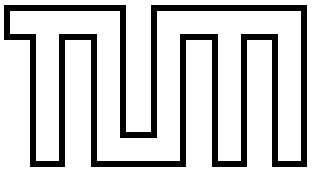


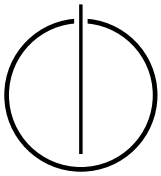
FAKULTÄT FÜR INFORMATIK

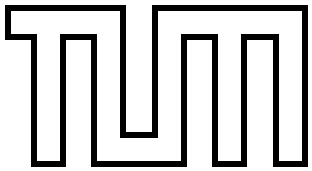
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Efficient Trajectory Modelling for Space Debris Evolution

Oliver Bösing





FAKULTÄT FÜR INFORMATIK

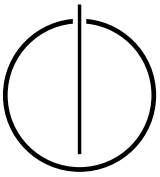
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Efficient Trajectory Modelling for Space Debris Evolution

Effiziente Modellierung zur Vorausberechnung von Weltraumschrott Flugbahnen

Author:	Oliver Bösing
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Fabio Alexander Gratl, M.Sc. and Pablo Gómez, Dr.-Ing.
Date:	15.09.2021



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 15.09.2021 Oliver Bösing

Acknowledgements

I thank Fabio and Pablo for their supervision and guidance during the writing of this thesis.

Special thanks also go to my friends Moe and Freddy for our bad habit of sooner or later starting to talk about mathematics, physics, and computer science until nobody around us has an idea what's wrong with us.

And at last, I thank my parents for being patient with me, even when I changed my degree program for the second time.

Abstract

The number of objects orbiting Earth is increasing and will continue to do so in the future. With the increasing amount of these objects, collisions become more probable. These collisions produce space debris orbiting Earth without an opportunity to manipulate its trajectories.

For this thesis a software solution to model the trajectories of such objects, was developed, using C++. This software solution uses numerical integration, namely the leapfrog integration method, to solve a system of differential equations. These differential equations model the two-body problem of the orbiting object and the Earth and uses Cowell's formulation to add perturbations to the two-body equations. The modeled perturbations are caused by the gravitation of the Sun and the Moon, the aspherical gravitational potential of the Earth, solar radiation pressure, and atmospheric drag.

The software solution is planned to be used as a part of larger space debris simulations systems, developed and used by the European Space Agency's Advanced Concept Team. It is open source and can be found on GitHub[1]

[1] https://github.com/Wombatwarrior/BA$_s pace_d ebris$

Summary

The goal of the following Bachelor's thesis *Efficient Trajectory Modelling for Space Debris Evolution* is developing a software solution in C++, that can be used to simulate the trajectories of a large number of objects orbiting Earth. This software is planned to be used as part of bigger simulation programs for space debris evolution.

In the Theoretical Background chapter of this thesis, first, the equations of motions, that are part of the model used to describe the forces acting on an object near the Earth, are introduced. Next, the used integration method is explained.

The second part of the thesis focuses on the implementation aspects of the developed software. Starting with the used data structures to represent the simulated system. The main focus of the Implementation chapter lies in the calculation of the accelerations, needed to determine the trajectories of the simulated particles. At the end of the chapter the implementation details of the leapfrog integration method and some aspects concerning I/O are explained.

Part three analyzes some experiments conducted. First, the error convergence of the developed software is analyzed. Additionally, the number of particle updates per second is measured for different amounts of simulated particles. To get a better idea of possible ways to increase simulation speed, the consumption of computation time during the simulation is determined. In the last section, the contributions of the different accelerations used in the simulation are compared for different orbit altitudes.

As result of this thesis, the developed software can simulate hundreds of thousands of particles, performing around 2 million particle updates per second. The error of the simulations achieves convergence for reasonable integration time steps. But because the used integration method is only second-order, this convergence is rather slow. To improve the performance of the software a higher-order integration method could be used in the future. Another promising approach could be parallelization because the performed calculations are independent per particle.

Contents

Part I.

Introduction and Background

1. Introduction

Since the beginning of the space age in the 1950s the number of objects in Earth's space environment is constantly increasing. For the year 2020, the amount of launched objects has exploded (Figure 1.1). And this trend is going to continue in the future[ESA21].

The dominant reason for this rapid growth is the reduction of launch costs and the planned installation of satellite mega-constellations like SpaceX's Starlink with around 1700 launched satellites and up to 42000 planned[1], Amazon's Project Kuiper with around 3200 satellites authorized[2] or China's planned communication constellation consisting of almost 13000 satellites[3], to name a few.

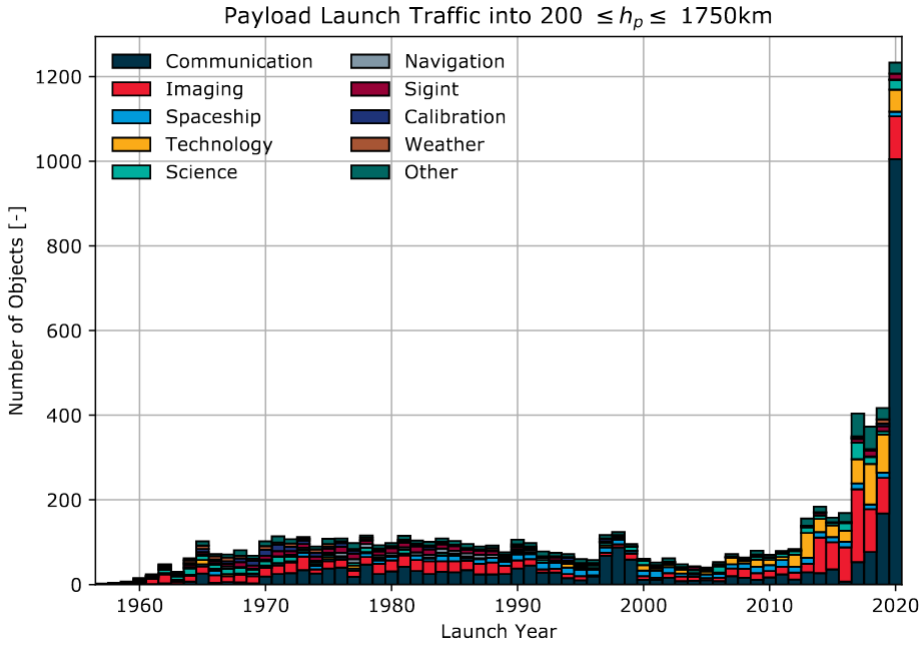


Figure 1.1.: Evolution of the launch traffic to low Earth orbit per mission funding. [ESA21]

With that many objects orbiting Earth, the number of potential collisions will grow exponentially. To address the growing number of space debris objects orbiting Earth and interfering with spaceflight missions, in 2002 the Inter-Agency Space Debris Coordination Committee (IADC) published Space Debris Mitigation Guidelines. These Guidelines are aimed to reduce space debris in protected regions of high importance for spaceflight missions.

[1] https://www.space.com/spacex-30000-more-starlink-satellites.html
[2] https://www.geekwire.com/2020/fcc-says-amazon-can-proceed-kuiper-satellites-will-accommodate-rivals/
[3] https://circleid.com/posts/20210329-guowang-starlink-will-be-chinas-global-broadband-provider/

The Low Earth Orbit with an altitude lower than 2000 km ("Region A" in Figure 1.2) and the Geosynchronous Region with altitudes between 35586 and 35986 km and an angle between +15° and −15° latitude ("Region B" in Figure
1.2).

