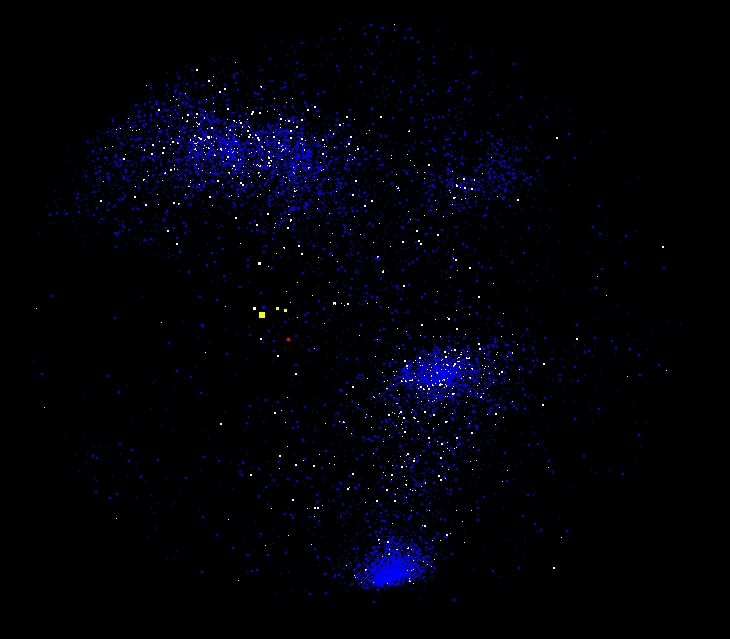
N-Body Simulation  
Sonnensystem und Galaxien



Inhaltsverzeichnis

[N-Body Simulation Sonnensystem und Galaxien 1](#_Toc153019679)

[Berechnung der Gravitationskraft 3](#_Toc153019680)

[Numerische Methoden 3](#_Toc153019681)

[Semi-implicit Euler method (2. Order) 3](#_Toc153019682)

[Runge-Kutta method (4. Order) 3](#_Toc153019683)

[Leapfrog method (2. Order) 4](#_Toc153019684)

[Vergleich der Methoden 5](#_Toc153019685)

[Barnes Hut Algorithmus 6](#_Toc153019686)

[Synchronisierung der Berechnungen 7](#_Toc153019687)

# 

# 

# Berechnung der Gravitationskraft

Um Graviton zwischen vielen unabhängigen Körpern zu berechnen muss die kraft zwischen jedem Objekt zu jedem Objekt berechnet werden, für diese N-Body Simulation gibt es keine einfache Formel die angibt wann sich welcher Körper wo befindet. Diese N-Body Problem ist ein bekanntes Problem in der Astrophysik und wird nie mathematisch perfekt lösbar sein. Da die Beschleunigung auf einen Körper nicht konstant sind numerische Methoden zur Annäherung nötig.

Um die kraft die auf einen Körper wirkt auszurechnen benutzten wir Newtons Gravitationsgesetz:

Mit der kraft die auf einen Körper wirkt können wir seine Beschleunigung ausrechnen:

### Numerische Methoden

Da die Beschleunigung a nicht konstant ist müssen wir sie numerisch integrieren. Dazu gibt es etliche Methoden wir haben die weitverbeitesten getestet und untersucht. Dabei war uns vor allem die Geschwindigkeit wichtig da wir natürlich nur begrenz Rechenleistung zur Verfügung haben.

### Semi-implicit Euler method (2. Order)

Wobei der zeitschritt ist bei dem wir annehmen die Beschleunigung sei konstant da wir nur begrenzt Rechenleistung haben.

Die Euler Methode ist die einfachste und auch die schnellste, doch sie ist auch recht ungenau was schon nach einer kurzen zeit eine extreme Auswirkung auf unsere Simulation haben kann.

