UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Simulation zur Darstellung von Objekten im Weltall und deren Wechselwirkung zueinander

Studienarbeit

im Studiengang Computervisualistik

vorgelegt von

Thorsten Schönbrunn

Betreuer: Dipl.-Inform Dominik Grüntjes
(Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Juni 2013

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

Ort, Datum Unterschrift

Zusammenfassung

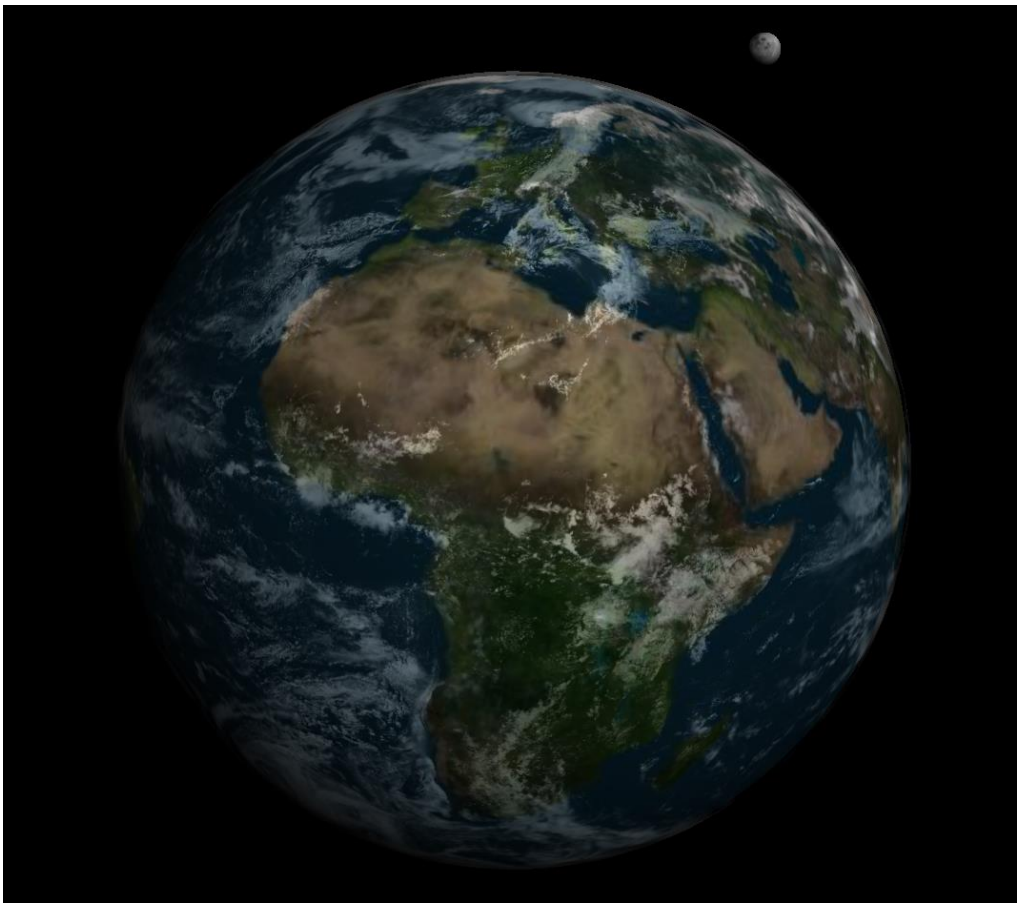
Die vorliegende Arbeit behandelt die Entwicklung einer Simulationsumgebung zur Darstellung von Objekten im Weltraum und ihrer gravitativen Wechselwirkung zu einander.

Vorab werden in Kapitel 1 Motivation und Zielsetzung der Arbeit erläutert, des Weiteren werden die verwendeten Werkzeuge benannt. Die nötigen astronomischen Grundlagen werden in Form von Begriffserklärungen und der Vorstellung der dieser Arbeit zugrunde liegenden physikalischen Gesetze in Kapitel 2 beschrieben.

Kapitel 3 befasst sich mit dem Aufbau der einzelnen Klassen. Hier wird insbesondere auf die Berechnung der Positionen und Geschwindigkeiten der simulierten Himmelskörper und den Aufbau und die Funktionsweise der verwendeten Elemente der Graphikengine Ogre3D eingegangen.

Im Kapitel 4 wird der Einsatz des Werkzeugs 3ds Max zur Erstellung der Geometrieobjekte und Materialien erläutert.

Abschließend wird in Kapitel 5 ein Fazit gezogen und mögliche zukünftige Erweiterungen erwogen.



Inhalt

1. Einleitung

1.1 Motivation

Den 15. Februar 2013 hatten sich Astronomen aus aller Welt seit längerem im Kalender markiert, erwartete man doch für 20:24 Uhr mitteleuropäischer Zeit „2012 DA 14“, einen mit ca. 50 Metern Durchmesser zugegebenermaßen kleinen Asteroiden, welcher die Erde jedoch in geschätzten 30.000 Kilometer Entfernung passieren sollte. Zum Vergleich: geostationäre Satelliten findet man in 35.786 km Höhe, weshalb 2012 DA 14 zwar keine Gefahr für unseren Planeten, potentiell jedoch für unsere künstlichen Objekte im Erdorbit darstellte.

In den Mittelpunkt des Tages sollte dann aber ein kleinerer Kollege des erwarteten Asteroiden rücken: mitten im morgendlichen Berufsverkehr russischer Ortszeit traf ein wesentlich kleinerer Meteorit die russische Region Tscheljabinsk ca. 1500 km östlich von Moskau, explodierte in einigen Kilometern Höhe und verursachte durch die Druckwelle und herabstürzende Gesteinsbrocken rund 950 Verletzte und zahlreiche Schäden an Gebäuden. Dank der in dieser Region nicht unüblichen Autokameras fanden sich in Internet und Medien recht schnell einige Videos, welche das Ereignis dokumentierten.

Es folgten die üblichen quotenversprechenden Katastrophensendungen, gespickt mit teils aufwendigen Animationen unseres Sonnensystems nebst allerlei Informationen um dessen Entstehung, die Planeten, Monde und auf welche möglichen Arten unser Planet einst zugrunde gehen würde.
An dieser Stelle stellte mein sieben Jahre alter Sohn mir ein paar Fragen, wie sie nur Kinder formulieren können, von denen mich drei letzten Endes zu vorliegender Arbeit animierten: „Wer hat denn das da alles gefilmt, und mit welchem Raumschiff waren die da? Außerdem, wieso fliegen die Planeten dann immer so im Kreis um die Sonne?“ und, als Folge auf meine Antwort, man könne dies heute mit Computern so schön darstellen, „Du machst doch auch an der Uni so mit Computern, kannst Du so etwas auch mal machen?“ …

1.2 Zielsetzung

Das Ziel dieser Studienarbeit soll eine Simulationsumgebung sein, in welcher die Bewegung der Planeten und Monde unseres Sonnensystems unter Berücksichtigung bekannter Eigenschaften der Objekte wie etwa Masse, Position und Geschwindigkeit zu einem fixen Zeitpunkt, nach Berechnungsmodellen der Astrophysik bestimmt und grafisch dargestellt werden soll. Hierbei soll besonders der gegenseitige Einfluss der Gravitation der Himmelskörper auf einander berücksichtigt werden. Final sollte es möglich sein, „neue“ Objekte, beispielsweise Asteroiden, in dieses System einzubringen und deren Verhalten zu beobachten. Es sei gesagt, dass hier keinesfalls der Anspruch einer absolut genauen Simulation vorliegt, welche die exakten Positionen aller Objekte unseres Sonnensystems für einige Millionen Jahre bestimmen kann. Hierzu wäre, abgesehen von bekannten und sicherlich berechenbaren relativistischen Faktoren, auch Kenntnis von bisher nur sehr vage bekannten Störfaktoren und -Einflüssen notwendig (Stichwort „dunkle Materie“).

Zur Realisierung dieses Vorhabens muss, neben der Recherche astronomischer Fakten sowie mathematisch/physikalischer Berechnungsmodelle, eine geeignete 3D-Engine zur grafischen Darstellung gefunden werden. Zwecks Benutzerfreundlichkeit und besseren Bedienbarkeit sollte diese idealerweise bereits über ein GUI (Graphical User Interface) verfügen oder zumindest um ein solches Erweiterbar sein.

Darüber hinaus müssen geeignete Werkzeuge zum Erstellen der 3D-Objekte und der grafischen Elemente hinzugezogen werden.

1.3 Werkzeuge

Für die Wahl der Programmiersprache wurde sich an den Vorlesungen der Arbeitsgruppe Computergraphik der Universität Koblenz-Landau orientiert: hier wird hauptsächlich C++ in Verbindung mit OpenGL eingesetzt, weshalb diese auch in dieser Studienarbeit unter Microsoft Visual Studio 2010 eingesetzt wird.