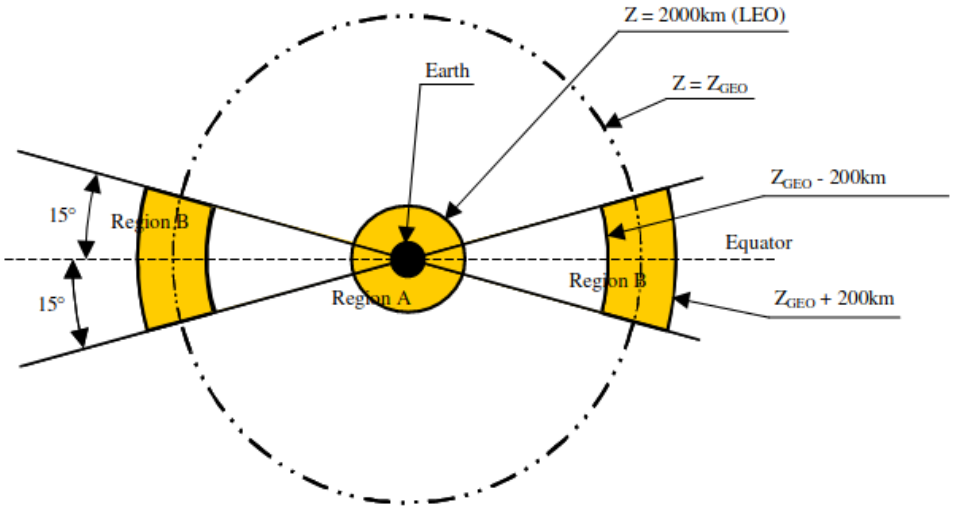


Figure 1.2.: Protected regions defined by the IADC [IAD07]

If collisions happen, this will result in debris objects in the orbit, which will raise the probability of more collisions. This can lead to a cascade of collisions, resulting in a belt of debris surrounding the Earth, the so-called Kessler effect.[KCP78]. This scenario makes it important, to be able to model the evolution of objects around Earth, to predict possible collisions and take adequate measures to avoid them.

The goal of this thesis is to develop a software solution, to model the trajectory of objects orbiting Earth. This is done by numerical integration to solve Newton's equations of motion for a selection of forces acting on such objects. This software is planned to be used in space debris evolution simulations of the European Space Agency's Advanced Concept Team.

2. Theoretical Background

2.1. The J2000 reference frame

As the main source to develop the model used in this thesis the book "Fundamentals of Astrodynamics and Applications" by Vallado and McClain is used[VM97e].

To represent the state of an object orbiting the Earth, an appropriate reference frame is needed. For this thesis, the reference frame is the J2000 reference frame. The J2000 reference frame is an Earth-centered inertial reference frame, i. e. its origin is at the center of the Earth and it is fixed with respect to the stars [VM97c]. Its x-axis points toward the mean vernal equinox, i.e. the intersection point between the Sun and The Earth's equator on the 20th or 21st of March. The y-axis is 90° to the east in the equatorial plane and the z-axis is pointing at the celestial north pole, along the rotational axis of the Earth.

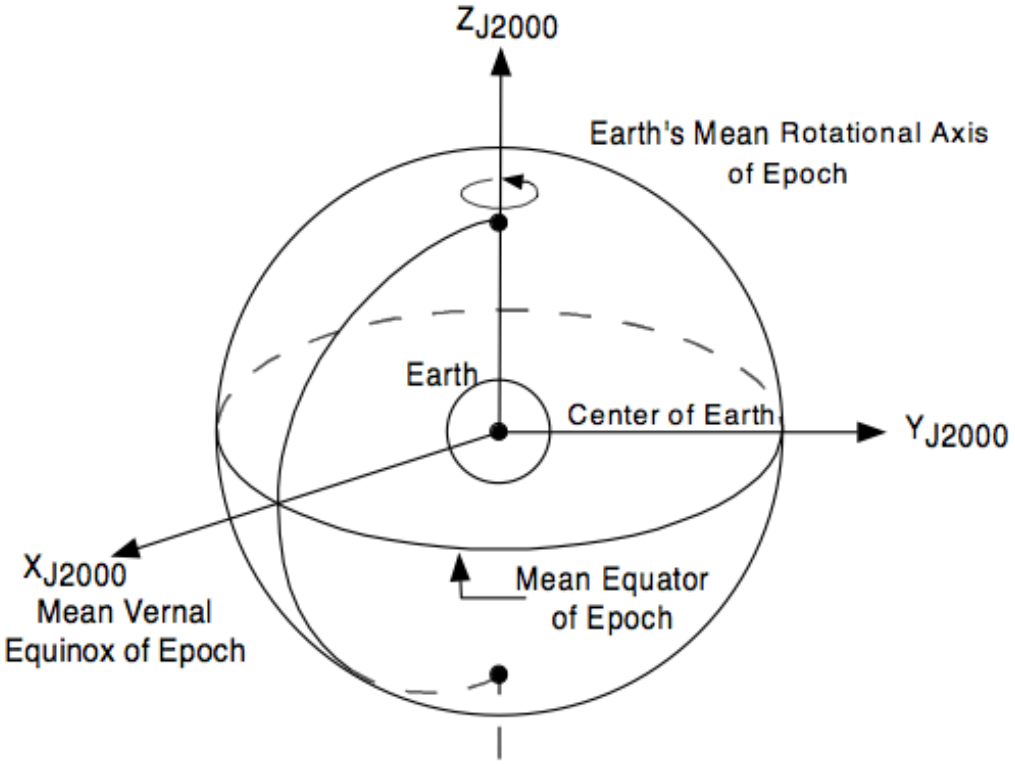


Figure 2.1.: Orientation of the J2000 reference frame axes. [Ros98]

To define an inertial reference frame the axes are defined at an epoch time. For the J2000 reference frame, this epoch is 12:00 Terrestrial Time on 1 January 2000. This is done because

the above definitions change over time, due to periodic changes of the Earth's orbit with respect to the stars i. e. precession and nutation[Ros98].

2.2. Equations Of Motion

The motion of an object can be calculated knowing its position $\vec{p}$ and its position's first and second derivative with respect to time i.e. its velocity $\vec{v}$ and acceleration $\vec{a}$. The model used for this thesis combines several accelerations applied to its objects and uses the resulting accelerations to calculate their positions and velocities.

2.2.1. Two-body equation

For objects orbiting Earth inside the protected regions depicted in Figure 1.2 by far the most important acceleration to take into account is the gravitational acceleration caused by the Earth itself. The basis to calculate this acceleration is Newton's Law of Gravitation. This law describes the attraction between two bodies and can be used to formulate an equation to calculate the acceleration acting on an object orbiting the Earth following an elliptic orbit described by Kepler's Laws.

$$\vec{a}_{Kep} = -\frac{G(m_{\oplus} + m_{obj})}{|\vec{p}|^3}\vec{p} \tag{2.1}$$

Where G is the gravitational constant, $m_{\oplus}$ is the mass of the Earth, and m_{obj} is the mass of the object. Considering that $m_{\oplus}$ is many magnitudes greater than m_{obj} for satellites or debris, the object's mass can be neglected. This results in the equations

$$\vec{a}_{Kep} = -\frac{Gm_{\oplus}}{|\vec{p}|^3}\vec{p} \tag{2.2}$$

Where $Gm_{\oplus} = 3.986004407799724 \times 10^5 km^3 sec^{-2}$ [VM97i].