### Runge-Kutta method (4. Order)

Die Runge-Kutta Methode(RK4) ist deutlich genauer doch sie ist auch deutlich rechenintensiver, was sie für unser Vorhaben ehr unpassend macht. Als wir am Anfang aber noch nur unserer Sonnen System simuliert haben war sie genau und hat zu am wenigsten Energie Verlust geführt. Da die Runge-Kutta auch noch erweiterbar ist so dass sie Ordnung 8 oder mehr hat könnten wir sie in Zukunft wieder einsetzten, wenn wir die Berechnungen an ein Rechenzentrum abgeben würden um somit genauere Ergebnisse zu bekommen.

### Leapfrog method (2. Order)

Kick drift kick (KDK):

Drift kick drift (DKD):

Die KDK-Methode ist laut unseren Auswertungen am genausten benötigt aber auch mehr Rechenleistung als die DKD-Methode da zweimal die Beschleunigung ausgerechnet werden muss. Trotzdem haben wir für die Galaxien Simulationen meistens, dass KDK-leapfrog verfahren benutzt, da es das beste Verhältnis zwischen Genauigkeit und rechen Intensivität bietet

## Vergleich der Methoden

Um die Genauigkeit der Methoden zu testen haben wir den Energie Verlust berechnet und diesen verglichen.

Die folgenden Energie Verluste wurden mit Daten von unserem Sonnensystem berechnet. Diese Daten haben wir vom NASA Horizons System, dass alle Positionen und Geschwindigkeiten von den planten in unserem Sonnensystem zu einem bestimmten Zeitpunkt bereitstellt.

Methoden bei unterschiedlichem bei gleicher gesamten zeit:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Euler: |  |  |  | Runge Kutta (RK4): | |  |
| 𝑛𝑡 | ∆𝑡 |  |  |  |  |  |
| 10 | 100000 | -8,64E+30 |  | 10 | 100000 | -2,29E+31 |
| 100 | 10000 | -1,05E+30 |  | 100 | 10000 | -2,79E+30 |
| 1000 | 1000 | -1,08E+29 |  | 1000 | 1000 | -2,84E+29 |
| 10000 | 100 | -1,11E+28 |  | 10000 | 100 | -2,88E+28 |
| 100000 | 10 | -1,39E+27 |  | 100000 | 10 | -3,17E+27 |
| 1000000 | 1 | -4,23E+26 |  | 1000000 | 1 | -6,06E+26 |
|  |  |  |  |  |  |  |
| DKD leapfrog: | |  |  | KDK leapfrog: | |  |
|  |  |  |  |  |  |  |
| 10 | 100000 | 3,30E+27 |  | 10 | 100000 | 1,90E+25 |
| 100 | 10000 | -2,51E+26 |  | 100 | 10000 | 5,22E+24 |
| 1000 | 1000 | -3,14E+26 |  | 1000 | 1000 | 7,66E+23 |
| 10000 | 100 | -3,16E+26 |  | 10000 | 100 | 6,42E+22 |
| 100000 | 10 | -3,16E+26 |  | 100000 | 10 | 6,27E+21 |
| 1000000 | 1 | -3,16E+26 |  | 1000000 | 1 | 4,72E+20 |

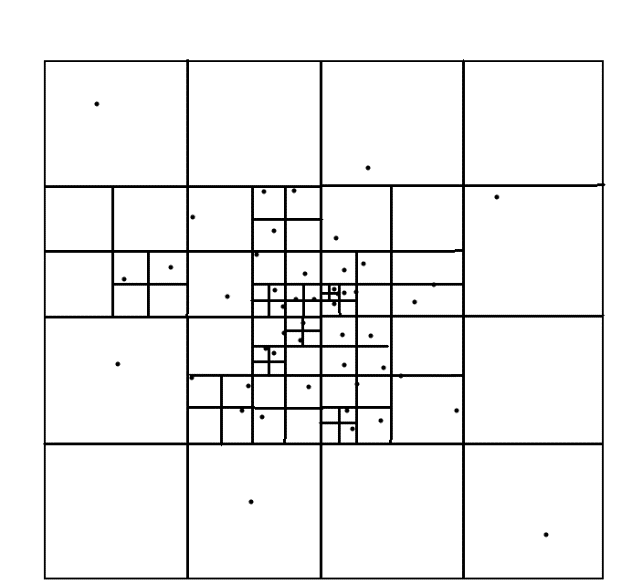
Wie in der Auswertung zu sehen schneidet KDK leapfrog am besten ab, Runge Kutta ist hier sehr ungenau, weil so groß ist, denn Runge Kutta wird erst ab einem sehr kleinem die beste Option. Nach dieser Auswertung haben wir für die Finalen Berechnungen immer KDK leapfrog als numerische Methode benutzt und zum schnellen testen Euler.

