zu Laufzeitbeginn aus der Parameterdatei *mdbasic.in* eingelesen. Die Einheiten wurden entdimensionalisiert, d.h $m = \sigma = \epsilon = 1$:

$$u(r_{ij}) = 4\epsilon \left(\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right) + \epsilon \qquad r_{ij} \leq r_c = 2^{1/6}\sigma$$

(a) Die Programmdateien befinden sich unter */home/comphys/06_MD/*. Erstellen Sie gegebenenfalls einen Ordner in Ihrem Homeverzeichnis und kopieren Sie alle Dateien dieser Übung dort hin.

(b) Öffnen Sie die Quellcodedatei *mdbasic.c* und versuchen Sie die Funktionsweise grob zu überblicken.

(c) Kompilieren Sie das Programm (./compile.sh). Da zur Bildgenerierung eine exteren Bibliothek verwenden wird, benötigen Sie die Zusatzoptionen *-I/usr/include/cairo -lcairo*, sowie die obligartorische Mathematikbibliothek *-lm*. Damit die Parameterdatei eingelesen wird, muss das kompilierte Programm *mdbasic* heißen. Führen Sie das Programm anschließend aus.

(d) Es wurde eine Bildausgabe implementiert, die Schnappschüsse der Konfigurationen zu regelmäßigen Zeiten (alle *stepSnap* Schritte) in den Ordner *moviedata* speichert. Um einen Film aus den erzeugten Bildern eignet sich der *mencoder* (./encode.sh). Die erzeugte Videodatei *output.avi* lässt sich beispielsweise mit dem *mplayer* wiedergeben. Bevor neue Bilder durch das Programm generiert werden, sollten alle Dateien im Ordner *moviedata* gelöscht werden, falls daraus ein Film erzeugt werden soll.

(e) Die Variable *rCut* beschreibt die Länge, ab der das Lennard-Jones Potenial abgeschnitten wird. Da das Programm zur Zeit Soft-Core-Teilchen simuliert, wurde *rCut* genau so gewählt, dass es dem Mininum des Potentials entspricht und so der attraktive Teil weggeschnitten wurde (siehe oben).

Ändern Sie das Programm nun so ab, dass die Abschneidelänge aus der Parameterdatei eingelesen wird.

Aufgaben:

(Bitte für jede Aufgabe jeweils die Quelltextdatei und die Inputdatei abgeben und ggf. Diagramme.)

1. [*3 Punkte*] Repulsive Randbedingungen

Implementieren Sie Randbedingungen, die die Teilchen mit dem gleichen Potential abstößt, wie die Teilchen untereinander.

(a) In der Headerdatei *in_proto.h* wurde bereits die Funktion *void ApplyBoundaryCondWall ()* deklariert. Schreiben Sie den entsprechenden Funktionsrumpf in die Quelledatei *mdbasic.c* und rufen Sie sie an einer geeigneten Stelle innerhalb der Funktion *void SingleStep ()* aus.

(b) Führen Sie das Programm aus und erzeugen Sie anschließend einen Film um sich zu überzeugen, dass die Erweiterung korrekt funktioniert.

(c) Schreiben Sie die Funktion *void ApplyGravity ()* die allen Teilchen in jedem MD-Schritt zusätzlich eine konstante Beschleunigung in positiver y-Richtung auferlegt und schauen Sie sich das Ergebnis der Simulation an.

Tipp:

- Benutzen Sie Kräfteberechnung in der Funktion *void ComputeForces()* als Vorlage.
- Die Positionen der Wände sind: $x = \pm\frac{1}{2} \cdot \text{region}.x$ und $y = \pm\frac{1}{2} \cdot \text{region}.y$.