Als passende 3D-Engine fiel die Wahl auf Ogre3D, eine OpenSource-Engine, welche einerseits über die freie Wahl des Renderers (OpenGL oder DirectX 9), andererseits über einen umfangreichen Scenemanager, grundlegende Verwaltung von Eingabegeräten (Maus, Tastatur, Joystick etc.) und zahlreiche Erweiterungsmöglichkeiten verfügt, nicht zuletzt dank einer aktiven Community. Zu diesen gehört auch ein grafisches Benutzerinterface namens CEGUI, welches sich relativ problemlos in Ogre3D-Projekt integrieren lässt. Zwar unterstützt Ogre3D in der verwendeten Version 1.8.3 des SDKs nur DirectX 9, der gegen Ende dieser Arbeit veröffentliche Release Candidate der Nachfolgeversion 1.9 verfügt nun über DirectX 11, ferner OpenGL 3+-Unterstützung.

Die 3D-Objekte wie etwa Planeten und Asteroiden wurden mit Autodesk 3ds Max 2013 entworfen, welches dankenswerterweise für Studenten kostenfrei angeboten wird. Auf grafischer Seite, etwa für Texturen, kam Adobe Photoshop CS 6 zum Einsatz, ebenfalls als Testversion kostenfrei erhältlich.

2. Astronomische Grundlagen

2.1 Astronomische Begriffe & Gesetze

2.1.1 Himmelsmechanik

Bereits im 3. Jahrtausend v. Chr. versuchten die Bewohner Mesopotamiens, die Bewegungen von Sonne und Mond vorher zu sagen. Den Ägyptern gelang es zur etwa selben Zeit, durch Beobachtung des Sterns Sirius, die Dauer eines Erdenjahres auf 365,25 Tage zu bestimmen. Diese schon früh von der Menschheit begründete und stetig weiterentwickelte Lehre von der Bewegung astronomischer Objekte aufgrund mathematischer Berechnungen und physikalischer Theorien trägt den Namen Himmelsmechanik.

Ein Beispiel für solche mathematischen Modelle bieten die Kepler'schen Gesetze, welche sich durch Newtons Gravitationsgesetz begründen lassen.

2.1.2 Kepler'sche Gesetze

Der Astronom und Naturphilosoph Johannes Kepler formulierte zwischen 1609 und 1618 drei heute nach ihm benannte Gesetze, welche die Bewegung idealer Himmelskörper beschreiben.

1. Kepler'sches Gesetz: *Die Umlaufbahn eines Trabanten ist eine Ellipse. Einer ihrer Brennpunkte liegt im Schwerezentrum des Systems.*
Wir wissen heute, dass das Schwerezentrum unseres Sonnensystems nicht etwa genau die Sonne ist. Diese bewegt sich vielmehr ebenfalls um diese, auch als Baryzentrum bezeichnete, Position. Gleiches gilt zum Beispiel für unsere Erde und ihren Mond. Auch dieser umkreist nicht etwa die Erde, sondern beide bewegen sich auf Bahnen um ein gemeinsames Baryzentrum, welches jedoch in diesem Fall stets innerhalb der Erde liegt. Ursache dafür ist die Wechselwirkung der Gravitationen der Objekte zu einander.

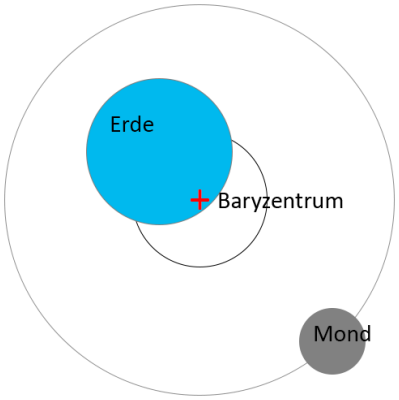


Abb. 1: Baryzentrum im 2-Körper-System

2. Kepler'sches Gesetz: *In gleichen Zeiten überstreicht der Fahrstrahl Objekt-Schwerezentrum gleiche Flächen.*

Unter der Bezeichnung Fahrstrahl versteht man die Verbindungslinie zwischen dem Schwerpunkt eines Objekts (z.B. des Mondes) und dem Baryzentrum des Systems (im Beispiel das Erde-Mond-System), um welches es sich bewegt. Anders ausgedrückt bedeutet dieses Gesetz für die Geschwindigkeit eines Objektes auf seiner Bahn: je näher das Objekt auf seiner Bahn an das Schwerezentrum gerät, desto schneller bewegt es sich.

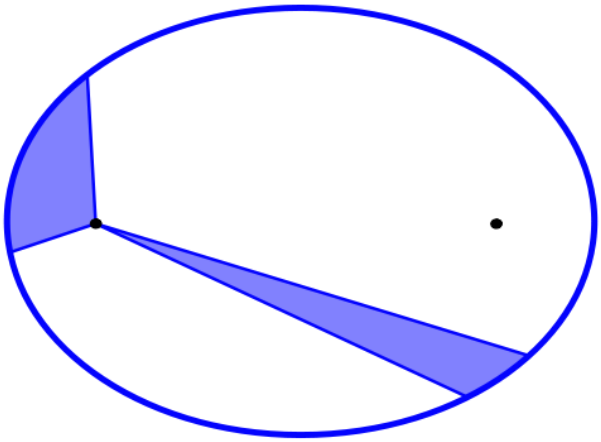


Abb. 2: 2. Kepler'sches Gesetz, Quelle: Wikipedia

3. Kepler'sches Gesetz: *Die Quadrate der Umlaufzeiten stehen im gleichen Verhältnis wie die Kuben der großen Halbachsen.*

$$\left(\frac{U_1}{U_2}\right)^2 = \left(\frac{a_1}{a_2}\right)^3$$

Kepler setzte zwei Planeten des Sonnensystems zu einander in Beziehung und versuchte auf diese Weise Gesetzmäßigkeiten zu finden. Er stellte schließlich fest, dass ein regulärer Zusammenhang zwischen der Größe der Umlaufbahn eines Planeten und der Zeit, welche dieser für diese Bahn benötigt, besteht. Setzt man diese Größen ins Verhältnis zu einander, erhält man einen konstanten Wert, welcher für alle Objekte des Systems gilt.

$$\frac{a^3}{U^2} = \text{konstant}$$

Vereinfacht gesagt: Kennt man den mittleren Abstand eines Planeten zur Sonne sowie seine Umlaufzeit, kann man für jeden weiteren Planeten den mittleren Abstand bestimmen, sofern man Kenntnis über dessen Umlaufzeit besitzt.

2.1.3 Newton'sches Gravitationsgesetz & Gravitationskonstante

Isaac Newton formulierte 1686 sein Gravitationsgesetz, welches besagt, dass jeder Massenpunkt jeden anderen Massenpunkt mit einer entlang der Verbindungslinie gerichteten Kraft anzieht.

$$F = G\,\frac{m_1\,m_2}{r^2}$$

Hierbei entsprechen m_1 und m_2 den Massen der beiden Massepunkte, r beschreibt den Abstand zwischen den Massepunkten. G ist die Gravitationskonstante, welche die Masse mit der Gravitation verknüpft. Im Allgemeinen wird die Gravitationskonstante als diejenige Naturkonstante mit der größten relativen Messungenauigkeit bezeichnet. Ursache ist die Schwierigkeit, welche sich ergibt, möchte man die Gravitationskraft im Laborversuch mit hoher Genauigkeit messen: ein Objekt von der Größe und Masse eines Planeten könnte zwar adäquate Messergebnisse liefern, doch stehen uns bis dato im Labor doch eher kleinere Testobjekte mit sehr viel geringerer Gravitationskraft zur Verfügung, weshalb selbst mit aufwändig konstruierten und gegen Störeinflüsse abgeschirmten Messgeräten lediglich eine Messgenauigkeit von $1{,}0 \cdot 10^{-4}$ erreicht wird. Die Gravitationskonstante ist nach CODATA 2010 wie folgt definiert:

$$G = 6{,}67384\,(80) \cdot 10^{-11}\,\frac{m^3}{kg \cdot s^2} \quad (\,bzw.\ \frac{N \cdot m^2}{kg^2}\,)$$

mit einer Standardunsicherheit von $0{,}00080 \cdot 10^{-11}$.

In vektorieller Form lässt sich Newtons Gesetz für die auf einen Massepunkt 1 wirkende Kraft $\overrightarrow{F_1}$ wie folgt beschreiben:

$$\overrightarrow{F_1} = G\,m_1 m_2\,\frac{\overrightarrow{r_2} - \overrightarrow{r_1}}{|\overrightarrow{r_2} - \overrightarrow{r_1}|^3}$$

wobei $\overrightarrow{r_1}$ und $\overrightarrow{r_2}$ die jeweilige Position der beiden Objekte repräsentieren. Obige Formel gilt für 2 aufeinander wirkende Objekte. Nun besagt das Newton'sche Gravitationsgesetz, dass jeder Massepunkt jeden anderen anzieht. Also muss die Formel entsprechend angepasst werden: die auf den Massepunkt 1 wirkenden Kräfte der anderen Massepunkte 2 bis n addieren sich zu einer Gesamtkraft

$$\overrightarrow{F_1} = G\,m_1 \sum_{i=2}^{n} m_i\,\frac{\overrightarrow{r_i} - \overrightarrow{r_1}}{|\overrightarrow{r_i} - \overrightarrow{r_1}|^3}$$

2.1.4 Gravitationsparameter

Als Gravitationsparameter eines Himmelskörpers bezeichnet man das Produkt seiner Masse mit der Gravitationskonstante, also

$$\mu = G \cdot M \quad \text{, mit der Einheit} \quad m^3 s^{-2}$$

Verfügt ein Himmelskörper über Begleiter, deren Umlaufbahn gemessen werden kann, so ist μ wesentlich genauer zu bestimmen als die Masse M des Himmelskörpers aus dem Durchmesser und Dichteverlauf geschätzt werden könnte.

