Programmieren einer Partikelsimulation für kurzreichweitige Interaktionen

Projektbericht

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Vorgelegt von:	Oliver Heidmann, Benjamin Warnke
E-Mail-Adresse:	adresse@email.de, 4bwarnke@informatik.uni-hamburg.de
Matrikelnummer:	1234567, 6676867
Studiengang:	Informatik
Betreuer:	Philipp Neumann

Hamburg, den 09.03.2017

Abstract

Ziel des dieses Projektes ist die Implementierung einer Partikel-Simulation für kurzreichweitige Partikel-Interaktionen mit Autotuning.

Contents

1 Einleitung

Bei Partikel Simulationen müssen normalerweise die Wechselwirkungen zwischen jedem möglichen Partikelpaar berechnet werden. Die Laufzeit des Programms steigt quadratisch zur Anzahl der Partikel. Dies ist besonders bei hohen Anzahlen von Partikeln kritisch für die Laufzeit. Die in diesem Projekt implementierte Partikel-Simulation ist für kurzreichweitige Interaktionen optimiert. Dadurch lassen sich die Interaktionen zwischen weit auseinanderliegenden Partikeln vernachlässigen, wodurch die Laufzeit kürzer werden kann. Es gibt verschiedene Möglichkeiten, die Interaktionen auf die kurzreichweitige Interaktion zu beschränken um das Programm zu beschleunigen. Das Problem bei diesen verschiedenen Möglichkeiten der Beschleunigung besteht darin, dass je nach Eingabe eine andere Art der Vereinfachung eine bessere Programm-Laufzeit ermöglicht. Die Besonderheit dieser Partikel Simulation liegt darin, dass das Programm zu beginn selbst entscheiden kann, welche Optimierungsstrategie für die gegebene Eingabe am sinnvollsten ist. Hieraus resultiert der Vorteil gegenüber anderen Programmen, dass die Laufzeit für jede beliebige Eingabe besonders schnell ist, und nicht nur wenn die Eingabe passend ist. Dies ist besonders dann spannend, wenn man selbst nicht sicher ist, welches verfahren für die Eingabe am besten geeignet ist, ohne selbst vorher alle Möglichkeiten einmal auszuprobieren.

2 Realisierung

2.1 Vorgehen

planung aufteilung + zeitplan -> realität zeiteinschätzung vs reale zeit unerwartete schwierigkeiten

