Gymnasium Bruckmühl *Abiturjahrgang 2023*

S E M I N A R A R B E I T

Rahmenthema des Wissenschaftspropädeutischen Seminars:
Die Erforschung des Weltalls

Leitfach: Physik (Astrophysik)

Thema der Arbeit:

Simulation des Sonnensystems

Verfasser: Florian Maier Kursleiter: Herr Koch

Abgabetermin: *8. November 2022*
(2. Unterrichtstag im November)

Bewertung	Note	Notenstufe in Worten	Punkte		Punkte
schriftliche Arbeit				x 3	
Abschlusspräsentation				x 1	
				Summe:	
				Gesamtleistung nach § 61 (7) GSO = Summe **: 2** (gerundet)	

Datum und Unterschrift der Kursleiterin bzw. des Kursleiters

Inhaltsverzeichnis

1. Einleitung

1.1. Ziel der Arbeit und Forschungsfrage

Meine Arbeit besteht aus der Erstellung eines Computerprogramms zur Simulation des Sonnensystems und der Analyse der Simulationsergebnisse. Das Programm soll in der Lage sein die Bewegung von Massen unter dem Einfluss der Schwerkraft zu simulieren, um damit das Sonnensystem abzubilden.
Die Simulation soll ein fortlaufendes, nach dem realen Vorbild funktionierendes System darstellen, aus dem man Messwerte entnehmen kann. Diese Messwerte sollen mit den entsprechenden Messwerten aus der Realität verglichen werden, um damit Aussagen über die Genauigkeit der Simulation zu machen.
Dafür nimmt das Programm reale Messwerte, des Sonnensystems zu einem Zeitpunkt t_0 entgegen und simuliert dann mithilfe der physikalischen Gesetze die Bewegung dieser Körper.

1.2. Datenquelle Stellarium

Stellarium ist eine Open Source Software, die seit dem Jahr 2000 von freiwilligen Programmierern und anderen Experten aus dem Bereich rund um Astrophysik entwickelt wird. Das Programm bietet für Laien einen einfachen und kostenlosen Weg, astronomische Beobachtungen vorauszuplanen, oder schlicht die beindruckend dargestellten Objekte zu bestaunen, die in der 3d-Ansicht zu sehen sind. [vgl. L1, S.225-226]
Außerdem werden dem Nutzer die Positionsdaten von allen angezeigten Objekten zugänglich gemacht und die Zeit kann auf die Sekunde genau eingestellt werden. Dadurch hat das Programm großen Wert für diese Arbeit, da verlässliche Anfangswert aus der Realität gebraucht werden. Diese Anfangswerte stammen alle vom Zeitpunkt t_0 und werden aus Stellarium bezogen.

2. Das N-Körper Problem der Gravitation

Allgemein sind beim N-Körper Problem N punktförmige Massen im dreidimensionalen Raum (im $\mathbb{R}^3$) gegeben ($i = 1, \dots, N$). Zwischen je zwei Objekten

(mit Index i und j) wirkt dabei eine Kraft der Form $\vec{F}_{i,j} = C\frac{m_i \cdot m_j}{|\overrightarrow{x_j}-\overrightarrow{x_i}|^2} \cdot \frac{(\overrightarrow{x_j}-\overrightarrow{x_i})}{|\overrightarrow{x_j}-\overrightarrow{x_i}|}$, wobei mit $C\frac{m_i \cdot m_j}{|\overrightarrow{x_j}-\overrightarrow{x_i}|^2}$ der Betrag der Kraft und mit $\frac{(\overrightarrow{x_j}-\overrightarrow{x_i})}{|\overrightarrow{x_j}-\overrightarrow{x_i}|}$ die Richtung berechnet wird.

In der Natur kommen Kräfte dieser Form bei dem Coulombschen Gesetz und dem Gravitationsgesetz vor. Die Formel für punktförmige Ladungen hat dann Form $\vec{F}_{i,j} = -\frac{1}{4}\pi\varepsilon_0 \frac{q_i \cdot q_j}{|\overrightarrow{x_j}-\overrightarrow{x_i}|^3}(\overrightarrow{x_j}-\overrightarrow{x_i})$, wobei $C = -\frac{1}{4}\pi\varepsilon_0$ und die Masse m_i durch die Ladung q_i ersetzt wird. [vgl. I5, S.2]

Im Falle des Gravitationsgesetzes gilt $C = G$, also ist die Formel für die Kraft:

$$\vec{F}_{i,j} = G\frac{m_i \cdot m_j}{\left|\overrightarrow{x_j}-\overrightarrow{x_i}\right|^3}(\overrightarrow{x_j}-\overrightarrow{x_i})$$

Damit gibt es eine Formel für die Kraft zwischen zwei Objekten, will man aber die Gesamtkraft $\vec{F}_i$ berechnen, muss man $\vec{F}_{i,j}$ für alle $0 < j \leq N, j \neq i$ ausrechnen und die gerichteten Kräfte nach dem Prinzip der Kräfteaddition addieren. Das Ergebnis ist die Schwerkraft, die auf den i-ten Körper durch die anderen $N-1$ Körper ausgeübt wird.

$$\vec{F}_i = G\sum_{j=1;j\neq i}^{N}\frac{m_i \cdot m_j}{\left|\overrightarrow{x_j}-\overrightarrow{x_i}\right|^3}(\overrightarrow{x_j}-\overrightarrow{x_i})$$

Mit $\vec{F}_i$ kann auch die Beschleunigung berechnet werden, weil nach dem Newtonschen Gesetz gilt, $\vec{F} = m \cdot \vec{a}$ und damit gilt dann für die Beschleunigung $\vec{a} = \frac{\vec{F}}{m}$. [vgl. L2, S.5]

Außerdem ist bekannt, dass die Beschleunigung nichts anderes als die erste Ableitung, der Geschwindigkeit und die zweite Ableitung der Position x nach der Zeit ist, also gilt $\vec{a} = \ddot{\vec{x}} = \dot{\vec{v}}$. [vgl. L2, S.11]

Wendet man diese Überlegungen an, erhält man die Newtonschen Bewegungsgleichungen der Gravitationskraft für N Körper [vgl. I4, S.6]

$$\ddot{\vec{x}}_i = \dot{\vec{v}}_i = G\sum_{j=1;j\neq i}^{N}\frac{m_i m_j(\overrightarrow{x_j}-\overrightarrow{x_i})}{m_i\left|\overrightarrow{x_j}-\overrightarrow{x_i}\right|^3} = G\sum_{j=1;j\neq i}^{N}\frac{m_j(\overrightarrow{x_j}-\overrightarrow{x_i})}{\left|\overrightarrow{x_j}-\overrightarrow{x_i}\right|^3}$$

Für diese Newtonschen Bewegungsgleichungen gibt es für $N > 2$ keine geschlossene analytische Lösung. [vgl. I4, S.8-9]

2.1. Numerischer Lösungsansatz

Da also keine analytische Lösung für die Funktion $\vec{x}_i(t)$ gefunden werden kann, weicht man auf einen numerischen Lösungsansatz aus. Hierfür berechnet man $\vec{x}_i(t)$ in kleinen Zeitschritten Δt, indem man die Ableitung annähert (Näherung der Tangentensteigung durch die Sekantensteigung). [vgl. L3, S.28-29]
Die Newtonschen Gleichungen werden hierfür als Differentialgleichungen für $\vec{x}_i(t)$ und $\vec{v}_i(t)$ formuliert.

$$\dot{\vec{x}}_i = \overrightarrow{v_i}$$

$$\dot{\vec{v}}_i = G \sum_{j=1;j\neq i}^{N} \frac{m_j\left(\overrightarrow{x_j} - \overrightarrow{x_i}\right)}{\left|\overrightarrow{x_j} - \overrightarrow{x_i}\right|^3}$$

