Simulation von Gravitation im Sonnensystem

Till Wegener (28891)
Hochschule Rhein-Waal
Fakultät Kommunikation und Umwelt
Methoden und Werkzeuge der modernen Astronomie
Moers, Deutschland
till.wegener@hsrw.org

Abstract—Die Bewegung der Sterne und Planeten fasziniert die Menschheit schon seit Urzeiten. In diesem Paper wird eine Methode zur Simulation der Gravitation im Sonnensystem vorgestellt, welche physikalisch akkurate Daten verwendet, um die Bewegung der Planeten zu berechnen. Die Simulation wird in C++ implementiert und visualisiert.

Index terms—Simulation, Gravitation, Sonnensystem, C++, Visualisierung

Contents

I. Einleitung 1
II. Historischer Kontext 1
III. Mathematische Grundlagen 1
IV. Vorbereitung 2
IV.A. Datensätze 2
IV.B. Wahl der Programmiersprache 2
IV.C. Wahl von Hilfsbibliotheken 2
V. Implementierung 2
V.A. Grundlegendes 2
V.B. Laden der Datensätze 3
V.C. Aufteilung von Simulation und Visualisierung 3
V.D. Implementieren der Simulation 3
V.E. Implementieren der Visualisierung 4
VI. Probleme und Lösungen 4
VI.A. 32-Bit gleitkommazahlen 4
VI.B. Floating Origin 4
VII. Ergebnisse 4
VIII. Diskussion 4
Quellen 4

I. Einleitung

Die Planeten in unserem Sonnensystem folgen wie alle anderen Objekt im Universum den Phsyikalischen Grundgesetzen. Für die Bewegungen der Planeten ist primär die Gratitationskraft verantwortlich. Durch das verstehen dieser Kraft ist es uns möglich, die Bewegungen der Planeten zu simulieren. Dieses Paper verwendet diese Grundgestze um eine Simulation zu entwickeln, welche die Bewegungen der Planeten im Sonnensystem berechnet.

II. Historischer Kontext

Im laufe der Zeit haben viele Phsyiker und Astronomen sich mit der Bewegung der Planeten beschäftigt. Über die Jarhunderte wurden viele Modelle entwickelt welche nah am heutigen Stand der Wissenschaft sind. Im folgenden werden einige dieser Modelle vorgestellt.

1. Im 16ten Jahrhundert entwickelte Nikolaus Kopernikus mit *Über die Umlaufbahnen der Himmelssphären* [1] das erste Modell, welches den heutigen erkenntnissen nahe kommt. Er beschrieb die Bewegung der Planeten dardurch, dass alle Planeten aufeinander Gravitation ausüben.
2. Zu beginn des 17ten Jahrhundertes entwickelte Johannes Kepler mit *Astronomia nova* [2] ein Modell, welches die Bewegung der Planeten genauer beschrieb. Seine Axiome besaten, dass die Planeten sich auf Ellipsen bewegen, und körper sich gegenseitig anziehen.
3. 1644 entwickelte Isac Newton mit *Philosophiæ Naturalis Principia Mathematica* [3] das Modell, welches bis heute zur beschreibung der Gravitation verwendet wird. Seine Gravitationsgesetze beschreiben, wie sich Körper aufeinander anziehen.

III. Mathematische Grundlagen

Wie durch Newton beschreiben, wirkt auf jeden Körper eine Gravitationskraft, welche durch die Masse der Körper und den Abstand zwischen den Körpern bestimmt wird. Die Gravitationskraft zwischen zwei Körpern kann durch die Formel

$$F_G = G * \frac{m_1 * m_2}{r^2} \tag{1}$$

beschrieben werden. Hierbei ist F_G die resultierende Gravitationskraft, G die Gravitationskonstante ($6.67 * 10^{-11} \frac{m^3}{kg*s^2}$), m_1 und m_2 die Massen der Körper und r der Abstand zwischen den Körpern. Die Kraft wirket hierbei entlang der Verbindungslinie der Körper.

Für ein 2-Körper-System kann die Bewegung der Körper durch diese Formel bestimmt werden. Die Gravitationskraft welche die Körper aufeinander ausüben, kann mit der Masse dieser verwendet werden, um die Beschleunigung der Körper mit $F = m * a$ zu bestimmen.

Für 2-Körper-Systeme ist es möglich, numerische Lösungen für die Bewegung der Körper zu finden.