Für kleinere Objekte mit einer Masse m, welche sich auf einer Bahn um ein großes Zentralgestirn mit Masse M befinden, gelten folgende Gleichungen:

sei $m \ll M$, *so gilt für Objekte auf kreisförmiger Umlaufbahn um ein zentrales Objekt*

$$\mu = r v^2 = r^3 \omega^3 = \frac{4\pi^2 r^3}{T^2}$$

mit r als Radius der Umlaufbahn, v ist die Bahngeschwindigkeit, ω die Winkelgeschwindigkeit und T die Umlaufzeit.

Eine einfache Verallgemeinerung für elliptische Umlaufbahnen lässt sich aus der letzten Gleichung bilden:

$$\mu = \frac{4\pi^2 a^3}{T^2}$$

wobei a für die Halbachse der Ellipse steht.

2.1.5 Ephemeriden

Als Ephemeriden bezeichnet man Tabellen und Tafelwerke, welche die Positionen von sich bewegenden astronomischer Objekte in konstanten Zeitabständen enthalten.

Solche Ephemeriden ermöglichen beispielsweise die genauere Bestimmung der Bahnentwicklung von Asteroiden und Kometen, aber auch zur Positionsbestimmung von Weltraumschrott, welcher eine Gefahr für die Raumfahrt darstellt und in die Missionsplanung einbezogen werden muss.

```
*******************************************************************************
Ephemeris / WWW_USER Wed Jun  5 16:46:28 2013 Pasadena, USA      / Horizons
*******************************************************************************
Target body name: Earth (399)                     {source: DE405}
Center body name: Solar System Barycenter (0)     {source: DE405}
Center-site name: BODY CENTER
*******************************************************************************
Start time      : A.D. 2013-Apr-05 00:00:00.0000 CT
Stop  time      : A.D. 2013-Apr-10 00:00:00.0000 CT
S[illegible]
$$SOE
2456387.500000000 = A.D. 2013-Apr-05 00:00:00.0000 (CT)
  -1.445385852480334E+08 -3.957753164823467E+07 -7.261333415034002E+03
   7.319472344574915E+00 -2.886762457977559E+01  4.337277734721336E-04
2456388.500000000 = A.D. 2013-Apr-06 00:00:00.0000 (CT)
  -1.438847884047775E+08 -4.206587402580588E+07 -7.212293630059176E+03
   7.814347944846779E+00 -2.873140568845881E+01  7.010476754749032E-04
2456389.500000000 = A.D. 2013-Apr-07 00:00:00.0000 (CT)
  -1.431883333159902E+08 -4.454206100772075E+07 -7.140486288766882E+03
   8.306883103451574E+00 -2.858624578773030E+01  9.582177354014532E-04
2456390.500000000 = A.D. 2013-Apr-08 00:00:00.0000 (CT)
  -1.424494349287006E+08 -4.700531972041687E+07 -7.047418361717455E+03
   8.796781976149290E+00 -2.843214369388602E+01  1.190998590240411E-03
2456391.500000000 = A.D. 2013-Apr-09 00:00:00.0000 (CT)
  -1.416683338727267E+08 -4.945487908920528E+07 -6.935727503687864E+03
   9.283747770998742E+00 -2.826914196427319E+01  1.387567540610059E-03
2456392.500000000 = A.D. 2013-Apr-10 00:00:00.0000 (CT)
  -1.408452960712881E+08 -5.188997350340348E+07 -6.808948192452433E+03
   9.767493456725601E+00 -2.809732373888065E+01  1.539052833725610E-03
$$EOE
```

Abb. 3: Beispiel zu Ephemeriden, hier dem Web-Interface HORIZONS der NASA entnommen

2.2 Klassifizierung von Objekten im Weltraum

Nach aktuellem Kenntnisstand können Sternensysteme aus einem oder mehreren Sternen bestehen, welche sich anhand ihres Lichtspektrums klassifizieren lassen. Dank stetigem Fortschritt bei der Entwicklung neuer und besserer Messgeräte und Teleskope sowie immer leistungsstärkeren Computern, welche aufwändige Berechnungen in akzeptablen Zeiten ermöglichen, finden wir heute immer öfter

Hinweise auf weitere Objekte, welche sich in relativer Nähe des Zentralgestirns befinden oder dieses sogar auf einer stabilen Umlaufbahn umkreisen. Um eben solche Objekte, vom wenige Meter Durchmesser fassenden Meteoriten bis hin zum Gasriesen, besser beschreiben zu können, gelten auch hier Klassifizierungen, welche Auskunft über die Zusammensetzung, Masse, atmosphärische Bedingungen, Temperatur usw. liefern.

2.2.1 Sterne

Sterne werden heute durch verschiedene Klassifizierungen genauer beschrieben. Dazu zählt einerseits die Einteilung anhand ihrer Farbe, welche von der Oberflächentemperatur abhängt. Diese **Spektralklassen**[1] bezeichnet man durch die Buchstaben O, B, A, F, G, K, M, häufig werden noch weitere Klassen L, T, Y (braune Zwerge) und R, N, S (rote Riesen) genannt. Da erstere jedoch 99% der heute bekannten Sterne ausmachen, nennt man diese auch die 7 Grundtypen. O-Sterne bilden hier die heißesten mit bis zu 30.000 Grad Kelvin und einer blauen Farbe, Sterne der Klassen M, L, T und darunter leuchten im roten bis infraroten Bereich und haben eine Oberflächentemperatur von 3300 bis nur noch 200 Grad Kelvin.

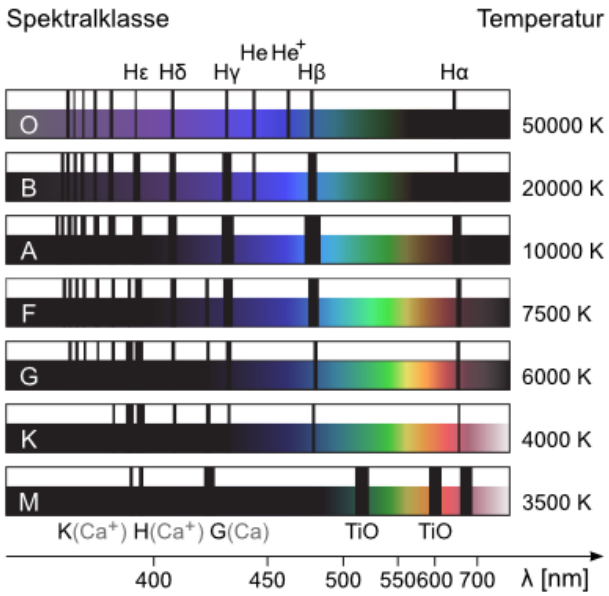


Abb. 4: Spektralklassen der Sterne Quelle (06.06.2013): http://lp.uni-goettingen.de/get/text/7004

Zur feineren Unterscheidung werden diese Klassen noch einmal in 10 Unterklassen von 0 bis 9 unterteilt, mit 0 als den heißesten bis 10 als den kältesten Sternen der jeweiligen Klasse. Unsere Sonne ist als G2-Stern also ein mit 5778 Kelvin ziemlich heißer Stern der G-Klasse.

Ferner werden Sterne anhand ihrer Leuchtkraft, also nach der absoluten Helligkeit unterteilt. Sterne eines gleichen Spektraltyps besitzen nämlich aufgrund unterschiedlicher Größen auch unterschiedliche Helligkeiten: je größer ein Stern, desto größer auch seine Oberfläche. Diese wiederum leuchtet bei gleicher Temperatur stärker, als bei einem kleineren Stern. So leuchtet beispielsweise ein roter Riese

[1] Quelle: ESA Link (05.06.2014): http://sci.esa.int/science-e/www/object/index.cfm?fobjectid=35774&fbodylongid=1700

wesentlich heller als ein roter Zwerg. Diese Leuchtkraftklassen werden in römischen Ziffern angegeben, wobei die Klasse I der sogenannten Überriesen noch einmal nach Ia und Ib unterteilt wird. Unsere Sonne gehört zur Klasse V, die geringste Leuchtkraft besitzen weiße Zwerge der Klasse VII.