Indem man die Ableitung durch die Sekantensteigung nähert, erhält man statt $\dot{\vec{x}}_i$ folgende Näherung

$$\dot{\vec{x}}_i(t) \approx \frac{\overrightarrow{x_i}(t+\Delta t) - \overrightarrow{x_i}(t)}{\Delta t}$$

$$\dot{\vec{v}}_i(t) \approx \frac{\overrightarrow{v_i}(t+\Delta t) - \overrightarrow{v_i}(t)}{\Delta t}$$

Somit ergibt sich genähert

Formel 1

$$\overrightarrow{x_i}(t+\Delta t) = \overrightarrow{x_i}(t) + \Delta t \cdot \overrightarrow{v_i}(t)$$

Formel 2

$$\overrightarrow{v_i}(t+\Delta t) = \overrightarrow{v_i}(t) + \Delta t \cdot G \sum_{j=1;j\neq i}^{N} \frac{m_j\left(\overrightarrow{x_j}(t) - \overrightarrow{x_i(t)}\right)}{\left|\overrightarrow{x_j}(t) - \overrightarrow{x_i(t)}\right|^3}$$

Dabei gilt

Formel 3

$$G \sum_{j=1;j\neq i}^{N} \frac{m_j\left(\overrightarrow{x_j}(t) - \overrightarrow{x_i(t)}\right)}{\left|\overrightarrow{x_j}(t) - \overrightarrow{x_i(t)}\right|^3} = \vec{a}_i(t)$$

Mit der obigen Formel kann man das N-Körper Problem approximativ lösen. Dabei muss klar sein, dass die Ergebnisse immer eine Abweichung von der eigentlichen Lösung haben werden, weil die Sekantensteigung nur näherungsweise der Tangentensteigung entspricht. Man kann die gewünschte Genauigkeit Vergrößern, indem man Δt kleiner macht. Die anfallenden Rechenschritte pro simulierter Zeit

sind dabei Proportional zu Δt, also $\frac{N}{T} \sim \Delta t$, wobei N die Anzahl der Rechenschritte ist und T die simulierte Zeitspanne.

3. Umsetzung des Programms

Um den Aufbau der Simulation zu erklären, müssen zuerst die Grundbedingungen geklärt werden, die Objekte haben. Das Konstrukt besteht aus einem kartesischen Koordinatensystem im $\mathbb{R}^3$ und Physik-Objekten, die in sich in diesem System befinden. Der Ursprung des Koordinatensystems kann theoretisch an jeder beliebigen Stelle liegen, ist aber, zur Vereinfachung als die Startposition der Sonne definiert, also ist $\vec{x}_{0,Sonne} = \vec{0}$

Jedes Physik-Objekt hat einen Positionsvektor $\vec{x}$ und einen Geschwindigkeitsvektor $\vec{v}$. Zusätzlich zu den beiden Vektoren hat auch jedes Physik-Objekt eine Masse m und unterliegt damit dem Gravitationsgesetz.

Der in

2. Das N-Körper Problem der Gravitation beschriebene Wert Δt wird auf den Wert 30 Sekunden gesetzt. Mit diesem Wert werden alle späteren Ergebnisse berechnet. Mehr zur Auswirkung dieses Wertes, wird in **4.4. Abhängigkeit von Δt** erklärt.

3.1. Grundsätzliches zum Programm

Im Programmcode werden Physik-Objekte mit der Klasse *PhysikObjekt* repräsentiert. Für jedes Objekt, dass man hinzufügen will, erschafft man ein neues Objekt von *PhysikObjekt*. Alle so erschaffenen Objekte werden in einer Liste *physikObjekte* gespeichert. Diese Liste befindet sich in der Klasse *PhysicThread*. Die Ressource **Quellcode 1** ist ein Ausschnitt des Programmcodes von *PhysicThread* und zeigt die wichtigsten Funktionen der Klasse.

3.2. Numerischer Lösungsansatz

In ***2.1.*** wurde erklärt, wie die Lösung des N-Körper Problems genähert werden kann. Die Umsetzung in Java kann man in der Ressource **Quellcode 2** sehen.

Dieser Code liegt in der Methode *update*, der Klasse *PhysikObjekt*, deren Aufruf schon vorher gezeigt wurde. Die Methode befindet sich im Objekt mit Index i. Das heißt die lokalen Variablen, **pos** und **mass**, die bei der Berechnung verwendet werden stehen für $\vec{x}_i$ und m_i. Dabei steht die Abkürzung *pos* für den Ortsvektor und *mass* für die Masse des Objektes.

Es ist bekannt, dass für die Gesamtkraft $\vec{F}_i$ alle Kräfte $\vec{F}_{i,j}$ für $0 < j \leq N, j \neq i$ addiert werden müssen und genau das macht die for-Schleife, die in der zweiten Zeile des Code-Ausschnitts steht.

Innerhalb der Schleife wird dann die Berechnung von $\vec{F}_{i,j}$ durchgeführt. Dafür wird zuerst der Betrag der Kraft *force* berechnet und dann der Verbindungsvektor *diffVektor* von i und j, also $\overrightarrow{x_i x_j}$. Dieser Vektor wird normalisiert und dann mit *force* multipliziert, dann kommt man auf den Vektor *fij*, der $\vec{F}_{i,j}$ darstellt. Als letzter Schritt in der Schleife wird jetzt *fij* zur Summe der Gesamtkräfte *f* addiert.

Nachdem durch alle Einträge von physikObjekte iteriert wurde, ist der Vektor *f* der Vektor der Gesamtkraft $\vec{F}_i$. Daraus kann in der letzten Zeile des Code-Abschnitts die Beschleunigung berechnet werden, in dem die Kraft durch die Masse dividiert wird. Der gerade erklärte Code setzt *Formel 3* in Programmform um. Mit der Beschleunigung können jetzt *Formel 1 und Formel* angewendet werden.

```
newPos = pos.add(speed.mulitply(PhysicThread.deltaT));

speed = speed.add(a.mulitply(PhysicThread.deltaT));
```

3.3. Funktionalitäten für den Umgang mit Realdaten

Bisher wurde erklärt, wie die Simulation des N-Körperproblems mit Software funktioniert. In diesem Unterkapitel geht es darum, dieser Software Realdaten zur Verfügung zu stellen, so dass am Ende eine Simulation des Sonnensystems steht. Diese Realdaten stammen alle von demselben, in der Vergangenheit liegenden Zeitpunkt t_0.

Wie in ***3.*** **Umsetzung des Programms** erklärt ist, wird ein dreidimensionales Koordinatensystem verwendet, wobei $x_{0,Sonne} = \vec{0}$.

Um ein Physik-Objekt zu initialisieren wird Code der folgenden Form verwendet:

```
PhysikObjekt neuerPlanet = new PhysikObjekt(x1, x2, x3);
neuerPlanet.initPhysics(v1, v2, v3, m);
```

Wichtig sind hier die beiden Vektoren $\vec{x}$ und $\vec{v}$, mit jeweils 3 Komponenten, welche Position und Geschwindigkeit für den Zeitpunkt t_0 festlegen. Außerdem wird die Masse m übergeben.

Es wurde bereits definiert, dass die Sonnenposition auf dem Nullpunkt liegt, außerdem ist sie der Bezugspunkt des Planetensystems also gilt für ihre Startgeschwindigkeit $\vec{x}_{0,Sonne} = \vec{v}_{0,Sonne} = \vec{0}$. Jetzt muss nur noch die Masse der Sonne eingesetzt werden und das Erste Objekt des Systems ist definiert.