In mehr-Körper-Systemen wird die Bewegung der Körper durch die Summe der einzelnen wirkenden Gravitationskräfte bestimmt. Für Systeme mit 3 oder mehr Körpern ist es nicht möglich, analytische Lösungen zu finden. Wenn wir hierzu Lösungen finden wollen, müssen wir numerische Methoden verwenden.

Im folgenden wird eine Methode vorgestellt, welche Eulers Methode verwendet, um die Bewegung der Planeten im Sonnensystem zu simulieren.

Eulers Methode ist eine numerische Methode, welche einen Differentialgleichungssystem durch diskrete Schritte approximiert. Die Methode ist einfach zu implementieren, kann jedoch aufgrund ihrer funktionsweise ungenau sein. Als erstes bestimmen wir unseren diskreten Zeitschritt Δt. Dannach bestimmen die Startwerte für die Position und Geschwindigkeit der Körper im Raum.

Nun können wir für alle n Körper die Gravitationskräfte berechnen, welche auf die Körper wirken. In diesem Fall ist die Kraft $\overrightarrow{F_G}$ ein Vektor, welcher die Richtung und Stärke der Kraft angibt. Mit dieser Kraft können wir die Beschleunigung der Körper bestimmen. Mit $\vec{a} = \frac{\overrightarrow{F_G}}{m}$ und $\vec{v} = \vec{v} + \vec{a} * \Delta t$ können wir nun die angepassten Geschwindigkeiten der Körper bestimmen. Die Geschwindigkeit ist in diesem Fall ein 3-Dimensionaler Vektor, welcher die Geschwindigkeit in x, y und z Richtung angibt. Für einen Körper kann nun die neue Position $\vec{r}$ im Raum mit $\vec{r} = \vec{r} + \vec{v} * \Delta t$ bestimmt werden.

Wenn wir dieses Verfahren nun für alle Körper im System durchführen, können wir die Bewegung der Körper simulieren. Dieses Verfahren von hand durchzuführen ist sehr aufwendig, weshalb sich die implementation als Computerprogramm anbietet.

IV. Vorbereitung

A. Datensätze

Für die Simulation des Sonnensytems benötigen wir Daten über die Planeten und ihre diversen Satellieten. Die wichtigen Daten für die Simulation sind die Masse der Körper, die Position der Körper im Raum und die Geschwindigkeit der Körper. Diese Daten können aus diversen Quellen bezogen werden. Für die Planeten wurden die Daten aus dem Projekt *Nasa Data Scraper* von *Devstronomy* [4] auf github verwendet. Das gleiche Projekt beinhält auch einen Datensatz für die Satellieten der Planeten. Dieser ist allerdings für unsere Simulation nicht geeignet, da keine Informationen zu Position und Geschwindigkeit der Satellieten enthalten sind.

Die Daten der Satellieten wurden vom *Solar System Dynamics* Projekt des *Jet Propulsion Laboratory* der *NASA* [5] verwendet. Vom *SSD* Projekt können wir sowohl die Phsyikalischen Daten der Satellieten [6], als auch die orbitale Parameter der Satellieten [7] beziehen. Diese Daten beinhalten die *Semi Major Axis* und weitere Parameter, welche für die Simulation benötigt werden.

B. Wahl der Programmiersprache

Da für diese Simulation eine große Anzahl an Berechnungen durchgeführt werden muss, stellte die Wahl der Programmiersprache eine wichtige Entscheidung dar. Für diese Simulation wurde deshalb C++ gewählt. C++ ist eine schnelle kompilierte Sprache.

C. Wahl von Hilfsbibliotheken

Zur Visualisierung unserer Simulation benötigen wir eine Bibliothek, welche das darstellen von 3D-Objekten ermöglicht. Hierzu wurde die Bibliothek *Raylib* [8] verwendet. Raylib ist eine C-Bibliothek, welche das erstellen von 2D und 3D Anwendungen ermöglicht. Neben der Darstellung von 3D-Objekten war es auch noch wichtig, dass wir einfache Interface-Elemente anzeigen können. Hierzu wird die Bibliothek *Dear ImGui* [9] verwendet. ImGui ist eine C++ Bibliothek, welche in der Spieleentwicklung verwendet wird, um einfache Interface-Elemente anzuzeigen. Um diese beiden Bibliotheken zusammen zu verwedne, wurde das Projekt *rlImGui* [10] verwendet.