2.2.2. Third-Body Equation

The last subsection considered only the Earth as an object gravitationally acting on an object in its proximity. But since gravitation is acting on large distances, the acceleration acting on an object, caused by a third body must be taken into account. As described by Vallado [VM97h] a general equation to calculate the gravitational acceleration on an object *obj* in a three-body system with Earth and the third body b is

$$\vec{a}_{\oplus,b} = \vec{a}_{Kep} + Gm_b\left(\frac{\vec{p}_b - \vec{p}_{obj}}{|\vec{p}_b - \vec{p}_{obj}|^3} - \frac{\vec{p}_b - \vec{p}_{\oplus}}{|\vec{p}_b - \vec{p}_{\oplus}|^3}\right) \tag{2.3}$$

The first term is the acceleration acting on the object, due to Earth's gravity, described in the last subsection. To get an equation to calculate the acceleration caused only by the body b the second term is examined further. Since the used J2000 reference frame is Earth-centered the term can be simplified to

$$\vec{a}_b = Gm_b \left(\frac{\vec{p}_b - \vec{p}_{obj}}{|\vec{p}_b - \vec{p}_{obj}|^3} - \frac{\vec{p}_b}{|\vec{p}_b|^3} \right) \tag{2.4}$$

Using $\vec{p}_b - \vec{p}_{obj} = -(\vec{p}_{obj} - \vec{p}_b)$ and $|\vec{p}_b - \vec{p}_{obj}| = |\vec{p}_{obj} - \vec{p}_b|$ [Esc65] the result is

$$\vec{a}_b = -Gm_b \left(\frac{\vec{p}_{obj} - \vec{p}_b}{|\vec{p}_{obj} - \vec{p}_b|^3} + \frac{\vec{p}_b}{|\vec{p}_b|^3} \right) \tag{2.5}$$

The needed variables m_b and $\vec{p}_b$ have to be determined for an arbitrary body to calculate its influence on an object. For relevant bodies m_b is constant and known but a body's position has to be calculated and changes over time. The model used in this thesis takes into account the Sun and the Moon as third bodies and has to determine their position.

Sun as The Third Body

Because the Sun's mass makes up over 99% of the complete mass of the solar system it is dominating the solar system with its gravity. Its effect on an object orbiting Earth is taken into account by the model used in this thesis. To calculate the position of the Sun $\vec{p}_\odot$ at a given point in time t, first its mean anomaly $\ell_\odot$, ecliptic longitude $\lambda_\odot$ and the distance $r_\odot$ between Sun and Earth are calculated.

$$\begin{aligned} \ell_\odot &= \varphi_{\odot,0} + \nu_\odot t \\ \lambda_\odot &= \Omega_\odot + \omega_\odot + \ell_\odot + \left(\tfrac{6892}{3600} \sin \ell_\odot + \tfrac{72}{3600} \sin 2\ell_\odot\right) \\ r_\odot &= 1 - 0.0167024241573 \cos \ell_\odot - 0.0001403565055 \cos 2\ell_\odot \end{aligned} \tag{2.6}$$

Where $\varphi_{\odot,0} = 357.5256°$ is the mean anomaly at $t = 0$, $\nu_\odot = 1.1407410259335311 \times 10^{-5} \frac{°}{s}$ is the mean angular motion, $\Omega_\odot + \omega_\odot = 282.94°$ is the sum of the longitude of ascending node and the argument of periapsis of the Sun orbiting the Earth.

To obtain the vector $\vec{p}_\odot$ between Sun and Earth, the equation

$$\vec{p}_\odot = \begin{pmatrix} r_\odot \cos \lambda_\odot \\ r_\odot \sin \lambda_\odot \cos \varepsilon \\ r_\odot \sin \lambda_\odot \sin \varepsilon \end{pmatrix} AU \tag{2.7}$$

is used. With the obliquity of the ecliptic $\varepsilon = 23.4392911°$.

Now this position can be substituted into the general third body equation 2.5

$$\vec{a}_{Sol} = -Gm_\odot \left(\frac{\vec{p}_{obj} - \vec{p}_\odot}{|\vec{p}_{obj} - \vec{p}_\odot|^3} + \frac{\vec{p}_\odot}{|\vec{p}_\odot|^3} \right) \tag{2.8}$$

Where $Gm_\odot = 1.32712440018 \times 10^{11} km^3 sec^{-2}$.

Moon as The Third Body

Compared to the Sun the Moon has an almost neglectable mass. But compared to the Sun the Moon is significantly closer to the Earth i.e. objects orbiting the Earth. Because of this proximity, the Moon's gravitational perturbation is also incorporated into the model. The calculation of the Moon's position $\vec{p}_\mathcal{M}$ at a given point in time t, requires additional

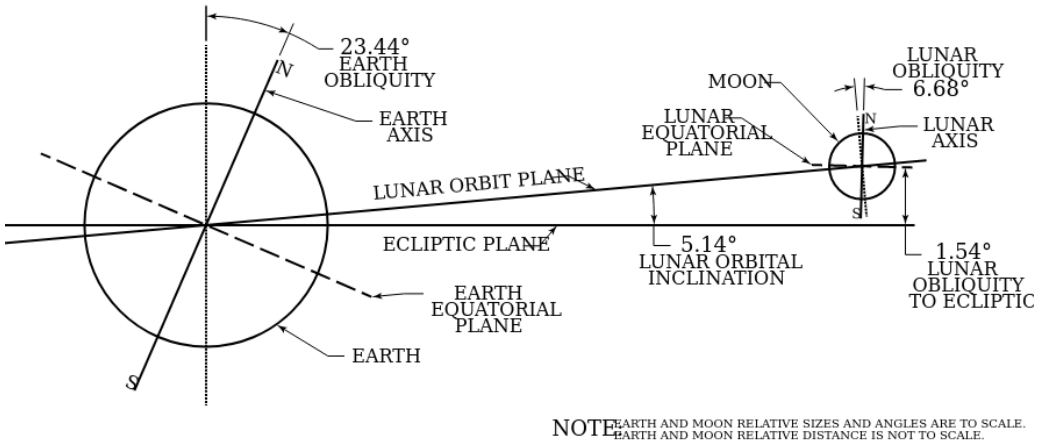


Figure 2.2.: Lunar orbit and orientation with respect to the ecliptic[a]

[a] https://upload.wikimedia.org/wikipedia/commons/4/46/Lunar_orbit_and_orientation_with_respect_to_the_ecliptic.svg

calculations. Reason for this is the complex motion of the Moon. The inclination of the Moon's orbit of about 5° to the ecliptic (Figure 2.2), the third body perturbation, caused by the Sun, and periodic variations of the Moon's orbit around the Earth, must be taken into account.