Die Erde ist der zweite Körper des Systems und es werden für ihre Initialisierung 3 Werte gebraucht: Der Abstand von Erde und Sonne d_E, der Vektor der Erdbahngeschwindigkeit und die Erdmasse. Es wird kein vollständiger Ortsvektor benötigt, weil für die Ebene, in der die Erdbahn liegt, noch keine Richtung durch andere Objekte ausgezeichnet wurde.

Es ist also egal welchen Wert der Verbindungsvektor von Erde und Sonne $\overrightarrow{x_{0,S}x_{0,E}}$ hat, solange $|\overrightarrow{x_S x_E}| = d_E$. Damit kann die Ausrichtung der Ebene frei und damit möglichst einfach gewählt werden. Deswegen wird die Erdbahn in die x1-x2-Ebene gelegt. $|\overrightarrow{x_S x_E}|$ hat also die x3-Koordinate 0. Um die Anfangssituation weiter zu vereinfachen, wird der Vektor entlang der x1-Achse ausgerichtet, damit ist er also definiert als $\overrightarrow{x_{0,S}x_{0,E}} = \begin{pmatrix} d_{0,E} \\ 0 \\ 0 \end{pmatrix}$

„Ist ein Vektor durch die Punkte A und B gegeben, so gilt: $\overrightarrow{AB} = \vec{B} - \vec{A}$“ [L3, S.97].

Also gilt, $\begin{pmatrix} d_E \\ 0 \\ 0 \end{pmatrix} = \overrightarrow{x_E} - \overrightarrow{x_S} = \overrightarrow{x_E}$ weil ja $\overrightarrow{x_S} = \vec{0}$. Daraus folgt $x_{0,E} = \begin{pmatrix} d_E \\ 0 \\ 0 \end{pmatrix}$

3.3.1. Berechnung von Geschwindigkeitsvektoren

Der Betrag der Bahngeschwindigkeit der Erde $|\overrightarrow{v_0}|$ ist gegeben und es ist bekannt, dass $v3_{0,E} = 0$, weil die Erdbahn in der x1-x2-Ebene liegt. Um mit diesen Informationen $\overrightarrow{v_0}$ zu bestimmen fehlt noch der Winkel α (vgl. *Abbildung 1)*. Dafür wird ein zweiter Zeitpunkt nach t_0 definiert: t_1

Aus der Position der Sonne, der Erdposition zum Zeitpunkt t_0, $\vec{x}_E(t_0)$ und der Erdposition zum Zeitpunkt t_1, $\vec{x}_E(t_1)$ entsteht das Dreieck aus *Abbildung 1*.

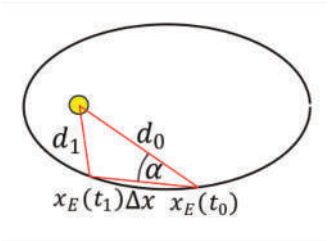

Abbildung 1 (der Abstand zwischen den Punkten ist zum bessren Verständnis übertrieben groß)

Die Seitenlängen d_0 und d_1 können aus den Stellarium Daten ermittelt und die Seitenlänge Δx näherungsweise mit der Formel $\Delta x = |\overrightarrow{v_0}| \cdot (t_1 - t_0)$ berechnet werden. [vgl. L2, S.11]

Für die Berechnung von α kann der Kosinussatz verwendet werden:

$a^2 = b^2 + c^2 - 2bc \cdot \cos(\alpha)$ [vgl . L4, S.44]

Umgestellt nach α und angewendet auf *Abbildung 1* lautet die Formel:

$$\alpha = arccos\left(\frac{{d_1}^2 - {d_0}^2 - (|\overrightarrow{v_0}| \cdot \Delta t)^2}{-2 \cdot d_0 \cdot |\overrightarrow{v_0}| \cdot \Delta t}\right)$$

Um die Richtung der Geschwindigkeit zu erhalten, muss der Richtungsvektor $\overrightarrow{x_E(t_0)x_S(t_0)}$ zwischen Erde und Sonne nun um den Winkel α gedreht werden. Die Rotation findet um die x3-Achse statt. Dafür wird eine 3-dimensionale Rotations-Matrix benutzt. Da $\overrightarrow{x_E(t_0)x_S(t_0)}$ so gewählt wurde, dass es auf der x1-Achse liegt, ist der normalisierte Vektor gleich $\vec{n} = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}$. Dieser Vektor kann dann mit der Rotationsmatrix um den Winkel α gedreht werden. [vgl. I3, S.10]

Der Zeitunterschied zwischen t_0 und t_1 wird für die Simulation so groß gewählt, dass die Abstände d_0 und d_1 in Stellarium unterschiedlich groß sind (typischerweise 1-24 Stunden).

Dieser Prozess zur Richtungsbestimmung des Geschwindigkeitsvektors lässt sich auch auf alle anderen Planeten anwenden. Hierbei ist jeweils der Betrag der Bahngeschwindigkeit gegeben und es muss der Geschwindigkeitsvektor bestimmt werden.

Es gibt aber einen Unterschied, denn gerade wurde vorausgesetzt, dass die Ekliptik-Ebene der Erde in der x1-x2-Ebene liegt. Das ist für andere Planeten nicht der Fall, denn ihre Bahnebene ist im Vergleich zur Ekliptik-Ebene der Erde gedreht. Der Winkel wird als Inklination, kurz i bezeichnet.

Um für einen solchen Planeten $\vec{v}_0$ zu bestimmen, befolgt man jetzt die gleichen Schritte, wie oben erklärt, am Ende verwendet man aber eine zusätzliche Rotationsmatrix. Man rotiert die Ebene, welche die Planetenbahn mit der Sonne bildet, um die x1-Achse. Deshalb wird der Vektor $\vec{v}_0$, entlang der x1-Achse um den Winkel i gedreht.
Abbildung 3 zeigt diese Schiefe und die entsprechenden Rotationswinkel. **Quellcode 3** zeigt die Umsetzung der oberen Überlegungen im Programm für einen beliebigen Planeten.

3.3.2. Umrechnung der Koordinaten

Um für die Planeten nicht nur den Geschwindigkeitsvektor, sondern auch die Position zu bestimmen, werden Realdaten aus Stellarium verwendet. Hier tritt aber ein Problem auf, denn die realen Planetenpositionen werden nicht im kartesischen Koordinatensystem der Simulation angegeben. Stattdessen werden sie im äquatorialen Koordinatensystem angegeben.
Das äquatoriale Koordinatensystem gibt die Position eines Himmelskörpers am Nachthimmel an und orientiert sich dabei an der Äquatorebene.
Der Bezugspunkt dieses Systems wird auf den sogenannten Frühlingspunkt gesetzt. Der Begriff Frühlingspunkt hat eine räumliche und zeitliche Bedeutung. Zeitlich ist der Frühlingspunkt ein Zeitpunkt, an dem die Sonne genau auf dem Äquator im Zenit steht. Also schneidet sich dabei die Ekliptik der Erde mit der der Erdumlaufbahn. Diese Konstellation entsteht ca. am 21. März jedes Jahr.
Am zeitlichen Frühlingspunkt befindet sich der Frühlingspunkt in Richtung des Vektors $\overrightarrow{x_E x_S}$ der von der Erde zur Sonne zeigt. [vgl. I1, Kapitel 2.1]
Der Grüne Pfeil in *Abbildung 2* liegt genau auf dem Vektor $\overrightarrow{x_E x_S}$, wenn T auf einen Frühlingspunkt fällt. Will man jetzt einen Punkt J (*Abbildung* 2) auf dem Himmel angeben, gibt man den Winkel α und den Winkel δ an. α wird Rektaszension genannt und beschreibt den Winkel den J und der Stundenkreis auf dem Äquatorebene einschließen. Die Rektaszension wird in Stunden von 0h bis 24h angegeben. Die Deklination δ wird in Grad angegeben und ist der Winkel auf dem Stundenkreis, den der Äquatorebene und die Sternbahn einschließen. Dabei ist mit Stundenkreis, der geografische Längengrad und mit Sternbahn der Breitengrad gemeint.