2. [*3 Punkte*] **Andersen-Thermostat**

In Experimenten wird in der Regel die Temperatur festgehalten ((N,V,T)-Ensemble) und nicht die Innere Energie. Um auch die kanonischen Ensemble mittels MD-Simulationen zu untersuchen, bedient man sich sog. Thermostate, die das System zu der gewünschten Temperatur hin thermalisiert. Erweitern Sie das Programm um einen sogenannten Andersen-Thermostat und benutzen Sie wieder periodische Randbedingungen.

(a) Fügen Sie einen Parameter in das Programm ein, der bestimmt, nach wie vielen Steps eine virtuelle Teilchenkollision vorgenommen wird (z.B *stepTherm*) und erweitern sie das Programm entsprechend. Benutzen Sie auch hier die Parameterdatei, anstatt dem Parameter direkt im Programm einen Wert zuzuweisen.

(b) Schreiben Sie die Funktion *void Thermalize ()*, die ein zufälliges Teilchen des Ensembles auswählt und setzen Sie dessen Geschwindigkeitskomponenten entsprechend einer Normalverteilung mit einer Halbwertsbreite, die der Quadratwurzel der Temperatur entspricht, neu. Die Zufallsgeneratoren: *RandInt(int)* und *RandGauss(double)* sind bereits implementiert.

(c) Stellen die sicher, dass die Funktion *void Thermalize ()* nur alle *stepTherm* ausgerufen wird, indem Sie den zeitlichen Verlauf der Temperatur für verschiedene *stepTherm* plotten und interpretieren Sie den Verlauf.

3. [*4 Punkte*] **Paarkorrelationsfunktion/Radial distribution function (Rdf)**

Ein wichtiges Hilfsmittel um Informationen über die Phase des Systems zu bekommen ist Paarkorrelationfunktion:

$$g(r_n) = \frac{Ah_n}{\pi N_a^2 r_n \Delta r},$$

wobei:

A: Systemfläche
h_n: normiertes Histogramm (siehe Aufgabenteil a))
N_a: Teilchenanzahl

$$\Delta r = \frac{\text{maxRangeRdf}}{\text{sizeHist}}$$

$$r_n = (n - \frac{1}{2})\Delta r$$

Die Rdf beschreibt die Dichteverteilung in Abhängigkeit vom Abstand eines bestimmten Teilchens.

(a) Berechnen Sie mit Hilfe eines Histogramms h_n die Paarkorrelationsfunktion im kanonischen Ensemble. Das Histogramm soll die Anzahl der Teilchenpaare mit einem Abstand r im Intervall $(n-1)\Delta r \leq r < n\Delta r$ speichern. Beachten Sie, dass Sie dem System eine ausreichende Zeit zum äquilibrieren lassen, bevor die mit der das Histogramm füllen und mitteln Sie dann über mehrere Zeiten, um eine aussagekräfige Rdf zu erhalten.

(b) Geben Sie die Daten in eine Datei aus. Zum Erstellen einer Ausgabedatei können Sie genauso vorgehen, wie es für die Datei *output.dat* bereits im Quellcode geschehen ist. Variieren Sie die Simulationsparameter und plotten Sie anschließend die Paarkorrelationsfunktion für die drei Aggregatzustände und zeigen Sie entsprechend, wenn keine Korrelation besteht, nur im Nahbereich und im Nah- und Fernbereich.

Liste hilfreicher Makros und Variablen, die im Quellcode verwendet werden:

nMol:	Anzahl der Teilchen
DO_MOL:	*for(n = 0; n < nMol; n ++)*
mol[i].r:	Ortsvektor des i-ten Teilchens
mol[i].rv:	Geschwindigkeisvektor des i-ten Teilchens (ra statt rv für Beschleunigung entsprechend)
mol[i].r.x:	x-Komponente des Ortes des i-ten Teilchens
region.x:	Systemlänge in x Richtung (y statt x entsprechend)

Infos und aktuelle Übungsblätter finden Sie unter: http://www.uni-saarland.de/fak7/rieger/homepage/teaching.html
Bei Fragen E-Mail an: andre@lusi.uni-sb.de