To do this the four time dependent angles φ_M, φ_{M_a}, $\varphi_{M_p}$and φ_{M_S} are needed.

$$\begin{aligned} \varphi_M &= \nu_\odot t \\ \varphi_{M_a} &= \nu_{M_a} t \\ \varphi_{M_p} &= \nu_{M_p} t \\ \varphi_{M_S} &= \nu_{M_s} t \end{aligned} \tag{2.9}$$

Based on these angles the mean anomaly of the Moon $l_\mathcal{M}$, the mean elongation from the Sun $D_\mathcal{M}$, the mean argument of latitude of the Moon, measured on the ecliptic from the mean equinox of date $F_\mathcal{M}$ and the Moon's longitude L_0 can be calculated with

$$\begin{aligned} l_\mathcal{M} &= \varphi_{M_a} + 134.96292° \\ D_\mathcal{M} &= \varphi_{M_p} + \varphi_{M_a} - \varphi_M + 297.85027° \\ F_\mathcal{M} &= \varphi_{M_p} + \varphi_{M_a} + \varphi_{M_S} + 93.27283° \\ L_0 &= \varphi_{M_p} + \varphi_{M_a} + 218.31617° \end{aligned} \tag{2.10}$$

[VM97g]

Using series expansion to calculate the position of the Moon, the values of its ecliptic latitude $\lambda_\mathcal{M}$, its ecliptic longitude $\beta_\mathcal{M}$ and the magnitude of its position vector $r_\mathcal{M}$ are determined.

$$\begin{aligned}\lambda_{\mathcal{M}} &= L_0 + \tfrac{1}{3600}(22640\sin(l_{\mathcal{M}}) + 769\sin(2l_{\mathcal{M}}) - 4856\sin(l_{\mathcal{M}} - 2D_{\mathcal{M}}) + 2370\sin(2D_{\mathcal{M}}) \\ &\quad -668\sin(\ell_\odot) - 412\sin(2F_{\mathcal{M}}) - 212\sin(2l_{\mathcal{M}} - 2D_{\mathcal{M}}) - 206\sin(l_{\mathcal{M}} + \ell_\odot - 2D_{\mathcal{M}}) \\ &\quad +192\sin(l_{\mathcal{M}} + 2D_{\mathcal{M}}) - 165\sin(\ell_\odot - 2D_{\mathcal{M}}) + 148\sin(l_{\mathcal{M}} - \ell_\odot) \\ &\quad -125\sin(D_{\mathcal{M}}) - 110\sin(l_{\mathcal{M}} + \ell_\odot) - 55\sin(2F_{\mathcal{M}} - 2D_{\mathcal{M}}))\end{aligned} \tag{2.11}$$

$$\begin{aligned}\beta_{\mathcal{M}} &= \tfrac{1}{3600}(18520\sin\left(F_{\mathcal{M}} + \lambda_{\mathcal{M}} - L_0 + \tfrac{1}{3600}(412\sin(2F_{\mathcal{M}}) + 541\sin(\ell_\odot))\right) \\ &\quad -526\sin(F_{\mathcal{M}} - 2D_{\mathcal{M}}) + 44\sin(l_{\mathcal{M}} + F_{\mathcal{M}} - 2D_{\mathcal{M}}) \\ &\quad -31\sin(-l_{\mathcal{M}} + F_{\mathcal{M}} - 2D_{\mathcal{M}}) - 25\sin(-2l_{\mathcal{M}} + F_{\mathcal{M}}) \\ &\quad -23\sin(\ell_\odot + F_{\mathcal{M}} - 2D_{\mathcal{M}}) + 21\sin(-l_{\mathcal{M}} + F_{\mathcal{M}}) \\ &\quad +11\sin(-\ell_\odot + F_{\mathcal{M}} - 2D_{\mathcal{M}}))\end{aligned} \tag{2.12}$$

$$\begin{aligned}r_{\mathcal{M}} &= 385000 - 20905\cos(l_{\mathcal{M}}) - 3699\cos(2D_{\mathcal{M}} - l_{\mathcal{M}}) \\ &\quad -2956\cos(2D_{\mathcal{M}}) - 570\cos(2l_{\mathcal{M}}) + 246\cos(2l_{\mathcal{M}} - 2D_{\mathcal{M}}) \\ &\quad -205\cos(\ell_\odot - 2D_{\mathcal{M}}) - 171\cos(l_{\mathcal{M}} + 2D_{\mathcal{M}}) - 152\cos(l_{\mathcal{M}} + \ell_\odot - 2D_{\mathcal{M}})\end{aligned} \tag{2.13}$$

To obtain the position vector $\vec{p}_{\mathcal{M}}$ in the J2000 reference frame the equation

$$\vec{p}_{\mathcal{M}} = \begin{pmatrix}1 & 0 & 0 \\ 0 & \cos(\varepsilon) & -\sin(\varepsilon) \\ 0 & \sin(\varepsilon) & \cos(\varepsilon)\end{pmatrix} \cdot \begin{pmatrix}r_{\mathcal{M}}\cos(\lambda_{\mathcal{M}})\cos(\beta_{\mathcal{M}}) \\ r_{\mathcal{M}}\sin(\lambda_{\mathcal{M}})\cos(\beta_{\mathcal{M}}) \\ r_{\mathcal{M}}\sin(\beta_{\mathcal{M}})\end{pmatrix} \tag{2.14}$$

is used [VM97a].

Now this position can be substituted into the general third body equation 2.5

$$\vec{a}_{Lun} = -Gm_{\mathcal{M}}\left(\frac{\vec{p}_{obj} - \vec{p}_{\mathcal{M}}}{|\vec{p}_{obj} - \vec{p}_{\mathcal{M}}|^3} + \frac{\vec{p}_{\mathcal{M}}}{|\vec{p}_{\mathcal{M}}|^3}\right) \tag{2.15}$$

Where $Gm_{\mathcal{M}} = 4.9028 \times 10^3 km^3 sec^{-2}$.

2.2.3. Spherical Harmonics

The Two-Body Equation 2.2 discussed in the subsection 2.2.1 makes assumptions. One of these assumptions about the two bodies in question is that they are both spherically symmetric and their density is uniform. Because of this assumption, the bodies can be treated as point masses [VM97i]. But this assumption is not true for the Earth. Due to its rotation, the shape of the Earth is flattened along its rotational axis. Additionally, its density is not constant. This results in a gravitational potential that is not spherical. Because of this, the used model incorporates perturbing accelerations, due to the Earth's nonspherical nature and mass distribution, using spherical harmonics.

To calculate accelerations caused by this nonspherical potential, first, the potential U itself is defined as

$$U = \frac{Gm_\oplus}{|\vec{p}|}\left[1 + \sum_{l=2}^{\infty}\sum_{m=0}^{l}\left(\frac{R_\oplus}{|\vec{p}|}\right)^l P_{lm}[\sin(\phi_{gc})]\{C_{lm}\cos(m\lambda) + S_{lm}\sin(m\lambda)\}\right] \tag{2.16}$$

Where the magnitude of the position vector $|\vec{p}|$, the geocentric latitude ϕ_{gc} and the geocentric longitude λ_{gc} are the polar coordinates of the location of the object. $P_{lm}[\sin(\phi_{gc})]$ are the associated Legendre functions for the geocentric latitude. The coefficients C_{lm} and S_{lm} are approximated by analyzing observation data of satellites orbiting the Earth [LPL$^+$89]. To calculate the acceleration due to this potential partial derivatives of the potential function are needed. Resulting in the equation for the acceleration

$$\vec{a}_{harmonics} = \frac{\partial U}{\partial |\vec{p}|}\left(\frac{\partial |\vec{p}|}{\partial \vec{p}}\right)^T + \frac{\partial U}{\partial \phi_{gc}}\left(\frac{\partial \phi_{gc}}{\partial \vec{p}}\right)^T + \frac{\partial U}{\partial \lambda_{gc}}\left(\frac{\partial \lambda_{gc}}{\partial \vec{p}}\right)^T \tag{2.17}$$

.