Man kann also mit dem äquatorialen System einen Punkt auf dem Himmel beschreiben. Mathematisch kann man das mit einem Vektor $\vec{n}_J$ darstellen, der vom Erdmittelpunkt in Richtung J zeigt. Der Vektor ist normalisiert, weil die äquatorialen Koordinaten keine Information über die Entfernung des Himmelskörpers geben.

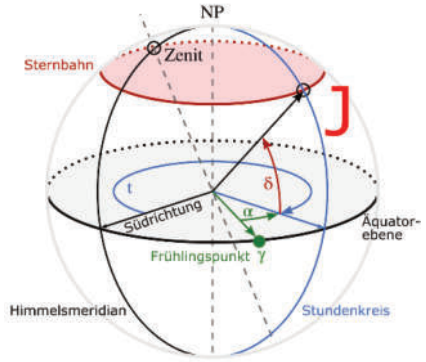

Abbildung 2 [Bildquelle: I6]

Jetzt muss der Vektor $\vec{n}_J$ mit den äquatorialen Koordinaten aus Stellarium in das Koordinatensystem der Simulation transformiert werden. Dafür muss der Vektor zum Frühlingspunkt, als Bezugspunkt im kartesischen System der Simulation gegeben sein. Indem t_0 als Frühlingspunkt des Jahres 2000 (20. März, 7 Uhr 34) gewählt wird, fällt die x1-Achse meines Systems mit dem Vektor zum Frühlingspunkt zusammen. Der normalisierte Vektor zum Frühlingspunkt $\overrightarrow{n_F}$ ist $\overrightarrow{n_F} = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}$, weil $\vec{x}_{0,Erde} = \begin{pmatrix} d_S \\ 0 \\ 0 \end{pmatrix}$.

In *Abbildung 5* kann man sehen, dass zum Frühlingspunkt 2000 die Sonne auf den äquatorialen Koordinaten 0, 0 liegt.

Für den Vektor eines Planeten werden nun die Koordinaten im kartesischen System der Simulation benötigt. Ausgangspunkt sind die Winkel α und δ im äquatorialen System. Dazu wird der Vektor um die Winkel α und δ gedreht. Die Rotation wird so gewählt, dass nach der Rotation der Vektor $\vec{n}_F$, der ja den Bezugspunkt des äquatorialen Systems darstellt, in die Richtung des Planeten zeigt.

Für die Rotation werden dreidimensionale Rotations-Matrizen verwendet. Jede Raumrichtung hat dabei ihre eigene Rotations-Matrix. Um einen Vektor beispielsweise um die x1-Achse um den Winkel α zu rotieren, multipliziert man ihn mit der Matrix $R_{x1}(\alpha)$. [vgl. I3, S.10]

Man muss im Kopf behalten, dass alle Winkelangaben von der Erde aus gelten und die Erde wie vorher erklärt zum Zeitpunkt t_0 auf der x_1-Achse liegt. Die Bewegung der Erde findet in der x1-x2-Ebene statt, diese Ebene kann also als Ekliptik-Ebene der Erde gesehen werden.
Die Äquatorebene der Erde schließt mit der Ekliptik-Ebene einen Winkel von ca. 23,5 Grad ein. Die Winkelangaben α und δ beziehen sich aber auf die Äquatorebene.
Um diesen Winkel auszugleichen, wird die Planetenebene, also die Ebene, in der seine Umlaufbahn um die Sonne liegt, um ca. 23,5 entlang der x1-Achse rotiert.
Durch die drei Rotationen geht der Vektor $\vec{n}_F$ in den Vektor $\vec{n}_J$ über.
Jetzt, wo $\vec{n}_J$ bekannt ist muss man damit noch den Ortsvektor $\vec{x}_J$ bestimmen. $\vec{n}_J$ ist bekanntlich normalisiert und hat deswegen die Länge 1. Wird jetzt ein Zahlenwert mit dem Vektor multipliziert, behält der Vektor seine Richtung und erhält die Länge des Zahlenwertes. Deswegen ist die Formel für den Ortsvektor:

$$\vec{x}_J = \vec{n}_J \cdot d$$

Quellcode 4 zeigt die Umsetzung der oberen Überlegungen im Programm. *ComputedDeclination* und *computedRektaszension* bedeutet dabei, dass die Rektaszension und die Deklination in einen Zahlenwert von 0 bis 1 formatiert wurden.

3.4. Visualisierung

Das Programm zeigt die simulierten Objekte in einer einfachen 3d-Darstellung und bietet außerdem frei steuerbare Bewegung und Blickwinkel.
Um dieses Ergebnis zu erzielen, wurde die Renderpipeline Opengl eingesetzt. Diese Software kann frei heruntergeladen und in Java-Projekte als Bibliothek importiert werden. Mit der Bibliothek können dann geometrische Formen, wie in diesem Fall Kugeln auf den Bildschirm gezeichnet werden. Diese Kugeln werden aber immer an dieselbe festgelegte Stelle gezeichnet und sind weder von der eigentlichen Planetenposition noch dem Blickfeld des Betrachters noch dessen Position im Raum abhängig. Diese Informationen werden in 3 separate Matrizen gespeichert und Opengl als zusätzliche Information übergeben. Nach der Multiplikation der Matrizen mit dem Positionsvektor erhält man dann ein perspektivisch korrektes Bild.

Die Umsetzung des 3d-Renderings war ein großer Aufwand, obwohl die eigentliche Arbeit nicht primär dieses Thema hat. Allerdings konnte nur mit dem visuellen Feedback die Software überhaupt zu einer realistischen Simulation weiterentwickelt werden. Das liegt daran, dass jedes komplizierte Programm während der Entwicklung verschiedene Fehler hat, die erst nach und nach behoben werden können. Bei einer einfacheren Form der Datenausgabe, wie zum Beispiel Textbasierte Positionsdaten wären Unstimmigkeiten im Bahnumlauf oder in der Konstellation der Planeten gar nicht, oder erst spät aufgefallen.
Aus diesem Grund war die Darstellung der Simulation als 3d-Welt die richtige Entscheidung für den Ausgang der Arbeit.

4. Ergebnisse

Die Simulation soll die Bewegung der Planeten möglichst realistisch darstellen, sie sollen also auf stabilen elliptischen Bahnen um die Sonne kreisen. Außerdem soll die Simulation in der Lage sein, Planetenkonstellationen zu einem gewissen Zeitpunkt realistisch darzustellen.
Zur Überprüfung können physikalische Werte ermittelt werden, die dann in der Simulation möglichst denselben Wert haben sollen wie in der realen Welt. Ein gutes Beispiel dafür sind Umlaufzeiten von Planeten.
Der Anspruch ist nicht, dass diese Größen genau übereinstimmen, denn es wurden Vereinfachungen in meinem Programm vorgenommen, wie zum Beispiel die Simulation mit Hilfe von Δt. Wichtiger wäre es, dass diese Abweichungen über die Simulationszeit hinweg stabil bleiben, denn wenn es eine steigende Abweichung gibt, kann ab einem bestimmten Zeitpunkt den Ergebnissen der Simulation nicht mehr vertraut werden.
Wenn dieser Anspruch also erfüllt wird, bleibt die Simulation über größere Zeiträume hinweg stabil und man kann beispielsweise einen näherungsweise richtigen Abstand der Erde zur Sonne im Jahr 2150 aus der Simulation ablesen.