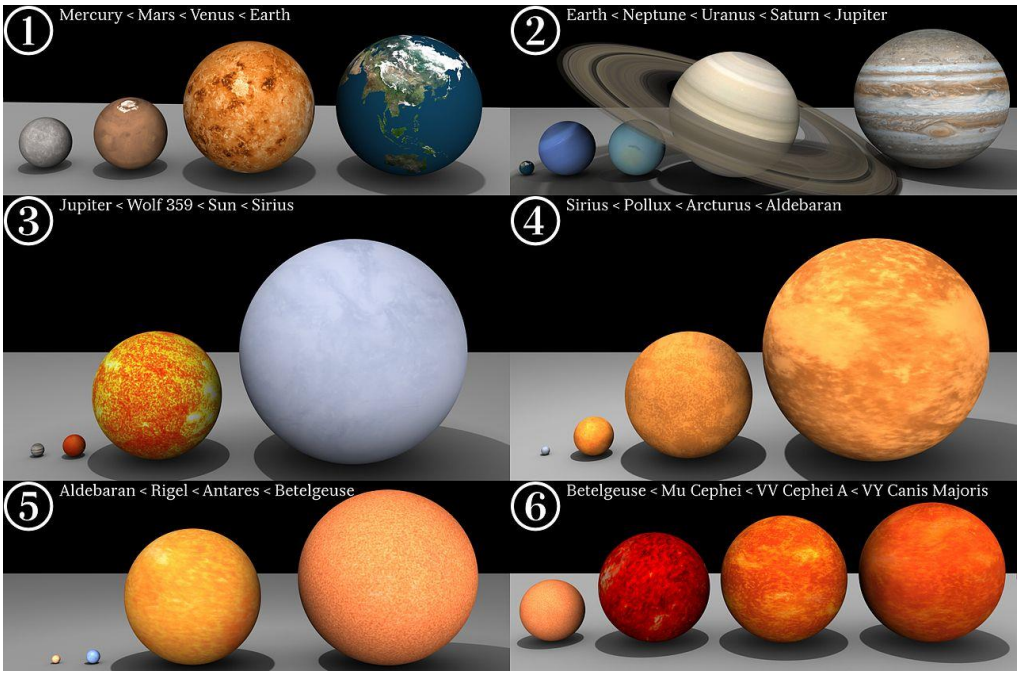


Abb. 5: Größenverhältnisse bekannter Sterne, im Vergleich die Planeten unseres Sonnensystems. Quelle: Wikipedia

Spektraltyp und Leuchtkraft können in Hertzsprung-Russel-Diagramm, kurz HRD, zu einander in Beziehung gesetzt werden. Mithilfe dessen untersucht man die Entwicklungszustände von Sternen.

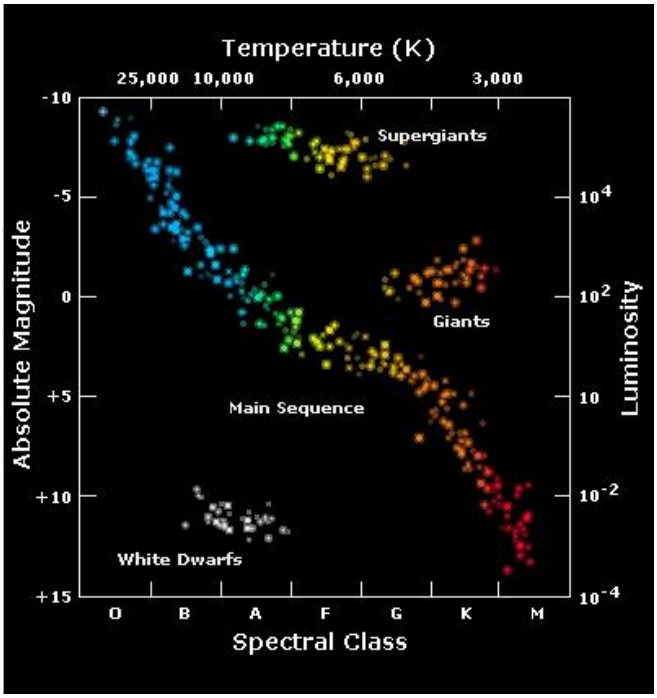


Abb. 6: Hertzsprung-Russell-Diagramm, Quelle: ESA

2.2.2 Planeten

Würde man heutzutage in einer Fußgängerzone willkürlich Passanten fragen, zu welchem Typ Planet die Erde zählt, erhielte man sicher nicht selten die Antwort „Die Erde? Natürlich ein Klasse M Planet.“ – Star Trek und Co lassen grüßen. Leider entstammt diese Einteilung lediglich der Filmindustrie und keiner wissenschaftlichen Grundlage.

Planeten werden heute vielmehr, bezogen auf unser Sonnensystem, nach ihrer Lage und ihrem Aufbau sowie ihrer Größe unterteilt. Lagetechnisch unterscheidet man die inneren und äußeren Planeten unseres Sonnensystems, getrennt durch den Asteroidengürtel. Zu den inneren Planeten zählen demnach Merkur, Venus, Erde und Mars, die Gruppe der äußeren bilden Jupiter, Saturn, Uranus und Neptun.

Bezüglich ihres Aufbaus spricht man von den terrestrischen oder auch erdähnlichen Planeten, welche in ihrem Aufbau der Erde gleichen, also vollständig oder fast vollständig aus festen Bestandteilen bestehen, zumeist gekennzeichnet durch einen Schalenartigen Aufbau. Die Erde bildet exemplarisch den Urtyp dieser Definition: ein fester innerer Nickel-Eisen-Kern, umgeben von einem äußeren Kernbereich geschmolzenem Eisens. Darüber der Mantel, zähplastisches Gestein (Silikate und Oxide) und letztlich die mit 0-35 km relativ dünne, harte Erdkruste.

Im Gegensatz dazu bestehen die sogenannten Gasplaneten überwiegend aus leichten Elementen wie Wasserstoff und Helium. Nach heutigem Wissensstand besitzen auch Gasplaneten einen festen Kern, den Großteil der Masse machen aber die leichten Elemente aus, welche im Inneren aufgrund des Drucks und niedriger Temperaturen auch in flüssigem und festem Aggregatzustand vorliegen. Die uns bekannten Gasriesen sind wesentlich größer und massereicher als unsere erdähnlichen Planeten, Jupiter und Saturn erreichen zusammen rund 1 ‰ der Sonnenmasse.

Die so definierten Planeten bilden eine von drei Klassen, welche die Internationale Astronomische Union (IAU) festgelegt hat. Eine weitere bilden die Zwergplaneten: Objekte, die sich auf einer Umlaufbahn um die Sonne befinden und das sogenannte Hydrostatische Gleichgewicht zu bilden, also durch Eigenrotation eine annähernde Kugelform zu halten. Zu diesen zählen als Unterklasse auch die Plutoiden, jene Zwergplaneten außerhalb der Bahn Neptuns. Pluto wurde 2006 von der IAU vom Planeten zu einem Plutoiden degradiert, neben ihm erfüllen Eris, Makemake und Haumea aktuell die Anforderungen eines Plutoiden. Innerhalb des Asteroidengürtels ist Ceres als größtes Objekt des Gürtels inzwischen als Zwergplanet anerkannt.

2.2.3 Sonstige Objekte

Die dritte Klasse bilden die Kleinkörper. Als solche versteht man Objekte, welche sich auf einer Bahn um die Sonne befinden, jedoch mangels ausreichender Masse über eben keine annähernde Kugelform verfügen und dazu auch keine Satelliten darstellen. Zu diesen gehören die, oft auch als Kleinplaneten oder Planetoiden bezeichneten, Asteroiden und die Kometen.

Anzumerken sei, dass diese Einteilung regelmäßig in der Kritik steht, nicht zuletzt dadurch, dass sie sich ausschließlich auf das Sonnensystem und nicht generell auf Planetensysteme bezieht.

3. Aufbau der Simulation

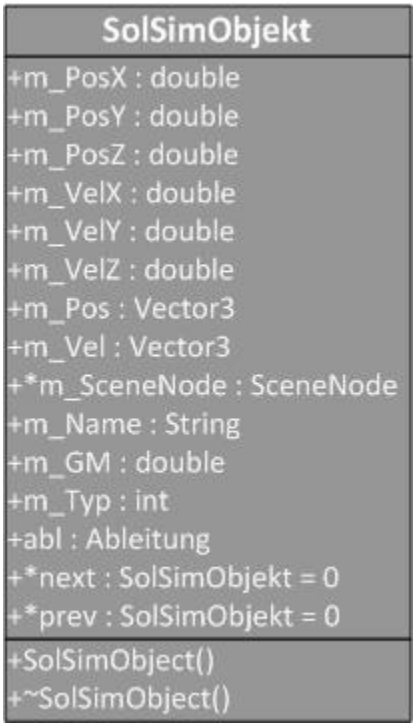
3.1 Übersicht

Sofern nicht anders beschrieben, werden in dieser Arbeit Wege in Kilometern, Zeiten in Sekunden und Massen in Kilogramm beschrieben und als solche auch in den Berechnungen verwendet.

Die im Rahmen dieser Studienarbeit entwickelte Simulation, im Folgenden kurz **SolSim** genannt, verfügt über 3 Klassen: *SolSimObject* und *SolSys* zur Verwaltung der Objekte des Sonnensystems inklusive der mathematischen Verfahren zur Berechnung der Umlaufbahnen anhand von Position, Geschwindigkeit und Gravitationsparameter, sowie S*olSim* mit der Einbindung der Grafikengine zur Darstellung des Sonnensystems, der Benutzeroberfläche und der Behandlung der Benutzereingaben per Tastatur und Maus. Diese Klassen werden in und dem folgenden Kapitel näher vorgestellt.