The terms in the equation 2.17 are

$$\begin{aligned}
\frac{\partial U}{\partial |\vec{p}|} &= -\frac{Gm_\oplus}{|\vec{p}|^2}\sum_{l=2}^{\infty}\sum_{m=0}^{l}\left(\frac{R_\oplus}{|\vec{p}|}\right)^l (l+1)P_{lm}[\sin(\phi_{gc})]\{C_{lm}\cos(m\lambda_{gc}) + S_{lm}\sin(m\lambda_{gc})\} \\
\frac{\partial U}{\partial \phi_{gc}} &= \frac{Gm_\oplus}{|\vec{p}|^2}\sum_{l=2}^{\infty}\sum_{m=0}^{l}\left(\frac{R_\oplus}{|\vec{p}|}\right)^l \{P_{l,m+1}[\sin(\phi_{gc})] - m\tan(\phi_{gc})P_{lm}[\sin(\phi_{gc})]\} \\
&\quad \times\{C_{lm}\cos(m\lambda_{gc}) + S_{lm}\sin(m\lambda_{gc})\} \\
\frac{\partial U}{\partial \lambda_{gc}} &= \frac{Gm_\oplus}{|\vec{p}|^2}\sum_{l=2}^{\infty}\sum_{m=0}^{l}\left(\frac{R_\oplus}{|\vec{p}|}\right)^l mP_{lm}[\sin(\phi_{gc})]\{S_{lm}\cos(m\lambda_{gc}) + C_{lm}\sin(m\lambda_{gc})\} \\
\frac{\partial |\vec{p}|}{\partial \vec{p}} &= \frac{\vec{p}^T}{|\vec{p}|} \\
\frac{\partial \phi_{gc}}{\partial \vec{p}} &= \frac{1}{p_x^2+p_y^2}\left(-\frac{\vec{p}^T p_z}{|\vec{p}|^2} + \frac{\partial p_z}{\partial \vec{p}}\right) \\
\frac{\partial \lambda_{gc}}{\partial \vec{p}} &= \frac{1}{p_x^2+p_y^2}\left(p_x\frac{\partial p_y}{\partial \vec{p}} - p_y\frac{\partial p_x}{\partial \vec{p}}\right)
\end{aligned} \tag{2.18}$$

Putting these terms together the resulting equations for the components of the acceleration $\vec{a}_{harmonics}$ are

$$\begin{aligned}
a_{harmonics_x} &= \left(\frac{1}{|\vec{p}|}\frac{\partial U}{\partial |\vec{p}|} - \frac{p_z}{|\vec{p}|^2\sqrt{p_x^2+p_y^2}}\frac{\partial U}{\partial \phi_{gc}}\right)p_x - \left(\frac{1}{p_x^2+p_y^2}\frac{\partial U}{\partial \lambda_{gc}}\right)p_y \\
a_{harmonics_y} &= \left(\frac{1}{|\vec{p}|}\frac{\partial U}{\partial |\vec{p}|} + \frac{p_z}{|\vec{p}|^2\sqrt{p_x^2+p_y^2}}\frac{\partial U}{\partial \phi_{gc}}\right)p_y - \left(\frac{1}{p_x^2+p_y^2}\frac{\partial U}{\partial \lambda_{gc}}\right)p_x \\
a_{harmonics_z} &= \frac{1}{|\vec{p}|}\frac{\partial U}{\partial |\vec{p}|}p_z + \frac{\sqrt{p_x^2+p_y^2}}{|\vec{p}|^2}\frac{\partial U}{\partial}
\end{aligned} \tag{2.19}$$

[VM97f]

Zonal Harmonics

If $m = 0$, the dependency on longitudinal potential vanishes completely. This results in a symmetrical potential w.r.t. the polar axis i.e. bands of latitude (Figure 2.3). The resulting spherical harmonics are called zonal harmonics. Zonal harmonics are named J_l. In the model used in this thesis, the J_2 i.e. $l = 2$ is used, because it is the most significant one and is used to simulate the oblateness of the Earth due to its rotation. Using $l = 2$ and $m = 0$, equation 2.19 can be converted into

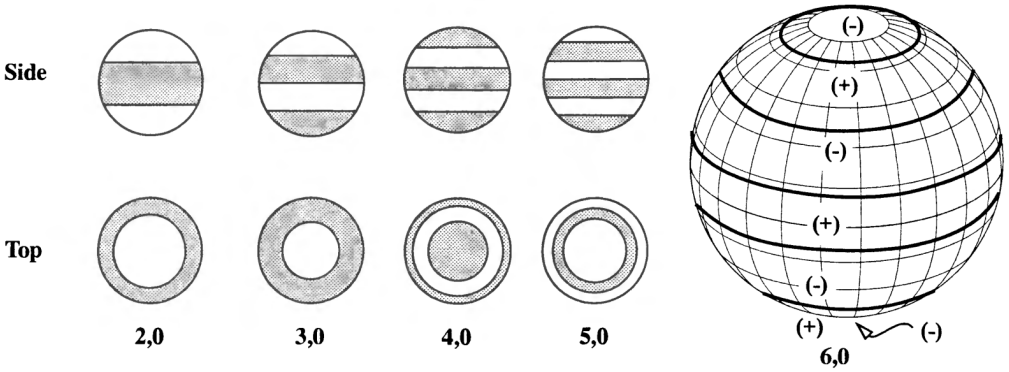


Figure 2.3.: Zonal harmonics for different l. [VM97f]

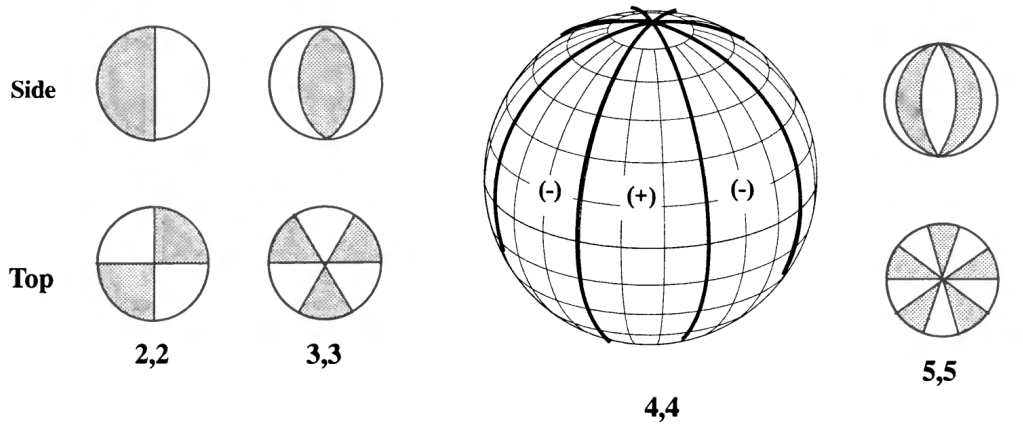


Figure 2.4.: Sectorial harmonics for different $l = m$. [VM97f]

$$\begin{aligned} a_{J2,x} &= \frac{Gm_{\oplus}R_E^2\sqrt{5}C_{20}p_x}{2|\vec{p}_{obj}|}\left(\frac{3}{|\vec{p}_{obj}|^4} - \frac{15p_z^2}{|\vec{p}_{obj}|^6}\right) \\ a_{J2,y} &= \frac{Gm_{\oplus}R_E^2\sqrt{5}C_{20}p_y}{2|\vec{p}_{obj}|}\left(\frac{3}{|\vec{p}_{obj}|^4} - \frac{15p_z^2}{|\vec{p}_{obj}|^6}\right) \\ a_{J2,z} &= \frac{Gm_{\oplus}R_E^2\sqrt{5}C_{20}p_z}{2|\vec{p}_{obj}|}\left(\frac{9}{|\vec{p}_{obj}|^4} - \frac{15p_z^2}{|\vec{p}_{obj}|^6}\right) \end{aligned} \tag{2.20}$$

[VM97b] Where the coefficient $C_{20} = -4.84165371736 \times 10^{-4}$.