2.2 Design

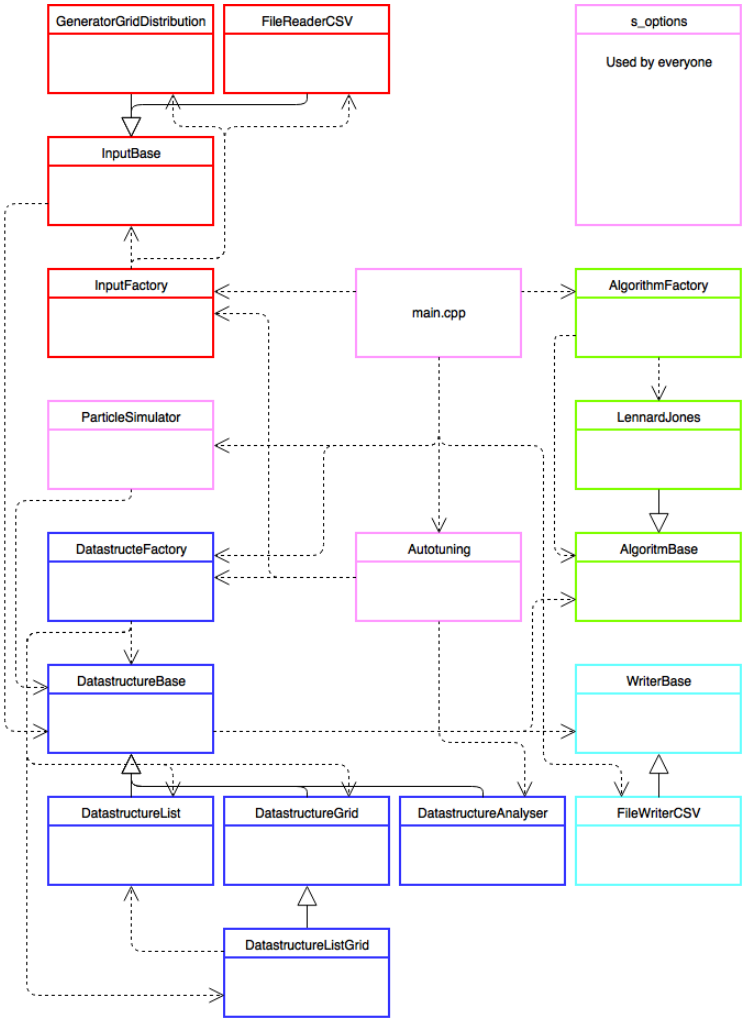

Figure 2.1: Klassendiagramm

welche klasse tut was für spätere programierer erklären

2.3 Implementierung

beschreiben der benutzung(parameter)

2.4 Analyse

2.4.1 Korrektheit

Um sicherzustellen, dass das Programm korrekte Ausgaben liefert, wurden verschiedene Arten von Test durchgeführt. Zum einen wurden einzelne Komponenten getestet, zum

anderen wurde auch das Gesamtverhalten des Programms überprüft.

- **unit-Tests** Um die Korrektheit von den einzelnen Komponenten des Programms zu gewährleisten wurden unit-Tests eingesetzt. Schon beim schreiben der unit-tests wurden viele Fehler gefunden und behoben. Bei späteren Refactoring-Maßnahmen wurden durch diese Testfälle viele neue Fehler verhindert.

- **Energieerhaltung** Aus dem Physikalischen Gesetz 'Aktion gleich Reaktion' ergibt sich, dass die Energie in einem geschlossenen System immer erhalten belieben muss. Eingebaute Funktionen im Programm ermöglichen die Ausgabe der aktuellen Energie im System, sodass leicht überprüft werden kann, ob die Energie in einem Akzeptablen Rahmen beleibt. Durch Rechenungenauigkeiten ist es sehr unwahrscheinlich, dass die Energie exakt gleich beleibt.

- **Visualisierung** Durch Visualisieren der Ausgabedaten wurde auch das fertige Ergebnis stichprobenartig überprüft. Bei den getesteten Parametern verhielt sich das Programm wie erwartet.

2.4.2 Laufzeit

Die Laufzeit des Programmes hängt von der Anzahl der zu simulierenden Partikel ab. Bei kurzreichweitigen Interaktionen kann man die Laufzeitabschätzung mit $p \cdot \frac{r^3}{V} \sim O(1)$ vereinfachen (siehe Tabelle 2.1).

	Basic	Nachbar-Listen	Linked-Cells	Kombination
Aufbau (Linked-Cells)	-	-	$\Theta(p)$	$\Theta(p)$
Aufbau (Nachbar-Listen)	-	$\Theta(p^2)$	-	$O\left(p^2 \cdot \frac{27 \cdot r^3}{V}\right)$ $\sim O(p \cdot 27)$
Iteration	$\Theta(p^2)$	$O\left(p^2 \cdot \frac{\frac{4}{3}\pi \cdot r^3}{V}\right)$ $\sim O\left(p \cdot \frac{4}{3}\pi\right)$	$O\left(p^2 \cdot \frac{27 \cdot r^3}{V}\right)$ $\sim O(p \cdot 27)$	$O\left(p^2 \cdot \frac{\frac{4}{3}\pi \cdot r^3}{V}\right)$ $\sim O\left(p \cdot \frac{4}{3}\pi\right)$