4.1. Visueller Vergleich der Simulation mit Stellarium

Wie in ***3.4. Visualisierung*** schon erwähnt, ist die grafische Ausgabe der Planetenpositionen bei der Entwicklung eine große Hilfe gewesen, weil

offensichtlich falsche Bahnbewegungen schnell als Fehler erkannt werden konnten. Ein visueller Abgleich ist auch für die Startpositionen der Planeten möglich. Wie in ***1.2.*** **Datenquelle Stellarium** erwähnt kann man mit Stellarium wegen des detaillierten Nachthimmels die Planetenpositionen am Nachthimmel nachvollziehen. Dasselbe kann man der Simulationssoftware machen, indem man den Blickwinkel hinter die Erde setzt. Wenn man jetzt in beiden Programmen denselben Zeitpunkt verwendet, sollten die Planeten relativ zur Sonne aus Sicht der Erde dieselbe Position haben.

Abbildung 5 zeigt die Sonne zusammen mit den Planeten zum Zeitpunkt t_0 in Stellarium. Die Abbildung zeigt, dass die Planeten alle in etwa auf einer gedachten Geraden liegen. Außerdem kann man erkennen, welche Planeten rechts und links der Sonne stehen.

Abbildung 6 zeigt die gleiche Situation in der Simulation. Auch hier stehen die Planeten ungefähr auf der Geraden und die gleichen Planeten stehen links, beziehungsweise rechts der Sonne.

Daraus kann man schließen, dass der in ***3.3.2.*** **Umrechnung der Koordinaten** beschriebene Prozess zum Umrechnen von äquatorialen in kartesische Koordinaten ein realitätsnahes Ergebnis produziert.

4.2. Stabilität der Simulation bei großen Zeitabständen

Nach der Überprüfung der Anfangswerte, soll auch die Stabilität der Simulation getestet werden. Das wird gemacht, indem das Programm die Zeitspanne von 1000 Jahren simuliert. Wenn zum Zeitpunkt $T = t_0 + 1000a$ die Planeten noch in ihren Bahnen sind und auch der Abstand der Planeten untereinander so bleibt, wie man ihn im Sonnensystem erwartet, ist die Simulation stabil geblieben.

Abbildung 4 zeigt das Ergebnis: die Planeten haben ihre ursprüngliche Reihenfolge behalten und auch der Abstand unter den Planeten ähnelt dem der Startsituation.

4.3. Überprüfung der Ergebnisse anhand von Umlaufzeiten

Die bisherigen Überprüfungen zeigen, dass eine stabile Simulation geschaffen wurde, deren Objekte ungefähr richtige Anfangspositionen haben und die sich bewegen, wie man es von Planeten des Sonnensystems erwartet.

Um eine genauere Prüfung zu erreichen, werden während der Simulation Daten, zum Beispiel der Abstand eines Objektes zur Sonne in regelmäßigen Zeitabständen zusammen mit der momentanen Simulationszeit gespeichert. Hieraus entsteht ein Zeit-Abstand Diagramm. (vgl. *Abbildung 7)*

Mit dieser Herangehensweise kann man schnell und einfach Umlaufzeiten bestimmen. Denn auf einer elliptischen Bahn gibt es immer zwei charakteristische Punkte, das Perihel und das Aphel. Im Perihel ist der Abstand zur Sonne minimal, im Aphel maximal. [vgl. L2, S.14]

Das heißt aus den Minima und Maxima des entstandenen T-d Diagramms kann man auf die Umlaufzeit des Planeten schließen. Wenn man die Zeitwerte von zwei aufeinanderfolgenden Maxima subtrahiert, bekommt man die Zeit, die für einen vollen Umlauf des Planeten benötigt wird. Diese Zeit nennt man die siderische Umlaufzeit und der Realwert kann aus Stellarium ausgelesen werden. Wendet man nun dieses Prinzip auf den Mars an, erhält man *Abbildung 7* als Diagramm.

Berechnet man jeweils den Abstand zwischen den 3 Maxima des Diagramms und bildet den Durchschnitt, ist das Ergebnis 1,88 Jahre. Aus Stellarium kann man ablesen, dass die siderische Umlaufzeit des Mars 1,881 Jahre beträgt, das heißt dass die Abweichung der simulierten Umlaufzeit und der realen ungefähr im Bereich eines drittel Tages liegen.

Diese Abweichung wird dem Anspruch an die Genauigkeit gerecht, es muss aber überprüft werden, ob die Abweichung zur realen Umlaufzeit mit steigender Simulationszeit zunimmt.

Dafür werden 100 Jahre simuliert. Die Umlaufzeit wird wieder durch das Subtrahieren der Zeitwerte der Maxima berechnet und die Ergebnisse in ein Diagramm übertragen. Das Ergebnis ist *Abbildung 8*.

Man kann, wie schon im vorigen Versuch feststellen, dass es eine Abweichung zwischen realem Wert und Messwert gibt. Es gibt aber keine erkennbare Tendenz zu einer größeren Abweichung mit zunehmender Simulationszeit. Der Mittelwert der Zeiten liegt gerundet bei 1,88 Jahren. Aus den Daten kann man schließen, dass die Abweichungen zwischen Simulation und Realität auch für größere Simulationszeiten stabil bleiben.

Dasselbe Prinzip der Überprüfung kann man auf die Erdbahn anwenden. *Abbildung 9 zeigt* das T-d-Diagramm für die Erdbahn. Um verlässlichere Abweichungswerte zu bekommen und zu überprüfen, ob die Abweichung zunimmt, wird ein Zeitraum von 50 Jahren ausgewertet.

Abbildung 10 zeigt die Umlaufzeit der Erde in der Simulation in Jahren. Wie schon bei dem Diagramm für den Mars kann man erkennen, dass die Abweichung keine Tendenz mit der Simulationszeit zeigt. Man kann also annehmen, dass die Erdbahn auch für größere Simulationszeiten zuverlässig simuliert werden kann.

4.4. Abhängigkeit von Δt

Wie schon in **2.1. Numerischer Lösungsansatz** erklärt wurde, ist die Genauigkeit der Simulation an den Wert Δt geknüpft. Für alle bisherigen Ergebnisse wurde der Wert 30 Sekunden verwendet.
Jetzt soll anhand der Umlaufzeit der Erde bestimmt werden, wie groß der Einfluss von Δt auf die Werte der Simulation ist.
Die Erdumlaufzeit wird wie gerade eben bestimmt, indem ein Diagramm mit dem Erdabstand zur Sonne auf der y-Achse und der Simulationszeit auf der x-Achse erstellt wird. Die Umlaufzeit kann über die Abstände zwischen den Maxima bestimmt werden.
Dann wird der Betrag der Abweichung von der erwarteten Umlaufzeit bestimmt. Da die Umlaufzeit der Erde 1 Jahr beträgt gilt $\overline{\Delta T_{sid}} = \overline{|1a - T_{sid}|}$. Dieser Mittelwert wird über ungefähr 10 simulierte Jahre gebildet.
Der Wert $\overline{\Delta T_{sid}}$ wird jetzt für mehrere Δt bestimmt. Das Ergebnis kann man in *Abbildung 11* sehen. In dem Diagramm wurde $\overline{\Delta T_{sid}}$ auf der y-Achse und die verwendeten Werte für Δt auf der x-Achse dargestellt.
Man kann eine Kurve erkennen, die Richtung $\Delta t = 1s$ stark abflacht. Zwischen 4000s und 9000s ist die Steigung groß. Bis 8000s sieht der Verlauf stark nach einer exponentiellen Steigung aus, dann nimmt die Steigung aber wieder ab. Im Intervall [9000s; 15000s] ist die Steigung annähernd linear.
Anhand dieser Graphik kann man erkennen, dass der Wert von 30 Sekunden für Δt gut geeignet ist. Erstens ist wichtig unter 4000 Sekunden zu bleiben, weil danach die Abweichung stark ansteigt. Wie auch schon in **2.1.** erklärt wurde, kann Δt nicht unbegrenzt klein werden, weil damit auch der Rechenaufwand steigt, also wurde der Wert 30 Sekunden gewählt, weil erstens ein noch kleinerer Wert die Geschwindigkeit des Programms zu stark beeinträchtigt hätte und zweitens eine Verkleinerung von Δt keine wesentliche Verbesserung mehr brächte.