Sectorial Harmonics

The other type of spherical harmonics, used in the model, are so-called sectorial harmonics, when $l = m$. Here the potential vanishes along longitudes resulting in an "orange-slice" like division into $2l$ sectors (Figure 2.4). These harmonics are used to model variations in the mass concentration of the Earth. In the model the sectorial harmonics of order 2 i.e. $l = m = 2$ are used.

Because the longitude affects the potential, first, the position vector $\vec{p}_{obj}$ of the object is rotated around the rotational axis of the Earth i.e. the z-axis, to align it with the current

orientation of the Earth. Where the angle $\theta_G + \nu_E t$ is the angle the Earth rotated since $t = 0$.

$$\vec{d} = \begin{pmatrix} \cos(\theta_G + \nu_E t) & \sin(\theta_G + \nu_E t) & 0 \\ -\sin(\theta_G + \nu_E t) & \cos(\theta_G + \nu_E t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \vec{p}_{obj} \tag{2.21}$$

Using $l = m = 2$ in the equation 2.19 the vectors $\vec{c}_{22}$ and $\vec{s}_{22}$ can be calculated.

$$\begin{aligned} c_{22,x} &= \frac{5Gm_\oplus R_\oplus^2 \sqrt{15} C_{22} d_x (d_y^2 - d_x^2)}{2|\vec{d}|^7} + \frac{Gm_\oplus R_\oplus^2 \sqrt{15} C_{22} d_y}{|\vec{d}|^5} \\ c_{22,y} &= \frac{5Gm_\oplus R_\oplus^2 \sqrt{15} C_{22} d_y (d_y^2 - d_x^2)}{2|\vec{d}|^7} - \frac{Gm_\oplus R_\oplus^2 \sqrt{15} C_{22} d_x}{|\vec{d}|^5} \\ c_{22,z} &= \frac{5Gm_\oplus R_\oplus^2 \sqrt{15} C_{22} d_z (d_y^2 - d_x^2)}{2|\vec{d}|^7} \end{aligned} \tag{2.22}$$

With the coefficient $C_{22} = 2.43914352398 \times 10^{-6}$.

$$\begin{aligned} s_{22,x} &= -\frac{5Gm_\oplus R_\oplus^2 \sqrt{15} S_{22} d_x^2 d_y}{|\vec{d}|^7} + \frac{Gm_\oplus R_\oplus^2 \sqrt{15} S_{22} d_x}{|\vec{d}|^5} \\ s_{22,y} &= -\frac{5Gm_\oplus R_\oplus^2 \sqrt{15} S_{22} d_x d_y^2}{|\vec{d}|^7} + \frac{Gm_\oplus R_\oplus^2 \sqrt{15} S_{22} d_y}{|\vec{d}|^5} \\ s_{22,z} &= -\frac{5Gm_\oplus R_\oplus^2 \sqrt{15} S_{22} d_x d_y d_z}{|\vec{d}|^7} \end{aligned} \tag{2.23}$$

Where $S_{22} = -1.40016683654 \times 10^{-6}$.

To rotate the results back to the actual position, the Matrix M is defined as

$$M = \begin{pmatrix} \cos(\theta_G + \nu_E t) & -\sin(\theta_G + \nu_E t) & 0 \\ \sin(\theta_G + \nu_E t) & \cos(\theta_G + \nu_E t) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.24}$$

Now the two vectors can be rotated around the z-axis resulting in the two acceleration vectors

$$\vec{a}_{C22} = M\vec{c}_{22} \tag{2.25}$$

$$\vec{a}_{S22} = M\vec{s}_{22} \tag{2.26}$$

The sum of $\vec{a}_{C22}$ and $\vec{a}_{S22}$ is a solution of equation 2.19 with $l = m = 2$.

2.2.4. Solar Radiation Pressure

After discussing gravitational effects and their resulting accelerations for an object in orbit, there are other sources of acceleration that are integrated into the model. The first one of them is the solar radiation pressure, caused by the impulse of the Sun's radiation that is transferred on an object if the object is hit by it. The acceleration aused by this pressure can be calculated using

$$\vec{a}_{SRP} = -\frac{P_{SRP} c_R A_\odot}{m} \left(\frac{\vec{p} - \vec{p}_\odot}{|\vec{p} - \vec{p}_\odot|} \right) \tag{2.27}$$

. Where $P_{SRP} = 4.56 \times 10^{-6} N/m^2$ is the solar radiation pressure at the distance of one astronomical unit from the Sun. $A_\odot$ is the surface of the object exposed to the Sun's

radiation, c_R is the reflectivity of the object, $\vec{p}-\vec{p}_\odot$ is the connecting vector between the Sun and the object, and m is the mass of the object. The Sun's position $\vec{p}_\odot$ can be calculated using Equation 2.7.

The used model is not using $A_\odot$, c_r and m directly. Instead, it introduces the factor AOM, which combines $A_\odot$ and m for a given object and is given before the calculation and c_r is assumed to be 1. This results in the equation

$$\vec{a}_{SRP} = P_{SRP} AOM \frac{\vec{p}-\vec{p}_\odot}{|\vec{p}-\vec{p}_\odot|} \tag{2.28}$$

This equation assumes the distance between the Sun and the object is exactly one astronomical unit. To get a better approximation of the acceleration the scaling factor $\frac{AU^2}{|\vec{p}-\vec{p}_\odot|^2}$ is introduced. The result is the used equation

$$\vec{a}_{SRP} = AU^2 P_{SRP} AOM \frac{\vec{p}-\vec{p}_\odot}{|\vec{p}-\vec{p}_\odot|^3} \tag{2.29}$$

2.2.5. Atmospheric Drag

The second non-conservative acceleration included in the model is the one caused by the friction between the Earth's atmosphere and an orbiting object.

Because the density of the Earth's atmosphere p (not to be confused with the position vector $\vec{p}$) decreases with the distance from the Earth, near the Earth, the influence of atmospheric drag is extremely strong compared to other perturbing effects, but at higher altitudes effects like third bodies or the solar radiation pressure become more dominant. The drag effect retards the object's motion i.e. the direction of the acceleration is opposite of the object's velocity w.r.t. to the atmosphere $\vec{v}_{rel}$, resulting in a loss of altitude.

The general equation to calculate the acceleration caused by the drag is

$$\vec{a}_{Drag} = -\frac{1}{2}\frac{c_D A}{m} p |\vec{v}_{rel}|^2 \frac{\vec{v}_{rel}}{|\vec{v}_{rel}|} \tag{2.30}$$

Where c_D is the coefficient of drag. This value reflects how strong the influence of the drag effect is on an object. Spheres approximately have a $c_D \sim 2$ to 2.1, a flat plate one of $c_D \sim 2.2$. A is the cross-section area normal to the object's velocity vector. Because to determine A the orientation of the object is needed with a high, almost impossible to know, accuracy, its value is approximated and constant in the used model. Calculating Equation 2.30 requires the values for p and $\vec{v_{rel}}$, that must be determined. To get the velocity relative to the atmosphere, a model of the motion of the atmosphere itself must be chosen.

The chosen model is only using the Earth's rotation and the position of the object, neglecting other factors like wind.

$$\vec{v}_{rel} = \vec{v}_{obj} - \vec{\omega}_\oplus \times \vec{p}_{obj} = \begin{pmatrix} v_{obj_x} - \omega_\oplus p_{obj_y} \\ v_{obj_y} + \omega_\oplus p_{obj_x} \\ v_{obj_z} \end{pmatrix} \tag{2.31}$$

Where

$$\vec{\omega}_\oplus = \begin{pmatrix} 0 \\ 0 \\ \omega_\oplus \end{pmatrix} \tag{2.32}$$

is the rotational vector of the Earth and $\omega_\oplus = 7.292115 \times 10^{-5} \frac{rad}{s}$. Determining the atmospheric density p can be arbitrarily complex, because the number of factors influencing it is high, and many of them are themselves highly complex to model and interact with each other.