Table 2.1: Laufzeitvergleich der Datenstrukturen

Je nachdem, welches Verfahren gewählt wird, ist die Laufzeit unterschiedlich. Bei der Variante bei der nur Nachbar-Listen verwendet werden ist der Aufwand diese Listen aufzubauen Quadratisch zur Anzahl der Partikel. Dies ist so ungünstig, dass es nicht empfehlenswert ist, diese Variante zu Benutzen. Die Linked-Cells-Variante benötigt wenig Zeit um die Datenstruktur aufzubauen, dafür aber wird pro Iteration viel Zeit benötigt. Die Kombinierte Variante benötigt mittelmäßig viel Zeit für den Aufbau der Datenstruktur kosten, und nur Minimale Zeit pro Iteration.

Damit das Auto-tuning entscheiden kann, welche Variante verwendet werden soll, muss das Programm abschätzen können, wie oft die Datenstruktur neu gebaut werden

muss. Es gibt mehrere Faktoren, die einen Einfluss auf die Häufigkeit des Neubauen der Datenstruktur haben.

- **Dichte** Wenn die Partikel dichter aneinander liegen, dann werden durch die nur Zellen-basierte Optimierung zu wenig Partikel frühzeitig für die Berechnung eliminiert, was zu einer stärkeren Laufzeiterhöhung führt, als wenn die Kombinierte Variante verwendet werden würde.
- **Anzahl** Wenn die Anzahl sehr gering ist, dann ist die Laufzeit des naiven Algorithmus ohne Optimierung kürzer, als eine der Optimierten Varianten, da der Naive Algorithmus nur sehr kleine Konstanten hat.
- **cut-off** Je größer der cut-off Radius gewählt wird, desto mehr Partikel befinden sich in der Nachbarschaft. Die Zellen werden hierdurch größer. Wenn der cut-off Radius relativ groß ist, dann ist der Listenaufbau auch in der Kombinierten Variante Laufzeit-intensiv, wodurch die nur Zellen basierte Variante im Vorteil ist. Da dieses Programm sich auf kurzreichweitige Interaktionen fokussiert, werden keine sehr großen cut-off-Radien auftreten.
- **cut-off Faktor** Je mehr Spielraum auf den cut-off hinzugefügt wird, desto seltener müssen die Datenstrukturen neu aufgebaut werden. Dies geht allerdings auch sehr zulasten der Laufzeit, die in den Iterationen gebraucht wird.
- **Startgeschwindigkeit** Je schneller sich die Partikel sich bewegen, desto schneller wird der cut-off Bereich verlassen. Hieraus folgt, dass die Datenstruktur besonders oft neu aufgebaut werden muss. Hier ist die nur Zellen basierte im Vorteil.
- **Startanordnung** Je nachdem wie die Partikel bei Programmstart angeordnet sind ergeben sich andere Effekte. Zum Beispiel wäre es möglich, dass sich zu beginn alle Partikel in einem kleinem Bereich des zu simulierenden Raum aufhalten. Hieraus folgt, dass einige Zellen sehr viel mehr Partikel enthalten als andere. Für die gemischte Variante würde dies Bedeuten, dass die zweite Aufbauphase in einigen Zellen sehr viel Laufzeit benötigt.

2.4.3 Auto-tuning

- **c** → cut-off-radius
- **f** → cut-off-radius-factor
- **s** → start-speed

$$2 \cdot s < c \cdot (f - 1) - 1$$

- **true** → linked-cells+verlet-list
- **false** → linked-cells

3 Zusammenfassung

4 Literatur

5 Anhang

Verwendete Bibliotheken und Programme zum ausführen des Programms

- Boost
  - unit-tests
- CMake
- Make
- clang-format
- paraview
  - Visualisierung der Ausgabe
- lcov
  - Testabdeckung ermitteln und visualisieren
- slurm
  - Messtabellen berechnen
- doxygen
  - Dokumentation des Quelltextes
- latex
  - dieses Dokument
  - Präsentationen