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 |

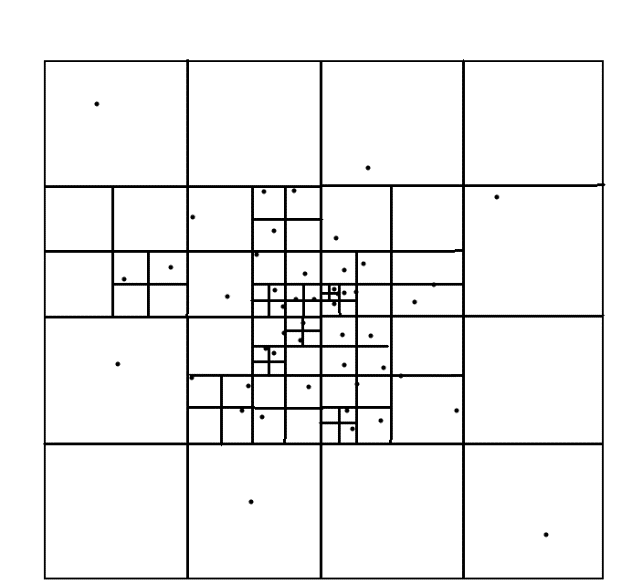
Wie in der Auswertung zu sehen schneidet KDK leapfrog am besten ab, Runge Kutta ist hier sehr ungenau, weil so groß ist, denn Runge Kutta wird erst ab einem sehr kleinem die beste Option. Nach dieser Auswertung haben wir für die Finalen Berechnungen immer KDK leapfrog als numerische Methode benutzt und 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 den 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 gibt es die Möglichkeit eines Stack Overflows, deswegen gibt es eine maximale tiefe im Moment 200 diese hat aber kein Einfluss auf die Genauigkeit der Simulation da sie nur im äußersten Notfall eingreift und der Distanz unterschied zu vernachlässigen ist.

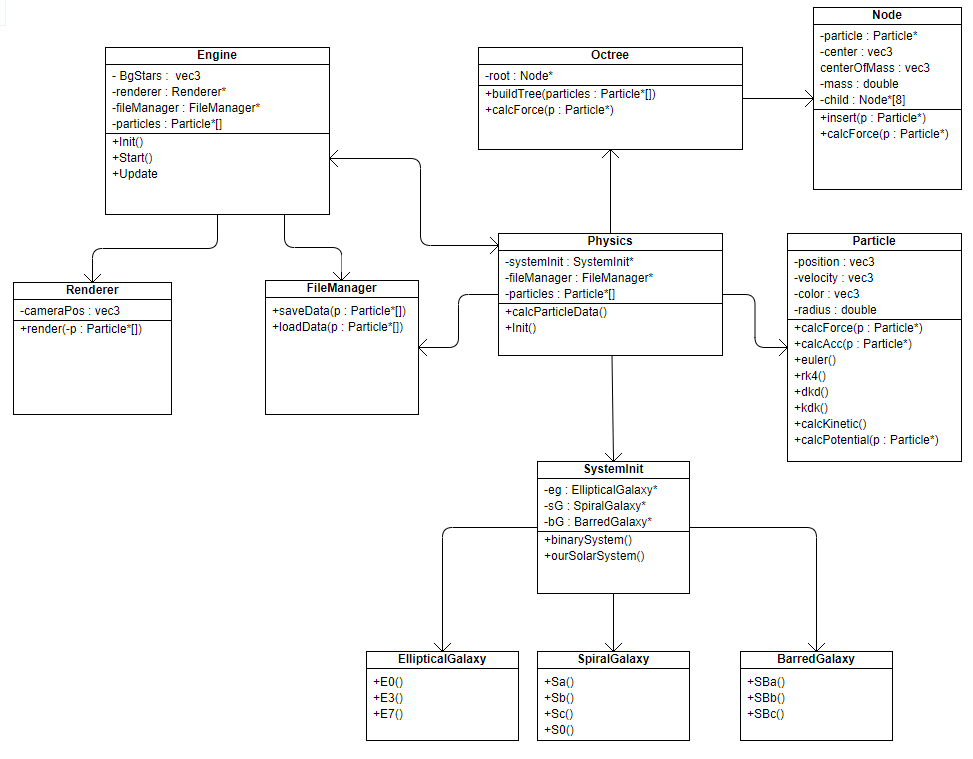
Software Struktur

Um die simulierten daten auch ansprechend anzuzeigen haben wir uns für OpenGL entschieden. Dazu haben wir eine Rendering Engine geschrieben die die Partikel in 3D raum anzeigt.

Verwendete Bibliotheken:

* GLEW (OpenGL Extension Wrangler Library) für die vereinfachte Nutzung von OpenGL Funktionen.
* GLFW (Graphics Library Framework) für Fenster und Kontext Erstellung
* GLM (OpenGL Mathematics) für die Double Vektoren und Matrizen

Vereinfachtes UML mit wichtigsten Attributen und Klassen:



Am Anfang ruft Physics SystemInit auf um einmal die Start Positionen/ Geschwindigkeiten aller Objekte festzulegen. Dabei können entweder unser Sonnensystem oder Ganze Galaxien nach Hubbles Klassifikation Ausgewählt werden.

Nach dem der Startposition berechnet Physics mit dem Barnes Hut Algorithmus(Octree) die Positionen für alle Körper und übergibt diesen dann an den File Manager dieser speichert die Daten in binär als Datei. Die Engine kann darauf hin sich die Daten zum aktuellen Zeitpunkt wieder rausladen was uns die Möglichkeit gibt die berechnete zeit vor und zurück zu spulen um die Veränderung im system besser zu verstehen. Zeitpunkt wieder rausladen was uns die Möglichkeit gibt die berechnete Zeit vor und zurück zu spulen um die Veränderung im system besser zu verstehen.

Unser Gesamte code ist auf GitHub öffentlich einsehbar: https://github.com/Philip-Spaeth/simulation

Simulation von Elliptischen Galaxien