For example one can consider [VM07]:

- The temperature of the atmosphere
- The magnetic field of the Earth
- Latitudinal and longitudinal variations of the atmosphere
- The influence of the Sun's activity on the atmosphere
- Winds
- Tides

The chosen model of the atmosphere is very simple. It assumes the density of the atmosphere decreases exponentially with the distance from the Earth's surface.

$$p = p_0 \exp\left(-\frac{|\vec{p}_{obj}| - R_\oplus}{H}\right) \tag{2.33}$$

Where $p_0 = 1.3 kgm^{-3}$ is the atmospheric density at ground level and $H = 8.5km$ is the atmospheric scale factor [Fit12].

2.2.6. Cowell's Formulation

As mentioned above, $\vec{a}_{Kep}$ is by far the main acceleration acting on an object near the Earth. The other described effects cause perturbations on this acceleration. To formulate total equations of motion to be integrated using numerical methods like in this thesis, the perturbing accelerations of n sources can be added linearly to $\vec{a}_{Kep}$ to get the acceleration

$$\vec{a} = \vec{a}_{Kep} + \sum_{i=1}^{n} \vec{a}_{pertubred_i} \tag{2.34}$$

This formulation is known as Cowell's formulation. Advantageous about this formulation is the fact, that perturbations can be added or removed easily, without affecting the calculations of any of the other ones [VM97d]. Substituting all accelerations, described above, into Cowell's formulation the result is

$$\vec{a} = \vec{a}_{Kep} + \vec{a}_{Sol} + \vec{a}_{Lun} + \vec{a}_{J2} + \vec{a}_{C22} + \vec{a}_{S22} + \vec{a}_{SRP} + \vec{a}_{Drag} \tag{2.35}$$

This equation is used in this thesis.

2.3. Integration

To determine the motion of an object, the state of the object i.e. its position $\vec{p}_i$ and its derivatives $\dot{\vec{p}}_i$ and $\ddot{\vec{p}}_i$ a.k.a its velocity $\vec{v}_i$ and acceleration $\vec{a}_i$, is calculated along a grid of points in time $t_i = t_0 + \Delta t * i$, with $i = 1, 2, 3...$, the step size Δt and starting time t_0.

This leads to the problem of determining the state at the next time step $i+1$. To solve this problem numerical integration is used. The used numerical integration method for this thesis is the leapfrog method. This method defines the equations

$$\begin{aligned} \vec{v}_{i+1/2} &= \vec{v}_{i-1/2} + \vec{a}_i \Delta t \\ \vec{p}_{i+1} &= \vec{p}_i + \vec{v}_{i+1/2} \Delta t \end{aligned} \tag{2.36}$$

This update scheme gives the leapfrog method its name, because the velocity and position are updated at different points in time and "leapfrog" over each other (Figure 2.5).

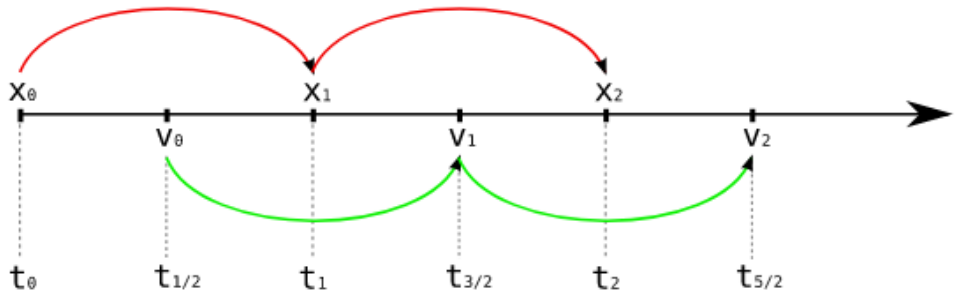


Figure 2.5.: Update scheme of the leapfrog method.[Art]

The equations 2.36 can be reformulated, to determine $\vec{p}$ and $\vec{v}$ both at $t_{i+1} = t_0 + \Delta t * (i+1)$[1] [Art].

$$\begin{aligned} \vec{p}_{i+1} &= \vec{p}_i + \vec{v}_i \Delta t + \tfrac{1}{2} \vec{a}_i \Delta t^2 \\ \vec{v}_{i+1} &= \vec{v}_i + \tfrac{1}{2} (\vec{a}_i + \vec{a}_{i+1}) \Delta t. \end{aligned} \tag{2.37}$$

The method needs only one evaluation of the acceleration per time step. But compared to other integration methods with this property, e.g. Euler's method, the order of leapfrog is 2 instead of 1, leading to higher accuracy. But one of the most important features of the leapfrog method is the conversation of energy for conservative forces, i.e. it is a symplectic method[You13].

[1] http://www.artcompsci.org/vol$_1$/$v1_web$/$node34.html$

3. Related Work

There are lots of software solutions, that are used for orbit propagation. One of the oldest groups of them is the Simplified General Perturbations (SGP) model series. The development of these orbit propagators began in the 1960s and one of them that is used by NASA and other organizations is the Simplified General Perturbations-4 (SGP4) propagator published in 1980[VCHK06].

Because the SGP4 model produces an error of more than 1 km per day, it is not suited to simulate large amounts of time into the future. It is used to track objects orbiting Earth, but due to the error in the predicted trajectories, the state of the objects frequently has to be corrected using observation data.

Another software developed and used by the ESA is the pykep library. The pykep library consists of highly specialized and optimized modules, that are used for mission planning and research. But no module for trajectory propagation of a great number of particles orbiting Earth is available yet[1].

Because the goal of this thesis is to build a software solution, that can efficiently calculate the trajectory of an object orbiting Earth, a state of the art tool is needed to compare the developed solution's performance. For this purpose, the *heyoka* tool is used. This is a C++ library, developed to integrate systems of ordinary differential equations, using Taylor's method. Taylor's method, in short, uses Taylor expansion around the point in time $t = t_0$ to determine the solution x_{t_1} at a point in time t_1, using the starting condition x_0.

$$x_{t_1} = x_0 + x'(t_0)h + \frac{1}{2}x''(t_0)h^2 + ... + \frac{x^p(t_0)}{p!}h^p \tag{3.1}$$

Where $h = t_1 - t_0$ is the time step, p is the order of the Taylor method and $x^n(t)$ is the n^{th} derivative of $x(t)$ [BI21].

By increasing p the integration error ε of the method can be reduced. On the other hand, the required calculations grow quadratic with respect to p, resulting in a trade-off between precision and speed. Because of this, *heyoka* can be used to calculate solutions of a given system of ordinary differential equations with ε being smaller than machine precision.

In this thesis *heyoka* is used to calculate solutions for the model derived in Section 2.2 with machine precision. These solutions are used as ground truth in Section 5.2. The software developed for this thesis tries to achieve a better error behavior than propagators like SGP4 for a large number of simulated particles, without being as computational demanding as a high order Taylor integrator like *heyoka*.

[1] https://esa.github.io/pykep/documentation/index.html

Part II.

Implementation

4. Implementation

As a practical part of this thesis, a C++ software solution is developed to use numerical integration, to efficiently model the trajectories of a great number of objects, following Newton's Laws of motion based on the acceleration model explained in Section 2.2.

4.1. Data Structures

To develop software to determine the trajectory of a particle using the model described in Section 2.2, first, a data structure to store the state of a particle is needed.