Die Bibliotheken *ImGui* und *rlImGui* wurden als Submodule in das Projekt eingebunden. *Raylib* wird während der kompilierung des Projektes heruntergeladen und verwendet. Hierzu wurde das Programm *Cmake* [11] verwendet. Cmake ist ein Build-System, welches es ermöglicht, C und C++ Projekte plattformunabhängig zu kompilieren.

V. Implementierung

A. Grundlegendes

Die Implementierung der Simulation folgt dem zuvor beschriebenen Ablauf. In der *Main*-Funktion der Simula-

tion werden zuerst die nötigen Bibliotheken initialisiert. Für dieses Projekt war es nötig, einen eigenen *Vektor*-Datentypen zu verwenden. Die Gründe Hierzu werden in Section VI.A erläutert. Dieser *SciVec* verwendet 64-Bit gleitkomma-Zahlen, um möglichst preziese Berechnungen zu ermöglichen. Der Datentyp und die dazugehörigen Funktionen zur Addition, Subtraktion, Multiplikation und Skalierung können in *SciVec.h* geufunden werden. Die Körper des Sonnensystems werden mithilfe eines Datentypens *Body* abgebildet. In *Body.h* wird das *struct Body* wie folgt definiert:

```
struct Body
{
  std::string name;
  double radius;
  double mass;
  Color color{};
  SciVec3 position;
  SciVec3 velocity;
  bool isPlanet;

  int satelliteCount = 0;
  std::string satelliteNames[64] = {""};

  ...
}
```

Listing 1: Definition des _Body_-Datentyps

Die Felder *radius*, *mass*, *postion*, und *velocity* sind für die Simulation am wichtigesten. Für jedes Objekt wird außerdem ein *Name*, eine *Farbe*, und eine Liste an Satllieten gespeichert. Die Simulation verwaltet eine danamische Liste (*std::vector*) an *Body*-Objekten. Diese Liste stellt den aktuellen zustand der Simulation da. Um diese Liste nun mit Daten zu füllen, müssen wir unsere Datensätze einladen.

B. Laden der Datensätze

Die Datei *Loader.h* beinhaltet die Funktionen, welche die Datensätze aus Section IV.A lädt. Die Funktion *loadPlanets* lädt die *planets.csv*-Datei aus dem *Assets*-Ordner. Zuerst erstellt diese Funktion allerdings die Sonne, welche nicht im Datensatz enthalten ist. Hierfür wurden die wichtigsten Werte in *Consts.h* definiert. Dannach iteriert die Funktion über die Zeilen der CSV-Datei und erstellt für jede Zeile ein *Body*-Objekt. Hierbei ist es wichtig, die einzelnen Werte in die Korrekten Einheiten umzuwandeln. Für diese Simulation wurde die Entscheidung getroffen, alle Werte in SI-Einheiten zu speichern. Insbesonders für die Datenwerte *Mass* war es nötig, 64-Bit Gleitkommazahlen zu verwenden. Um die Visualisierung besser zu gestalten, wurden allen Planeten eine individuelle Farbe zugeordnet.

Neben der bereits genannten Daten, war die bestimmung der initialen Position und Geschwindigkeit der Körper eine wichtige Entscheidung. Der Datensatz der Planeten beinhält sowohl Werte für den *Aphelion*, den *Perihelion*, die *Inclination* und die *Exzentrizität* der Planeten. Des Weiteren wurde ein Wert für die durchschnittliche *orbital_velocity* angegeben.

Die Funktion *loadSatallites* funktioniert ähnlich wie *loadPlanets*. Hierbei wird die Dateien *satellites_data.csv* und *satellites_orbit.csv* kombiniert, um die Satellieten der Planeten zu erstellen. Für jeden Sateliet wird zuerst der *Parent*-Körper gesucht. Dannach werden die weiteren Planetaren informationen bestimmt. Dieser Datensatz weißt die Besonderheit auf, dass nicht die Masse der Objekte angegeben ist, sondern der *Standard Gravitational Parameter* μ. Dieser Wert ist definiert als $\mu = G * m$, wobei G die Gravitationskonstante ist. Um die Masse der Satellieten zu bestimmen, wird der Wert μ durch G geteilt.

Die bestimmung der initialen Position und Geschwindigkeit der Satellieten ist etwas komplizierter. Hierzu müssen wir nicht nur die Orbitalen Parameter des Satllieten, sondern auch die des Parent-Körpers verwenden.