## Barnes Hut Algorithmus

Da die Gravitationskraft für jeden Körper in der Simulation mit jedem Körper berechnet werden muss ist die Laufzeitkomplexität was zur folge hat, dass größere und damit auch realistischere Mengen von Körpern zu einem quadratischen Anstieg des rechen Aufwands führen. Bei Hunderttausend Objekten und zehntausend zeitschritten wären es also schon 100 Billionen Berechnungen was natürlich mit unseren Rechnern einfach zu Lange dauern würde. Um dieses Problem zu beheben haben wir den Barnes Hut Algorithmus implementiert. Mit diesem können wir die Laufzeitkomplexität auf senken was einen riesigen unterschied macht:

Um dies umzusetzen wird der Raum in Würfel unterteilt und alle Objekte in einer Baum Struktur verteilt. Dabei wird der Würfel, im Baum also die node, so lange aufgeteilt bis auf der untersten ebene in jeder node nur ein Objekt ist. Dieser Baum wird bei jedem zeitschritt mit den neuen Objekt Positionen neu gebaut, damit die Würfel auch bei Verschiebungen die richtigen Objekte in sich haben.

Außerdem wird der massenschwerpunkt der node berechnet was die Genauigkeit des Algorithmus um ein Vielfaches verbessert.



Nachdem der Baum gebaut wurde schaut der Algorithmus für jeden Körper zu jeder Obersten Node:

= Schwellwert Theta ( desto näher zu 0 desto genauer)

r = Radius der aktuellen node

d = Distanz von Körper zu Node Massenschwerpunkt

Baum aus würfeln in 2D dargestellt

Falls

-> Berechne Kraft zur Node

Sonnst

-> gehe tiefer in die nächsten 8 nodes rein um die Genauigkeit zu erhöhen

Da der Algorithmus rekursiv ist, gab es ein Problem mit dem stack Speicher und es führte zu einem Stack Overflow. Ein Stack overflow ist ein Speicherüberlauf. Dieser entsteht, wenn sich eine Funktion zu häufig selber aufruft (Rekursion), da sich das Programm merken muss, welche Funktion welche Funktion aufgerufen hat und dies in den Stack Speicher schreibt.  
Deswegen gibt es eine maximale tiefe. Im Moment 200. Diese hat aber keinen Einfluss auf die Genauigkeit der Simulation, da sie nur im äußersten Notfall eingreift und der Distanz unterschied zu vernachlässigen ist.

# Synchronisierung der Berechnungen

Um den Prozess der Berechnungen zu beschleunigen, wurden Multithreading ins Programm eingebaut. Multithreading ist die synchrone Verteilung der Berechnungen und abzuarbeitenden Aufgaben auf alle Kerne des Prozessors. Dies beschleunigt den Vorgang der Berechnungen um die Anzahl der Kerne. Ein normaler PC hat im Durchschnitt 4-8 Kern. Ein Server hingegen hat häufig 32-128 Kerne, was die Berechnung um ein Vielfaches Beschleunigt.

Ein weiterer Ansatz wäre es die Berechnungen auf die Grafikkarte zu übertragen, da diese im Durchschnitt 10-20-mal so schnell rechnet wie ein Prozessor.  
Eine weitere Überlegung war die Berechnungen auf mehrere Computer zu verteilen. Ein Computer hätte bei diesem System die Fäden in der Hand und steuert die anderen Computer. Die Computer wären über die Netzwerk Schnittstellen miteinander verbunden.  
Beide dieser Ansätze wurden aber bis her noch nicht weiter verfolgt, da die Variante mit einem Server am schnellsten und einfachsten umzusetzen ist. Außerdem ist es schwieriger genug Computer zu bekommen, auf welchen zur gleichen Zeit gerechnet werden kann.