3.2 Klasse SolSimObject

SolSim soll allgemein Objekte im Sonnensystem und deren Wechselwirkung zu einander darstellen. Daher müssen neben der Sonne als Zentralgestirn sowohl die bekannten Planeten als auch deren Monde und wichtige Plutoiden und Planetoiden berücksichtigt werden. Folglich verfügt die hier vorgestellte Klasse über die wichtigsten Attribute und Funktionen, um in der Simulation ein Objekt vom Typ SolSimObject, sei es nun ein Planet, Mond oder Asteroid, repräsentieren zu können.



Für die zugrunde gelegten Bahnberechnungen werden Position *m_Pos* und aktuelle Geschwindigkeit *m_Vel* als dreidimensionale Vektoren sowie deren einzelne Koordinaten x, y und z vom Typ double verwendet.

Jedes Objekt verfügt zudem über einen Zeiger *m_SceneNode* auf den ihm zugehörigen Knoten im Szenegraphen der Simulation, auf welchen in Kapitel 4 näher eingegangen wird. Diese Knoten können komfortabel angesprochen werden, um etwa eine Kamera direkt an den ihnen zugewiesenen Objekten zu positionieren.

Wie bereits in Kapitel 2.1.4 beschrieben, lässt sich der Gravitationsparameter eines Himmelskörpers wesentlich genauer bestimmen als man die Masse schätzen könnte und eignet sich daher besser zur Berechnung der Gravitationskräfte.

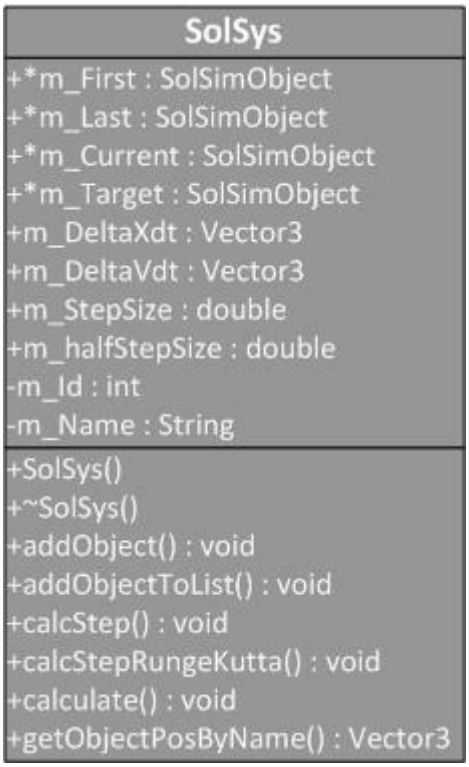
Weiter nutzt die Klasse eine Struktur *Ableitung*, welche mit der Implementierung des klassischen Runge-Kutta-Verfahrens Einzug integriert wurde und als Array *abl* verwendet wird. Näheres hierzu bei der

Beschreibung des Runge-Kutta-Verfahrens.

Um während der Bahnberechnung mittels Schleifenstrukturen einfach über alle Objekte verfügen zu können, werden diese in der Klasse SolSys in einer Liste gespeichert, weshalb hier bereits die Zeiger *next* und *prev* für das jeweils vorherige und nachfolgende Objekt in der Liste bereit gestellt werden.

3.3 Klasse SolSys

3.3.1 UML-Diagramm und Beschreibung



Hauptaufgabe dieser Klasse ist es, ein Sternensystem in Form einer Liste von Objekten zu verwalten und deren Bahnen zu berechnen. Um diese Liste zu erstellen, wird mittels der Methode *addObject()* ein neues Objekt vom Typ SolSimObject erzeugt, dessen Werte mit den Funktionsparametern versehen werden. Abschließend wird dieses neue Objekt über *addObjectToList()* zur Liste hinzugefügt. Anhand des Attributs *m_Typ* wird sichergestellt, dass derzeit nur Sterne, Planeten und deren Monde aufgenommen werden. Sollten später weitere Objekttypen wie etwa Asteroiden oder Raumschiffe und ähnliche Objekte mit im Verhältnis geringen Massen, müsste die Methode zur Bahnberechnung entsprechend erweitert werden, da diese massemäßig kleinen Objekte zwar der Gravitation der Planeten und Sterne unterliegen, jedoch selber einen derart geringen Einfluss auf eben diese hätten, dass an dieser Stelle mit der aktuellen Methode lediglich eine höhere Rechenzeit entstünde.

3.3.2 Bahnberechnung

Auch wenn SolSim nicht den Anspruch erhebt, eine wissenschaftlich exakte Simulation zu sein, sollte doch ein Mindestmaß an Realismus vorhanden sein. Zur möglichst genauen Berechnung der Positionen und Geschwindigkeiten, erhalten die einzelnen Objekte des Sonnensystems ihre Ausgangswerte aus den Ephemeriden, welche die NASA durch ein Web-Interface des vom Jet Propulsion Labortory betriebenen HORIZON-Projekts[2] zur Verfügung stellt. Als Startzeitpunkt werden aktuell Werte vom 05.04.2013 00:00 Uhr verwendet. Das Ziel war es, möglichst geringe Abweichungen zu den Werten an einem beliebigen Folgezeitpunkt zu erreichen. Daher wurden die Berechnungen im frühen Stadium dieser Studienarbeit in einer separat entwickelten Konsolenanwendung durchgeführt und die damit gewonnenen Methoden auf die Simulationsumgebung übertragen.

[2] NASA JPL HORIZONS-Web-Interface (05.06.2013): http://ssd.jpl.nasa.gov/horizons.cgi

Um in der grafischen Darstellung eine durchgängige Animation der Himmelskörper erzeugen zu können, sollten die Berechnungen wenigstens 30-mal pro Sekunde, also mit 30 fps, durchführbar sein. Aufgrund dessen muss ein möglichst effizientes Verfahren gefunden werden, welches dieser Anforderung bei einer simulierten Zeit von einem Tag pro real vergangene Sekunde nachkommt. In Anbetracht astronomischer Maßstäbe ist dies sicherlich eine sehr geringe Schrittweite, daher wäre es wünschenswert, wenn auch höhere Schrittweiten (und somit höhere Simulationsgeschwindigkeiten) mit möglichst geringem Fehlerpotential realisierbar wären. Ideal wäre natürlich ein Berechnungsdurchlauf pro simulierter Sekunde, allerdings bringt dies einen enormen Rechenaufwand mit sich: Nach Newtons Gravitationsgesetz wirkt die Gravitation jedes Körpers auf jeden anderen Körper. Das bedeutet, dass ein Aufwand von N^2 besteht, da jedes Objekt auf jedes andere wirkt und somit die Berechnung des Positions- und Geschwindigkeitsvektors innerhalb zweier Schleifen stattfände. Nun mag das für 8 bzw. 9 Planeten (ich zähle Pluto hier aus Nostalgiegründen gerne noch mit) mit dann 72 Berechnungen (81-9, also abzüglich der Wirkung der Objekte auf sich selbst) noch relativ überschaubar, aber sobald man noch alle Monde, Zwergplaneten und Planetoiden, möglicherweise auch noch ein paar tausend Weltraumschrottelemente auf ihrer Bahn um die Erde simulieren möchte, steigt die Zahl der Berechnungen drastisch an. Im Zeitalter von Vier- und Mehrkernprozessoren mag das für sich genommen immer noch akzeptabel sein, aber eine schöne Simulation kann schnell noch weitere Berechnungen beinhalten, beispielsweise Physikelemente wie eine Kollisionserkennung.

Führt man hingegen die Berechnung in zu großen Zeitintervallen durch, also mit höherer Schrittweite, sinkt zwar der Gesamtrechenbedarf pro reale Sekunde, jedoch mit fortlaufender Simulation auch die Genauigkeit der ermittelten Werte.

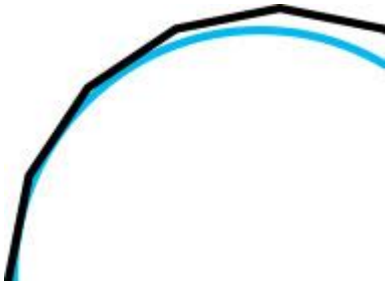


Abb. 7: eine zu hohe Schrittweite (schwarz) kann schnell zu Abweichungen von der realen Bahn (blau) führen

Ein simulierter Tag pro Sekunde bedeutet nun eine Schrittweite von 2880 Sekunden, welche dann in 30 Berechnungen pro Sekunde einfließt. Die Position und Geschwindigkeit aller Objekte wird also alle 2880 Sekunden neu berechnet. Da z.B. die Erde eine Bahngeschwindigkeit von 29,78 Kilometer pro Sekunde besitzt, hat sie in dieser Zeit bereits einen Weg von 85.766,4 Kilometer zurückgelegt. Somit liegt bereits hier ein gewisses Fehlerpotential vor, welches es durch ein geeignetes Verfahren zu minimieren gilt.