C. Aufteilung von Simulation und Visualisierung

Die Simulation und Visualisierung sind zwei verschiedene Aufgaben mit unterschiedlichen Zielen. Die Visualisierung in geregelten Abstände den Zustand der Simulation darstellen. Dieser Vorgang ist größtenteils abhängig von der Grafikkarte. Die Simulation hingegen ist abhängig von der CPU. Ideallerweise sind die beiden Prozesse soweit wie möglich voneinander getrennt. Dies wurde in diesem Projekt so realisiert, dass die Simulation in einem getrennten *Thread* läuft. Die Visualisierung hingegen läuft im *Main*-Thread der Anwendung. Beide Threads greifen auf die selbe Liste an Daten zu, weshalb es nötig war, die Zugriffe auf diese Liste zu synchronisieren. Um eine möglichst geringe Latenz für den Nutzer zu ermöglichen, hat die Visualisierung die priämre Kontrolle über die Liste. Sie kann angeben wann die Simulation auf die Liste zugreifen kann, indem die Simulation temporär pausiert wird.

D. Implementieren der Simulation

Die Simualtion ist der Kern der Anwendung. Um möglichst große Zeiträume simluieren zu können, ist es also nötig, diesen Vorgang möglichst effizient zu implementieren. Die Simulation wird in der Funktion *simulate* in *main.cpp* implementiert. Diese Funktion wird in einem eigenen *Thread* ausgeführt. Dieser Thread arbeitet komplett unabhängig von der Visualisierung. Solang die Simulation nicht pausiert ist, entwedet durch den Nutzer, oder durch die Visualisierung, wird die Simulation in einer Schleife ausgeführt. Im folgenden wird eine vereinfachte Version der Funktion *simulate* gezeigt:

```
void simulate()
{
  if(simulationRunning)
  {
    for(body1 in bodies)
    {
      for(body2 in bodies)
      {
        if(body1 == body2)
          continue;

        Force = calculate_gravity(body1, body2);

        body1.acceleration += Force / body1.mass;
        body2.acceleration -= Force / body2.mass;

      body1.velocity += body1.acceleration * delta_t;
      body2.velocity += body2.acceleration * delta_t;

        body1.position += body1.velocity * delta_t;
        body2.position += body2.velocity * delta_t;
      }
    }
  }
}
```

Listing 2: Vereinfachte Version der Funktion _simulate_

Die Funktion folgt dem zuvor definierten Aufbau. Für jeden Körper wird die Gravitationskraft berechnet, welche auf den Körper wirkt. Mit dieser Kraft wird die Beschleunigung des Körpers bestimmt. Mit der Beschleunigung wird die Geschwindigkeit, und damit die Position des Körpers bestimmt. Dieser Vorgang wird für alle Körper in der Liste *bodies* durchgeführt.

Wichtig ist hierbei, dass die Berechnung von der globalen variable *delta_t* abhängig ist. Diese Variable bestimmt den Zeitschritt der Simulation. Je kleiner dieser Wert ist, desto genauer ist die Simulation, jedoch auch langsamer. Je nach verfügbarer Rechenleistung kann dieser Wert auf bis zu 1s pro iteration gesetzt werden. Mehr hierzu in Section VII.

E. *Implementieren der Visualisierung*

Ideen:

- Visualisierung der Planeten
- Visualisierung der Satellieten
- Trails
- Spherecial camera movement
- Interface Elemente

VI. Probleme und Lösungen

A. *32-Bit gleitkommazahlen*

B. *Floating Origin*

VII. Ergebnisse

VIII. Diskussion

Quellen

[1] Nikolaus Kopernikus, *De revolutionibus orbium coelestium*. 1543.

[2] *Astronomia nova*. 1609.

[3] *Philosophiæ Naturalis Principia Mathematica*. 1644.

[4] "NASA Data Scraper." Jul. 08, 2024. [Online]. Available: https://github.com/devstronomy/nasa-data-scraper/tree/master

[5] "Solar System Dynamics." [Online]. Available: https://ssd.jpl.nasa.gov/

[6] "Satellite physical data." [Online]. Available: https://ssd.jpl.nasa.gov/sats/phys_par/

[7] "Satellite orbital data." [Online]. Available: https://ssd.jpl.nasa.gov/sats/elem/

[8] raysan5, "Raylib." Jul. 08, 2024. [Online]. Available: https://github.com/raysan5/raylib.git

[9] ocornut, "Imgui." Jul. 08, 2024. [Online]. Available: https://github.com/ocornut/imgui.git

[10] "Rlimgui." Jul. 08, 2024. [Online]. Available: https://github.com/raylib-extras/rlImGui.git

[11] "CMake." [Online]. Available: https://cmake.org/