5. Fazit

Abschließend kann ich sagen, dass ich mit der Genauigkeit meiner Simulation zufrieden bin. Es wurde gezeigt, dass Größen wie Umlaufzeiten erhalten werden und auch die Konstellation der Planeten der Realität mindestens nahekommt.
Die Simulation kann nicht als Datenquelle für wissenschaftliches Arbeiten verwendet werden, das war aber auch nie der Anspruch. Dafür wurden die physikalischen Gesetze, sowie man sie in der Natur beobachten kann in dem Programm umgesetzt.

Anhang

Abbildungsverzeichnis

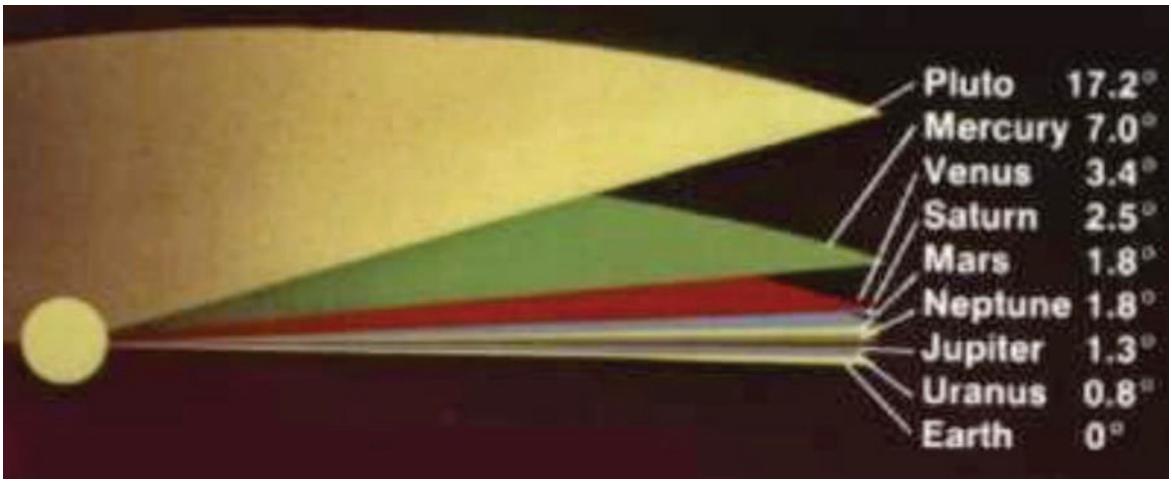

Abbildung 3: Schiefe der Bahnebenen der Planeten [Bildquelle: I2, S.18]