Ein simpler Euler'scher Ansatz würde für den aktuellen Positionsvektor eines jeden Objekts jeweils einen Vektor $\Delta\overrightarrow{Pos}$ bestimmen und das Produkt aus diesem mit der Schrittweite auf den Positionsvektor addieren, analog für den Geschwindigkeitsvektor. Wieder gilt: je größer die Schrittweite, desto höher die zu erwartende Abweichung. Um dem vorzubeugen, nutzt diese Simulation das klassische Runge-Kutta-Verfahren.

Als Runge-Kutta-Verfahren[3] bezeichnet man nach Carl Runge und Martin Wilhelm Kutta benannte Einschrittverfahren zur näherungsweisen Lösung von Anfangswertproblemen in der numerischen Mathematik. Kann für ein solches Anfangswertproblem keine exakte Lösung bzw. diese nicht effizient ermittelt werden, dienen solche Verfahren durch die Verwendung von Zwischenschritten als geschickte Methode, sich einer exakten Lösung zu nähern. Als klassisches Runge-Kutta-Verfahren versteht man ein explizites vierstufiges Runge-Kutta-Verfahren, welches den Ansatz verfolgt, Ableitungen mithilfe von Differenzquotienten zu approximieren. In für Nicht-Mathematiker verständlichem Deutsch bedeutet dies auf unser Problem bezogen: statt einen Vektor $\Delta\overrightarrow{Pos}$ über die gesamte Schrittweite direkt zu berechnen, wird dieser aus einer Kombination von 4 Zwischenrechnungen ermittelt. In der ersten wird die Berechnung anhand der Ausgangssituation ohne Berücksichtigung der Schrittweite durchgeführt. Die zweite verwendet die hierbei gewonnenen Werte für eine erneute Berechnung, diesmal unter Berücksichtigung der halben Schrittweite. In der dritten Zwischenrechnung werden wiederrum die Ergebnisse der zweiten verwendet, um diese abermals mit der halben Schrittweite zu verrechnen. Die vierte und letzte Rechnung basiert auf den Ergebnissen der dritten, diesmal jedoch unter Einbeziehung der vollen Schrittweite. Abschließend wird das Produkt aus der Kombination der Zwischenrechnungen und der Schrittweite auf den alten Positions- bzw. Geschwindigkeitsvektor addiert.

Sei

$$y' = f(x, y)$$

eine gewöhnliche Differentialgleichung 1. Ordnung mit den Anfangsbedingungen

$$y(x_0) = y_0$$

und sei weiter h die gewünschte Schrittweite. Dann wird der angenäherte Wert von

$$y(x_0 + h) = y(x_1)$$

nach Runge wie folgt berechnet:

$$y_0' = f(x_0, y_0)$$

$$y_A = y_0 + \frac{h}{2} \cdot y_0' \qquad y'_A = f(x_0 + \frac{h}{2}, y_A)$$

$$y_B = y_0 + \frac{h}{2} \cdot y'_A \qquad y'_B = f\left(x_0 + \frac{h}{2}, y_B\right)$$

$$y_C = y_0 + h \cdot y'_B \qquad y'_C = f(x_0 + h, y_C)$$

$$y = y_0 + h \cdot \frac{1}{6}(y'_0 + 2(y'_A + y'_B) + y'_C)$$

Auf diese Art werden zwar nun 4 statt bisher eine Rechnung benötigt, allerdings lässt sich die Schrittweite im Vergleich zum Euler'schen Verfahren deutlich erhöhen, ohne dabei an Genauigkeit einzubüßen, wie die nachfolgende Grafik schematisch darstellt.

[3] Vgl. Wikipedia: http://de.wikipedia.org/wiki/Runge-Kutta-Verfahren

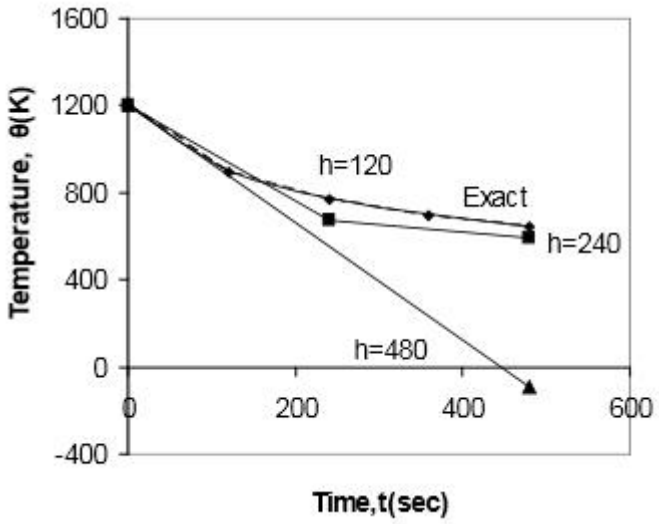


Abb. 8: ein RKV 4ter Ordnung liegt trotz hoher Schrittweite deutlich genauer an den exakten Werten, als beispielsweise ein Euler-Verfahren[4]

[4] Quelle: http://mathforcollege.com/nm/mws/gen/08ode/mws_gen_ode_txt_runge4th.pdf

3.4 Klasse SolSim

3.4.1 UML-Diagramm

3.4.2 Erläuterung

Die Hauptklasse dieser Simulation beinhaltet die Anbindung an die Graphik-Engine, welche die Bereiche Scene Management, Ressource Management und das Rendering abdeckt. Ferner wird durch das Plug-In CEGUI eine grafische Benutzeroberfläche mitsamt Behandlung der Benutzereingaben mittels Maus und Tastatur bereitgestellt.

Die eigentliche Simulation startet mit dem Aufruf der Methode *go()*. Innerhalb dieser Methode erfolgt die Initialisierung globaler Variablen, wie etwa *mDistanceFactor*, welche benötigt wird, um die sich oft auf mehrere Millionen Kilometer belaufenden Distanzen zwischen den einzelnen Objekten der Simulation auf ein sinnvoll darstellbares Maß zu reduzieren. Analog dazu *mScaleEarth* zur Skalierung der Geometrie der Erde und der weiteren Objekte im entsprechenden Verhältnis zur Erde.

Kern der Ogre3D-Engine bildet das Root-Element, welches mit der Übergabe der Plugin-Konfiguration (plugins.cfg) die Engine startet.

Im nächsten Schritt werden die verwendeten Ressourcen mittels der Methode *setupRecources()* bereitgestellt. Dabei wird eine Konfigurationsdatei geladen und ihre Informationen an den ResourceGroupManager weitergereicht. Die Ressourcen selber werden dabei nicht geladen, können nun aber innerhalb der Simulation verwendet werden.

Nun wird das eigentliche RenderingSystem bestimmt und konfiguriert, welches nachher unsere Szene darstellen soll. Ogre3D bietet hier die Wahl zwischen DirectX und OpenGL nebst einigen Parametern, wie etwa der Wahl zwischen Vollbild- und Fenstermodus, Auflösung, Antialiasing und weitere, welche sich wiederum per Konfigurationsdatei festlegen lassen. Alternativ kann vor jedem Programmstart ein Auswahldialog geladen werden, mittels welchem die gleichen Einstellungen vorgenommen werden können.

Danach wird das Objekt *mWindow* erzeugt, welches das Ausgabefenster der Simulation beherbergt. Bevor die Umgebung nun dargestellt wird, besteht mit der Methode *loadResources()* die Möglichkeit, weitere Texturen, Geometrieobjekte, Schriftarten usw. für die Verwendung innerhalb der Simulation bereit zu stellen. Durch die Trennung von der Methode *setupResources()* wird die Übersicht über eigene und Standardressourcen gewahrt.

Zur Verwaltung der eigentlichen Szene und ihrer Objekte, Lichter, Kameras usw. wird ein *SceneManager* erzeugt. Ogre3D bietet hier neben dem Standardmanager auch einen speziellen Terrainmanager, welcher in der aktuellen Version der Simulation jedoch nicht benötigt wird.

Die Methode *createScene()* generiert die eigentliche Szene. Eingangs werden die Elemente der Benutzeroberfläche erzeugt. Licht soll in SolSim nur durch Sterne entstehen, also wird das Umgebungslicht auf tiefes Schwarz gesetzt, dazu das Schattierungsmodel auf additive Stancil-Schatten eingestellt.

Für die Darstellung einer real wirkenden Sternenumgebung bestünde zwar die Möglichkeit, einzelne, mit Lichtern versehene Objekte innerhalb der Szene zu verteilen, eine SkyBox mit Sternentextur vereinfacht dies jedoch erheblich und soll vorerst ausreichen. Optional kann diese aber per Tastendruck deaktiviert werden.

Abb. 9: Mars und seine beiden Monde mit deaktivierter und aktivierter SkyBox

Schließlich werden die Sonne, die Planeten und deren Monde als Geometrieobjekte erzeugt, mit Texturen versehen und an jeweils einen eigenen Knoten des Szenegraphen gebunden. Über diesen lassen sich die einzelnen Elemente relativ

unkompliziert verwalten. Auch können so beispielsweise kleine Monde, für welche keine explizite Positionsberechnung durchgeführt wird, einfach an ein Elternelement angefügt werden. Wird die Position des Elternelements nun verändert, ändert sich auch die der angegliederten Kinder relativ dazu.

Im gleichen Schritt wird jeder später zu animierender Himmelskörper als *SolSimObject* einem Objekt *actSolSys* der bereits beschriebenen Klasse *SolSys* zugewiesen, jeweils ausgerüstet mit dem jeweiligen Gravitationsparameter, Name sowie Ausgangsposition und –Geschwindigkeit. Die Position wird ferner an den jeweiligen Knoten im Szenegraph übertragen. Da nicht für jeden Planeten ein eigenes Geometrieobjekt mit den entsprechenden Maßen erstellt wurde, wird abgesehen von der Erde abschließend jedes Objekt im Verhältnis zur Erde skaliert. Dies geschieht unter Berücksichtigung des bereits genannten Skalierungsfaktors *mScaleEarth*, um möglichst einfach alle Objekte der Simulation später über eben jeden Faktor größenmäßig anpassen zu können. Dies ist insbesondere dann notwendig, wenn über den Distanzfaktor die Entfernungen der Objekte zu einander verringert würden, da hier schnell die Sonnennahen Planeten Merkur und Venus in einer nun zu großen Sonne verschwänden.

Ist die Szene erzeugt, folgt die Positionierung der Kameras. Aktuell wird nur eine Hauptkamera verwendet, welche gleichzeitig die Sicht des Benutzers bildet. Weiter wird ein ViewPort erstellt, welcher sich über den gesamten Bildschirm erstreckt. Denkbar wären auch weitere ViewPorts & Kameras, etwa um eine Mini-Karte des Systems am Bildschirmrand zur besseren Orientierung wiederzugeben.

Da der Benutzer die Kamera frei mit der Kamera navigieren bzw. über die Buttons der Benutzeroberfläche zwischen den Planeten hin und her springen können soll, müssen diese Eingaben abgefangen und verarbeitet werden. In der Methode *createFrameListener()* wird mittels der OIS-Bibliothek, welche ein Bestandteil des Ogre3D-SDK ist, als Brückenglied zwischen der Engine und CEGUI genau dies bereitgestellt. Die Eingabeobjekte *mKeyboard* und *mMouse* werden initialisiert, um jegliche Eingabe über Tastatur und Maus zu erkennen und zu verarbeiten. Dabei stehen für die Tastatureingaben zwei, für die Maussteuerung 3 wichtige Funktionen vom Typ Boolean bereit:

- *keyPressed()*

  Wie der Name bereits vermuten lässt, wird diese Methode aufgerufen, sobald eine Taste auf der Tastatur gedrückt wird. Sie beherbergt unter anderem die Navigation der Kamera über die Tasten „a“, „s“, „d“, „w“. Dabei wird ein Translationsvektor der Bewegungsrichtung entsprechend modifiziert. Dieser Vektor wird im MainLoop, der Methode *frameRenderingQueued*(), abgefragt. Weiter kann z.B. mittels der Taste „z“ die SkyBox de-/aktiviert werden, oder über R der Polygonmodus zwischen solider Darstellung und Gitternetzansicht gewechselt werden.

- *keyReleased()*

  Für manche Eingaben ist es notwendig zu wissen, wann diese enden, beispielsweise bei der Kamerasteuerung: wird eine Taste los gelassen, soll schließlich auch die Kamera in ihrer aktuellen Position verweilen. Dazu wird wieder jede der vier bereits in keyPressed berücksichtigten Tasten verarbeitet, nur wird diesmal dem Translationsvektor für die jeweilige Koordinate der Wert 0 zugewiesen.

- *mouseMoved()*

  Natürlich ist eine Steuerung der Kamera auch per Maus möglich. In SolSim steuert der Benutzer jedoch nicht die Position, sondern die Blickrichtung. Problematisch wird dies jedoch, sobald der Benutzer beispielsweise ein Objekt im All oder ein Element der Benutzeroberfläche anklicken möchte. In diesem Fall ist eine Bewegung der Blickrichtung eher störend oder ungewünscht. Eine mögliche Lösung bietet die Verwendung einer Bool'schen Variable, welche nur im Zustand „true" eine Änderung der Blickrichtung erlaubt. Die Zustandsänderung erfolgt in SolSim über die beiden folgenden Methoden.

- *mousePressed*()

  Solange der Benutzer die rechte Maustaste gedrückt hält, erhält die oben bereits angedeutete Variable *mMouseMove* den Wert „true", wodurch die Änderung der Blickrichtung der Kamera ermöglicht wird.

- *mouseReleased()*

  Analog hierzu ändert sich der Zustand in „false" beim Loslassen der rechten Maustaste: die Blickrichtung friert ein und der Nutzer kann beispielsweise Buttons des GUI drücken, ohne den aktuellen Bildausschnitt ungewollt zu verändern.

Als für die Simulation sehr wichtige Methode sorgt *frameRenderQueued()* dafür, dass die Szene neu gezeichnet und Änderungen überhaupt graphisch sichtbar werden. Noch so viele berechnete Bewegungen bringen schließlich nichts, wenn sie nicht im Bild dargestellt werden.

Auch die beiden definierten Eingabeobjekte *mKeyboard* und *mMouse* prüfen an dieser Stelle mittels einer eigenen Methode *capture()* jeweils einmal pro Frame auf mögliche Eingaben.

Die bereits beschriebene Änderung der Kameraposition wird hier ebenfalls vorgenommen, sofern der Translationsvektor kein Nullvektor ist. Dieser Fall liegt nur so lange vor, wie die Bewegungstasten auch gedrückt werden. Dadurch, dass die Bewegung erst innerhalb dieser Funktion verarbeitet wird, erreicht man eben diese andauernde Bewegung. Würde die Translation bereits in der Methode *keyPressed()* durchgeführt, fände nur eine einmalige Veränderung der Position um den dort definierten Wert statt. Für weitere Modifikationen müsste die entsprechende Taste losgelassen und erneut gedrückt werden.

Zur Simulation der Himmelskörperbewegungen wird an dieser Stelle auch die Methode *moveObjects()* aufgerufen. Da die Bewegungen in Abhängigkeit der eingestellten Simulationswerte auf jedem PC gleich schnell ablaufen sollen, muss diese Methode vom eigentlichen Bildaufbau entkoppelt werden. Dies geschieht mit der einfachen Abfrage der vergangenen Zeit seit dem letzten Bildaufbau. Überschreitet diese 33 Millisekunden, entspricht das dem gewünschten Wert von 30 Bildern pro Sekunde und *moveObjects()* wird aufgerufen. Die Zeit wird zurückgesetzt und wieder so lange aufsummiert, bis der Fall erneut eintritt.

Neben den mathematisch berechneten Positionsänderungen der Himmelskörper, werden in der Methode *moveObjects()* auch weitere Bewegungen wie etwa die Eigenrotation der Sonne und der Erde verarbeitet.

3.5 Benutzeroberfläche

Die Benutzeroberfläche bietet in ihrer aktuellen Form Nutzern die Möglichkeit, per Knopfdruck die Kamera mit einem der 8 Planeten + Pluto zu verknüpfen. Dazu wird an jeden der Buttons eine click-Methode gebunden, beispielsweise *clickMerkur()* für den Button, welcher die Kamera zum Merkur bewegen soll. Diese Methoden rufen wiederum die Methode *bindCamToObject()* auf und übergeben dabei als Funktionsparameter den jeweiligen Knoten innerhalb des Szenegraphen. Die Kamera wird daraufhin von ihrem aktuellen Elternknoten entkoppelt und stattdessen an den übergebenen Zielknoten gehangen.

Eine weitere wichtige Funktion nimmt der Beenden-Button ein: per Klick on selbigen wird die Methode *quit()* aufgerufen, welche das Ende der Simulation einleitet. Dadurch wird ferner die Methode *windowClosed()* aufgerufen, welche die Eingabeobjekte entlädt und den Eingabemanager herunterfährt.

Abb. 10: Das aktuell verwendete GUI - spartanisch, aber zweckmäßig.

Das verwendete Graphikschema ist ein Standardschema der CEGUI-Bibliothek. Mittels eines Layout-Editors können jedoch auch Elemente in individuell erstellten Layouts generiert werden.

4. Geometrie-Objekte und Texturen

Zwar bringt Ogre3D bereits einige geometrische Standardobjekte wie etwa Kugeln oder Quader mit, für die Darstellung der Planeten und eine unkomplizierte Auswahl der zu verwendenden Texturen bieten sich entsprechende Werkzeuge etwa das hier verwendete 3ds Max an. Spätestens bei den Asteroiden mussten eigene Meshes erstellt werden, um sich nicht nur auf kugelförmige Objekte zu beschränken.