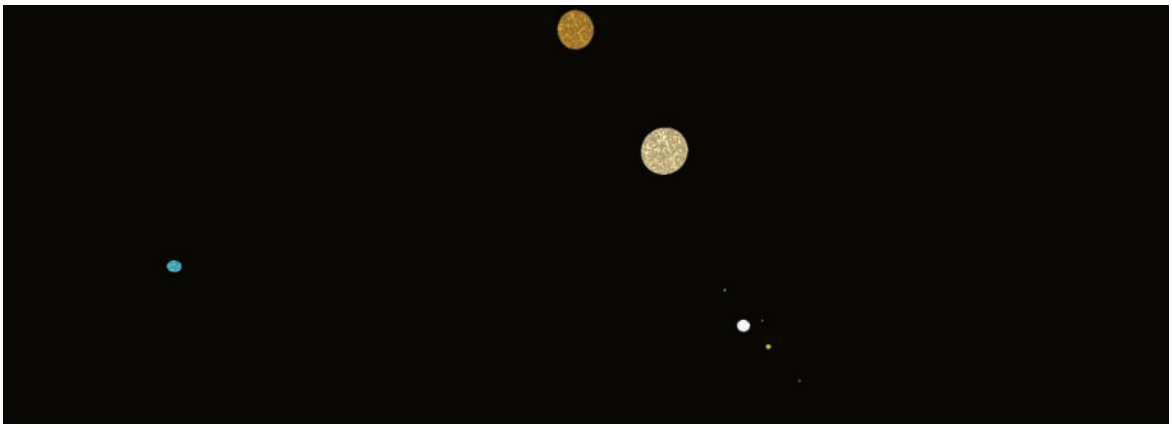

Abbildung 4: Stand der Simulation nach 1000 simulierten Jahren

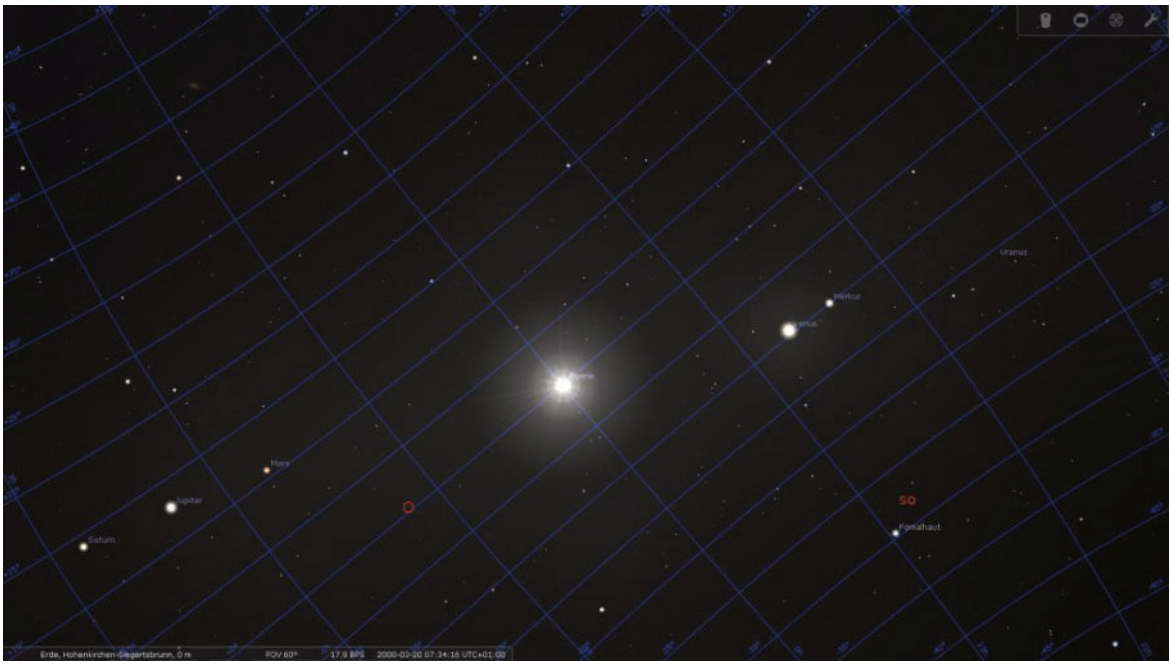

Abbildung 5: Sonne am Frühlingspunkt 2000

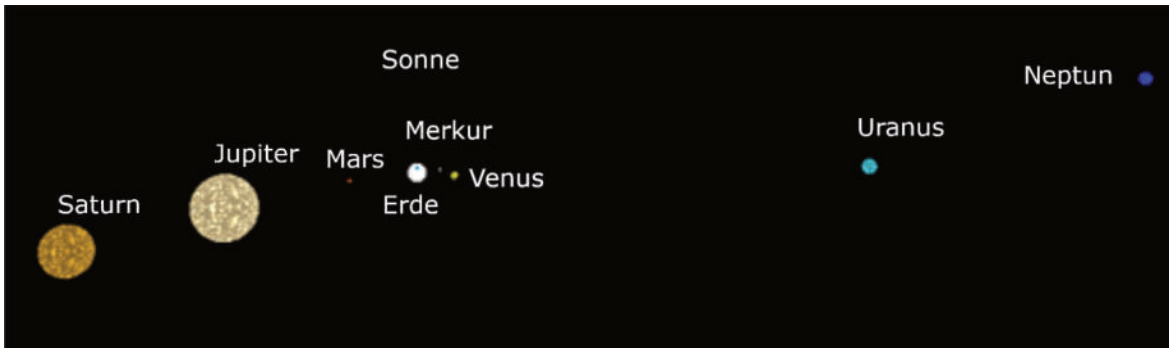

Abbildung 6: Planeten in der Simulation zum Zeitpunkt t_0

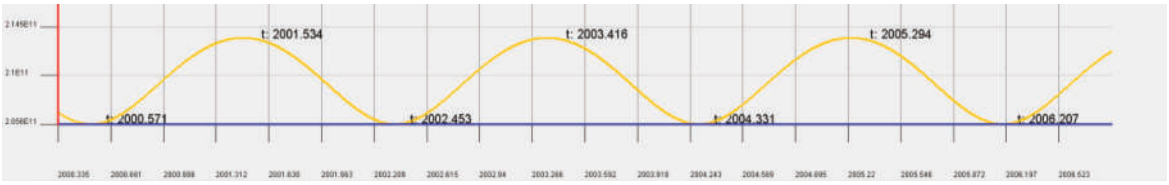

Abbildung 7: $|(x\_M\ x\_S)\ \vec{}|$-T Diagramm (Entfernung von Mars und Sonne)

Abbildung 8: (simulierte) siderische Umlaufzeit des Mars zwischen 2000 und 2100 in Jahren

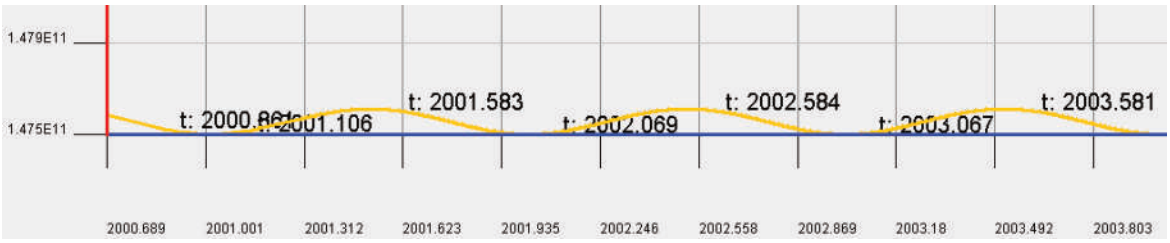

Abbildung 9: $|\overrightarrow{x_E x_S}| - T\ Diagramm\ (Entfernung\ von\ Erde\ und\ Sonne)$

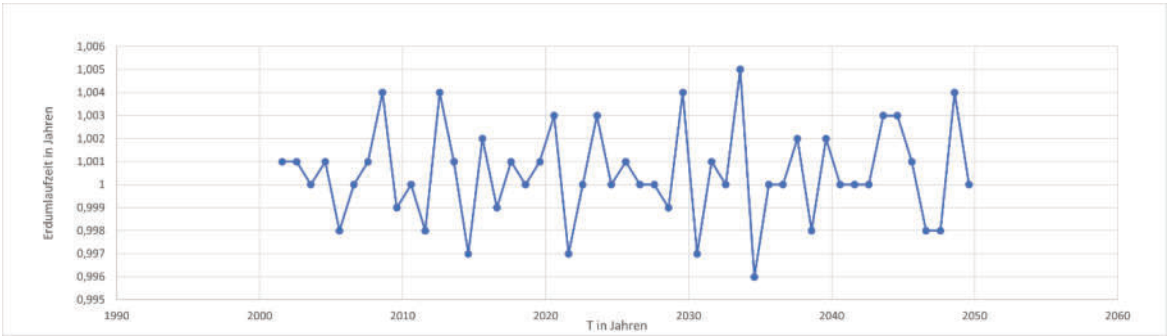

Abbildung 10: (simulierte) Umlaufzeit der Erde im Zeitraum 2000 bis 2050 in Jahren

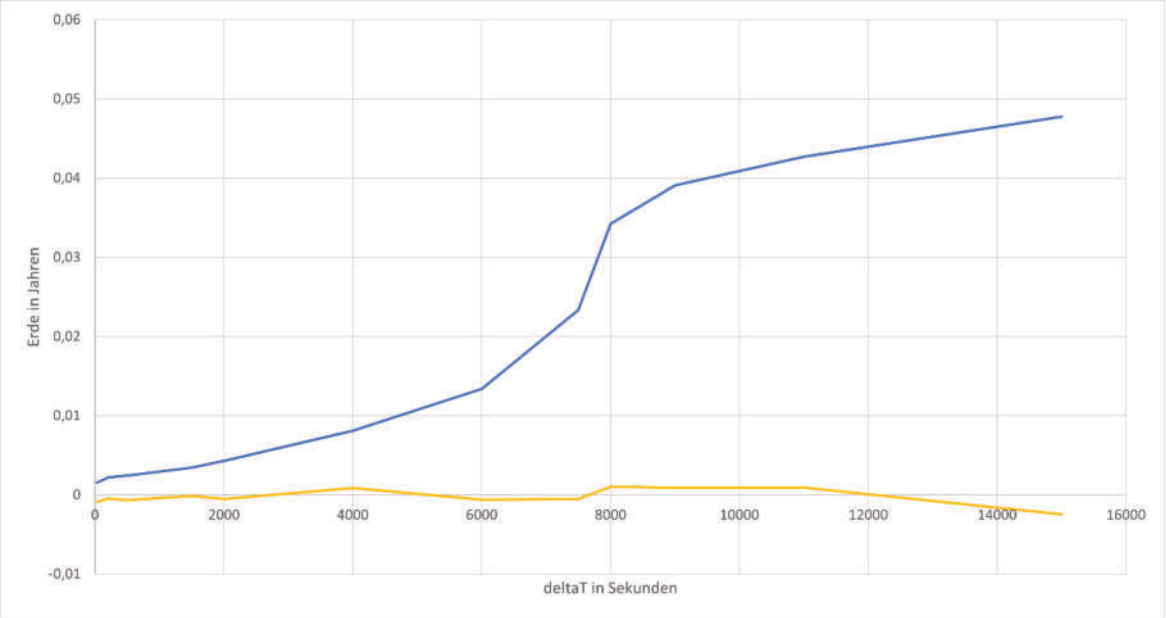

Abbildung 11: Auswirkungen von Δt auf Abweichungen in der Erdumlaufdauer

Quellcodeverzeichnis

Quellcode 1

public boolean running = **true**; (solange *running* true ist, läuft die Schleife in *run*)
physikObjekte enthält alle erschaffenen Objekte, also die Sonne und alle Planeten, die in der Simulation dargestellt werden.

```
public static List<PhysikObjekt> physikObjekte = new ArrayList<>();

public static double deltaT = 30; // die Zeit, die in der Simulation ein Zeitschritt einnimmt in Sekunden
public static double timePassed = 0; // Zeit die seit Simulationsstart vergangen ist (T)
```