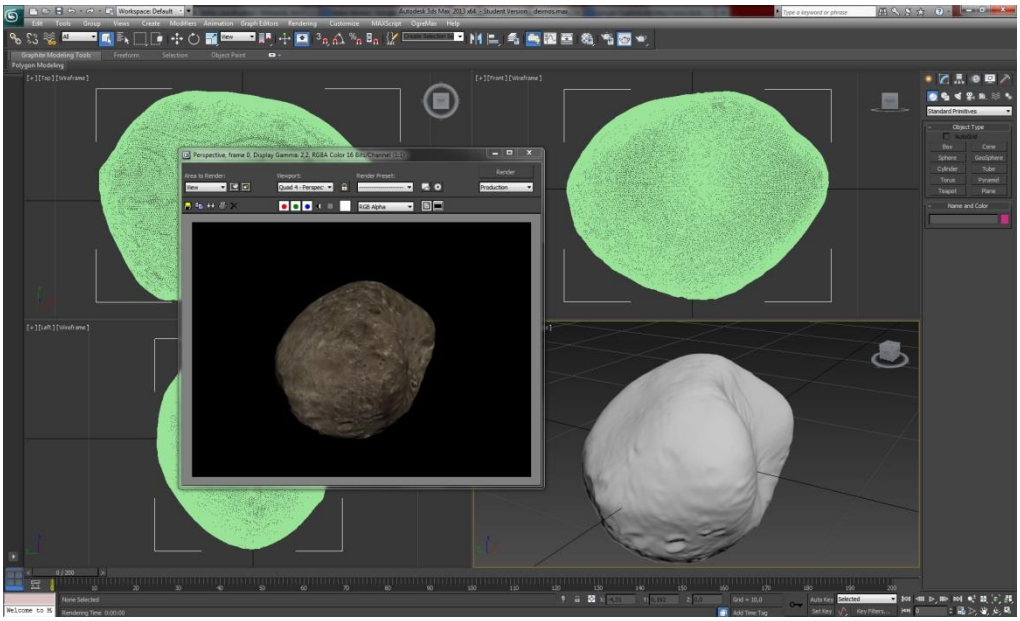


Abb. 11: Deimos als Drahtgittermodell und gerendert in 3ds Max 2013

Texturen für die SkyBox und die einzelnen Himmelskörper wurden entweder in Adobe Photoshop CS 6 erzeugt, oder im Rahmen der Lizensierung bei nichtkommerziellem Gebrauch dem reichhaltigen Angebot im Internet verfügbarer Repositorys[5] entnommen und in 3ds Max verarbeitet.

Die so erzeugten Objekte und Informationen über die verwendeten Texturen können per Plugin direkt in die von Ogre3D gefordertem .mesh und .material-Dateien exportiert und eingebunden werden. So lässt sich etwa einer der beiden Marsmonde, Deimos, relativ leicht in der Simulation darstellen:

[5] Repository mit zahlreichen Texturen zu Objekten unseres Sonnensystems (05.06.2013): http://www.celestiamotherlode.net

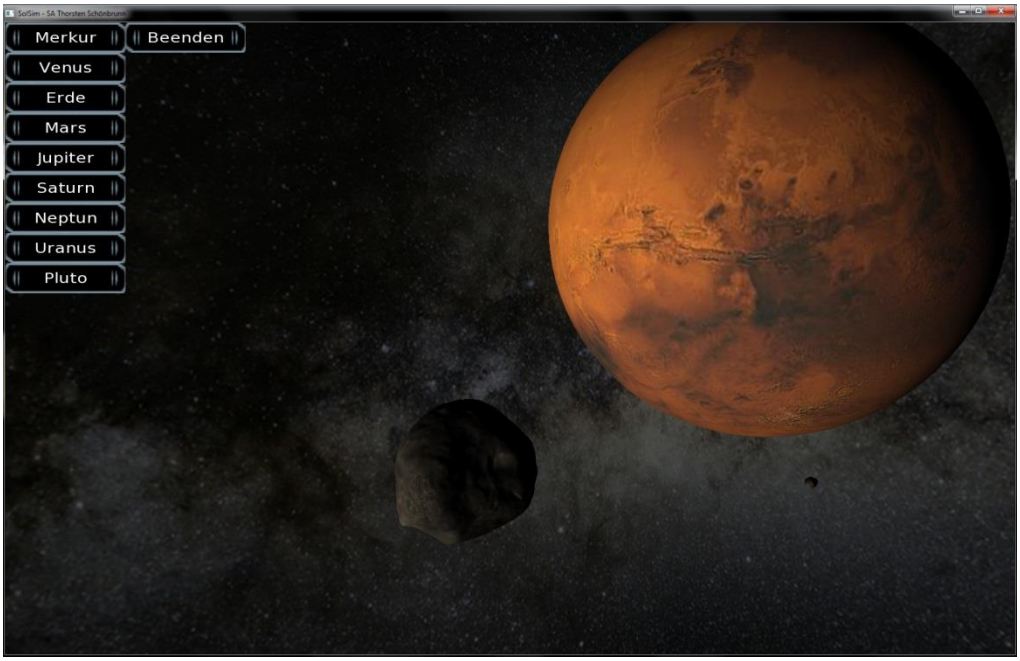


Abb. 12: Deimos, diesmal als Objekt in der Simulation. Im Hintergrund Mars und darunter Phobos, der 2. Marsmond.

Eine .material-Datei ist in der Regel ähnlich dem folgenden Beispiel aufgebaut:

```
material Deimos
{
        technique
        {
                pass
                {
                        ambient 0.588235 0.588235 0.588235 1
                        diffuse 0.588235 0.588235 0.588235 1
                        specular 0 0 0 1 10

                        texture_unit
                        {
                                texture deimos.jpg
                        }
                }

        }

}
```

Hier wird der Name definiert, durch welchen das Material innerhalb der Engine angesprochen wird. Im Rahmen dieser Simulation werden bisher nur einfache Texturen verwendet, theoretisch können bereits über die Material-Datei Texturen verschiedener Auflösung, etwa für ein LOD-Verfahren, hinterlegt werden. An dieser Stelle werden lediglich Angaben zu den Materialeigenschaften gemacht und schließlich die verwendete Textur-Datei, hier im JPG-Format, benannt. Neben JPG versteht Ogre3D eine Reihe weit verbreiteter Grafik- und Texturformate, darunter PNG und DDS. Für SolSim wurden verschiedene Formate erfolgreich getestet, die Vor- und Nachteile diverser Formate werden im Ogre3D-Handbuch näher beschrieben.

5. Fazit und Ausblick

Die Annäherung der Umlaufbahnen über das verwendete Runge-Kutta-Verfahren liefert für eine Berechnungsdauer Positionswerte, welche nur wenige Kilometer von den durch die NASA im HORIZON Ephemeriden angegebenen Werten abweichen. Möglicherweise würde ein Verfahren, welches relativistische Faktoren berücksichtigt, noch etwas genauer arbeiten, doch war es nie Anspruch dieser Arbeit, eine auf den Meter genaue Simulation zu kreieren.

Die graphische Ausgabe bietet in ihrer aktuellen Form viel Raum für Erweiterungen und Verbesserungen. So sollte das GUI-Layout durch ein eigenes Layout mit angepassten Größenverhältnissen ersetzt werden, anstelle der starren Buttons könnte ein DropDown-Menü treten. Darüber hinaus könnten Positionsdaten in Textform dargestellt werden. Zur finalen Implementierung der Idee, neue Objekte an der Position des Mauszeigers zu erstellen, könnte ebenfalls ein GUI-Element erstellt werden, um etwa den Typ, Größe, Masse und Geschwindigkeit des Objekts festzulegen.

Ferner könnten Effekte wie etwa ein reelles „Strahlen“ der Sonne (Godrays) ebenso wie die Darstellung der Atmosphären implementiert werden.

Über sogenannte RenderToTexture-Funktionalitäten sollten des Weiteren die Namen der Himmelskörper neben diesen angezeigt werden, nicht zuletzt, um die Navigation im Raum erheblich zu erleichtern. Ebenso könnten eine Funktion integriert werden, um Objekte anklickbar zu gestalten und dann z.B. Informationen über das gewählte Objekt darzustellen.

Wäre es stark übertrieben, an prozedural erzeugten Planetenoberflächen zu denken?

Es besteht also noch viel Raum für Erweiterungen, welche allerdings auf meiner Seite zwingend mit weiteren Studien der Materie einhergehen. Aber.. irgendwo hat es ja doch eine Menge Spaß gemacht…

6. Abbildungsverzeichnis

7. Verwendete Literatur

Ogre3D API Reference (26.02.2013)
http://www.ogre3d.org/docs/api/html/

Ogre3D Wiki – Tutorials (26.02.2013)
http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Tutorials

Website der Georg-August-Universität Göttingen, Vorlesung „Einführung in die Astro- und Geophysik“ (04.03.2013)
http://lp.uni-goettingen.de/get/text/6805

Wikipedia zum Thema „Himmelsmechanik“ sowie viele der dort angeführten Referenzen (04.03.2013f)
http://de.wikipedia.org/wiki/Himmelsmechanik

Vorlesungsskript zum Thema „Einschrittverfahren, insbesondere Runge-Kutta-Verfahren“ der Universität Hamburg (28.03.2013)
http://www.math.uni-hamburg.de/home/oberle/skripte/num-gew-dgln/dgl4.pdf

Website der ESA (12.03.2013)
http://sci.esa.int/science-e/www/object/index.cfm?fobjectid=35774&fbodylongid=1700

NASA JPL HORIZONS-Web-Interface (08.03.2013)
http://ssd.jpl.nasa.gov/horizons.cgi

In den Klammern jeweils das Datum des ersten Abrufs.