Die Methode *run* wird beim Programmstart ausgefährt und enthält die folgende while-Schleife, die so lange die Simulation vorantreibt, bis die Variable *running* false wird (durch äußeren Einfluss).

```
public void run() {
   while (running) {
      updatePhysics();
            (hier liegt Code, der die Programmsteuerung mit der Tastatur regelt)
   }
   Scene.fb.finish();
   System.exit(0); (beendet das Programm und speichert eventuelle Ergebnisse)
```

```
}
```

Die Methode *updatePhysics* wird, in *run* aufgerufen und iteriert über die Liste *physikObjekte*. Für jedes PhysikObjekt in der Liste wird die Methode update aufgerufen. Abschließend wird die Methode updateTime aufgerufen.

```
private void updatePhysics() {
   for(int i = 0;i < physikObjekte.size(); i++) {
      physikObjekte.get(i).update(physikObjekte);
   }
   for(int i = 0;i < physikObjekte.size(); i++) {
      physikObjekte.get(i).commitNewPos();
   }
   updateTime();
}
```

Die eigentliche Funktionalität zur Berechnung von $\vec{x}_i(t + \Delta t)$ befindet sich in der Methode *update* in *PhysikObjekt*, die in *updatePhysics* für jedes Element von *physikObjekte* aufgerufen wird. Die Berechnung basiert auf den Werten für den Zeitpunkt t. Nachdem in der Methode *updatePhysics* $\vec{x}_i(t + \Delta t)$ für jedes $i = 1, \dots, N$ berechnet wurde, wird als letztes die Systemzeit T in der Methode *updateTime* erhöht.

```
private static void updateTime() {
   timePassed = timePassed + deltaT;
}
```

Quellcode 2

Quellcode aus der Methode *update* der Klasse *PhysikObjekt*:

```
Vector3d f = new Vector3d(0, 0, 0);
for (int j = 0; j < physikObjekte.size(); j++) {
   PhysikObjekt Oj = physikObjekte.get(j);
   if (Oj == this) continue; // i != j

   Vector3d diffVektor = Oj.pos.sub(pos);
   double r = diffVektor.length();
   double mi = mass, mj = Oj.mass;
   double force = (PhysicThread.G * mj * mass) / Math.pow(r, 2);
```

```
    Vector3d fij = diffVektor.normalize().mulitply(force);

    f = f.add(fij);
}
Vector3d a = f.divide(mass);
```

Quellcode 3

Quellcode der Methode *berechneGeschwindigkeit* aus der Klass *PhysikObjekt*:

```
public void berechneGeschwindigkeit(double d0, double d1, double dt, double betragV, double
bahnebenenSchiefe, @org.jetbrains.annotations.NotNull PhysikObjekt sonne) {
    Vector3d sonnenVektor = pos.sub(sonne.pos).normalize();

    double cosAlpha = (Math.pow(d1, 2) - Math.pow(d0, 2) - Math.pow(betragV * dt, 2)) / (-2 * d0 *
betragV * dt);
    double alpha = Math.acos(cosAlpha);

    Matrix3d rotationsMatrixX = Maths.createRotationMatrixX(bahnebenenSchiefe);
    Matrix3d rotationsMatrixZ = Maths.createRotationMatrixZ(alpha);

    Vector3d rotatedSonnenVektor = rotationsMatrixZ.multiply(sonnenVektor);
    rotatedSonnenVektor = rotationsMatrixX.multiply(rotatedSonnenVektor);
    setSpeed(rotatedSonnenVektor.mulitply(betragV));
}
```

Quellcode 4

```
public Vector3d berechnePosition(EquatorialCoordinateSystem system) {
    double declinationAngle = (computedDeclination) * 2.0 * Math.PI;
    double rektaszensionAngle = computedRektaszension * 2.0 * Math.PI;

    Matrix3d rotationMatrixZ = Maths.createRotationMatrixZ(rektaszensionAngle);
    Matrix3d rotationMatrixY = Maths.createRotationMatrixY(declinationAngle);
    Matrix3d rotationMatrixX = Maths.createRotationMatrixX(SonnenSystem.schiefeDerEkliptik);

    Vector3d rotatedVector = system.earthSunVector;
    rotatedVector = rotationMatrixZ.multiply(rotatedVector);
    rotatedVector = rotationMatrixY.multiply(rotatedVector);
    rotatedVector = rotationMatrixX.multiply(rotatedVector);
```

```
    return rotatedVector.mulitply(distance);
}
```

Literaturverzeichnis

Literaturquellen

Quelle L1:

Georg Zotti, Susanne M. Hoffmann, Alexander Wolf, Fabien Chéreau, Guillaume Chéreau; The Simulated Sky: Stellarium for Cultural Astronomy Research; Journal of Skyscape Archaeology; Band 6 Ausgabe 2, Seiten 221-258

Quelle L2:

Dr. Tilman Pehle, Dr. Lutz Engelmann; Formelsammlung Naturwissenschaften Gymnasium Bayern; Cornelsen

Quelle L3:

Herbert Götz, Manfred Herbst, Christine Kestler, Hans-Georg Kosuch, Dr. Johannes Novotný, Barbara Sy, Thomas Thiessen, Arnold Zitterbart; Lambacher Schweizer 11 Mathematik für Gymnasien (Bayern); Ernst Klett-Verlag

Quelle L4:

Prof. August Schmid, Prof. Dr. Ingo Weidig; Lambacher Schweizer 10 Mathematik für Gymnasien (Bayern); Ernst Klett-Verlag

Internetquellen

Quelle I1:

Völkel Klaus; Vermessung der Wandrichtung aus Schattenpositionen eines Lotes zum Entwurf einer Sonnenuhr; www.bnv-bamberg.de; https://www.bnv-bamberg.de/home/ba2380/suhr/sonnenuhr.pdf

Quelle I2:

Robi Banerjee; Einführung in die Astronomie & Astrophysik I Teil V Himmelsmechanik Gezeiten; www.physik.uni-hamburg.de; https://www.physik.uni-hamburg.de/en/hs/group-banerjee/_documents/teaching/ws-15-16-astro/astro-einf-1-ws15-vl05.pdf; Hamburger Sternwarte

Quelle I3:

Dr. Markus Mugrauer; Aufsuchen und Klassifizieren astronomischer Beobachtungsobjekte; www.astro.uni-jena.de; https://www.astro.uni-jena.de/Users/markus/Astropraktikum/Versuch-A.pdf; Astrophysikalisches Institut und Universitätssternwarte Jena

Quelle I4:

Oliver Leersch; Das N-Körper Problem eine approximative Lösung mit Python.; unipub.uni-graz.at; https://unipub.uni-graz.at/obvugrhs/content/titleinfo/2581338/full.pdf; Karl-Franzens-Universität Graz

Quelle I5:

Matthias Kemper, Guido Willems; Das 3-Körper Problem; www.uni-muenster.de; https://www.uni-muenster.de/imperia/md/content/physik_ft/pdf/ws1112/seminar/111918/willems-kemper.pdf; Universität Münster

Quelle I6:

Joachim Herz Stiftung; Astronomische Koordinatensysteme; www.leifiphysik.de; https://www.leifiphysik.de/astronomie/sternbeobachtung/grundwissen/astronomische-koordinatensysteme

Erklärung zur Seminararbeit

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche gekennzeichnet habe.

Bruckmühl, den ..... ……

Unterschrift