Since the leapfrog method described in Section 2.3 is used for integration, this data structure has to store the values used in Equation 2.37. Other required properties of the objects are the factors needed to calculate the solar radiation pressure using Equation 2.29 and the atmospheric drag using Equation 2.30. The first one uses the factor AOM that represents the ratio between the object's area affected by the solar radiation and the object's mass. The second one contains the factor $\frac{c_D A}{m}$, i.e. the inverse of the so-called ballistic component $\frac{m}{c_D}$ times the cross-section area A of the object. Both of these values are assumed constant in the used model and are saved in the object's data structure.

The used data object is the `Debris` class with the following data members:

- The 3D `position` vector of the object of the current time step in km
- The 3D `velocity` vector of the object of the current time step in $\frac{km}{s}$
- The 3D acceleration vector of the last time step `acc_t0` in $\frac{km}{s^2}$
- The 3D acceleration vector of the current time step `acc_t1` $\frac{km}{s^2}$
- The object's area over mass factor `aom` in $\frac{km^2}{kg}$
- The object's inverse ballistic component times the object's cross section area `bc_inv` in kgm^2

Because the developed software is designed to simulate multiple objects in parallel, the `DebrisContainer` class is introduced. The purpose of a `DebrisContainer` object is to hold a vector object of `Debris` objects and managing access to these `Debris` objects.

4.2. Acceleration Calculation

The developed software uses a pattern of linearly adding up accelerations due to different perturbation forces. This calculation scheme is an implementation of Cowell's formulation described in Section 2.2.6.

The `AccelerationAccumulator` class is responsible for managing the calculations of these accelerations. To achieve this an `AccelerationAccumulator` object holds the current time

of the simulation and a reference to a central `DebrisContainer` object, holding the `Debris` objects to simulate. Objects of this class can be configured to toggle the calculation of the different acceleration sources. This is done using the `confi` array of boolean values indexed by an enumeration of the `AccelerationComponents`.

The eight `AccelerationComponents` and their corresponding accelerations are:

- `KepComponent` : a_{Kep} Equation 2.2
- `J2Component` : a_{J2} Equation 2.20
- `C22Component` : a_{C22} Equation 2.25
- `S22Component` : a_{S22} Equation 2.26
- `LunComponent` : a_{Lun} Equation 2.15
- `SolComponent` : a_{Sol} Equation 2.8
- `SRPComponent` : a_{SRP} Equation 2.29
- `DragComponent` : a_{Drag} Equation 2.30

The calculation of the accelerations per time step can be divided into two parts:

1. Determine the object independent state of the simulated system.
2. Calculate the activated acceleration components for all particles. Per particle this can be divided into two sub steps by itself:
   a) Calculate values needed for more than one component.
   b) Calculate the actual component accelerations.

During step 1. some values needed for the calculation of the accelerations can be determined and reused for all particles to reduce the number of calculations per particle. The two terms $\cos(\theta_G + \nu_E t)$ and $\sin(\theta_G + \nu_E t)$ needed for the spherical harmonics components J2, C22 and S22 are both independent of the particle and constant for one time step. The calculations of the Sun's and Moon's position relative to the reference frame, and terms depending only on them, are also independent of the particle and constant for one time step. These values are calculated in the `setUp` method of the corresponding `AccelerationComponent`. At the beginning of each time step, the `AccelerationAccumulator` object calls these methods for all activated components and passes references to the results as parameters for the actual calculations done per particle.

During step 2. the `AccelerationAccumulator` iterates over all `Debris` objects, managed by the central `DebrisContainer`, and applies all activated components by accumulating the accelerations returned by the components `apply` method. For a given particle and a given time step, there are again some values, that can be calculated once and used in multiple calculations for different components. In particular, the calculations of a_{Sun} and a_{SRP} both depend on the distance between the particle and the Sun. So if both components have to be calculated the value is only calculated once and passed as a parameter to the `apply` methods of the `SRPComponent` after being calculated by the `SolComponent`.

Algorithm 1: Acceleration Step

```
   Data: container
 1 if config[SOL] or config[SRP] then
 2     sun_params ← SolComponent.setUp(t)
 3 if config[LUN] then
 4     moon_params ← LunComponent.setUp(t)
 5 if config[C22] or config[S22] then
 6     c_term ← cos(θ_G + ν_E t)
 7     s_term ← sin(θ_G + ν_E t)
 8 forall d ∈ container do
 9     new_acc_total ← 0⃗
10     d_srp ← 0
11     if config[KEP] then
12         new_acc_total ← new_acc_total + KepComponent.apply(d)
13     if config[J2] then
14         new_acc_total ← new_acc_total + J2Component.apply(d)
15     if confi[C22] and config[S22] then
16         new_acc_total ←
             new_acc_total + C22S22Component.apply(d, c_term, s_term)
17     else
18         if confi[C22] then
19             new_acc_total ←
                 new_acc_total + C22Component.apply(d, c_term, s_term)
20         if config[S22] then
21             new_acc_total ←
                 new_acc_total + S22Component.apply(d, c_term, s_term)
22     if config[SOL] then
23         new_acc_total ←
             new_acc_total + SolComponent.apply(d, d_srp, sun_params)
24     if config[LUN] then
25         new_acc_total ← new_acc_total + LunComponent.apply(d, moon_params)
26     if config[SRP] then
27         new_acc_total ←
             new_acc_total + SRPComponent.apply(d, d_srp, sun_params)
28     if config[DRAG] then
29         new_acc_total ← new_acc_total + DragComponent.apply(d)
30     d.acc_t1 ← new_acc_total
```

Figure 4.1.: Acceleration calculation per time step

Two other components that can share calculation results are the `C22Component` and

`S22Component`. Because the Equations 2.25 and 2.26 share a lot of terms, an extra `AccelerationComponent` `C22S22Component` is introduced. This new component calculates the sum $a_{C22} + a_{S22}$ in one `apply` method, taking advantage of the similarity between these components. As the last step for each particle, the accumulated acceleration is written to the `Debris` object. The resulting algorithm is depicted in Figure 4.1 for one time step. To achieve better calculation performance, some basic optimizations are made for all components whenever possible.

Calculate constants up front If the equations to calculate the acceleration contain terms that can be precalculated at compile time, the `AccelerationComponent` provides constant methods to avoid recalculating these terms.

Replace divisions with multiplications For many architectures the division operation is significantly slower than the multiplication operation. If the divisions $\frac{b}{a}$ and $\frac{c}{a}$ must be calculated, the number of division operations can be reduced by calculating the inverse $a^{-1} = \frac{1}{a}$ once and replacing all divisions by a with a multiplication with a^{-1} to ba^{-1} and ca^{-1}.

Factorize terms and calculate factors once If two terms a and b can be factorized into the form $a = cd$ and $b = ce$ the shared factor c is calculated only once.

Use multiplication instead of `std::pow` method To avoid overhead when using the standard library `std::pow`, all powers are hardcoded as multiplications.

To avoid numerical errors, due to rounding errors, when adding to numbers of different magnitude, the operations are ordered in such a way, that numbers are added in ascending order. This is mainly used for the series expansions in Equations 2.11., 2.12 and 2.13 while calculating a_{Lun}.

4.3. Numerical Integration

The class diagram in Figure 4.3 shows the classes responsible for integrating.

To drive the integration over time the `Integrator` class is used. An `Integrator` object holds references to the central `DebrisContainer` object, holding the particles to simulate, and an `AccelerationAccumulator` object with the desired acceleration configuration. Per time step the Integrator object updates the state for each particle in the container object following the leapfrog method. During one time step, the `Integrator` object first updates the position, because the leapfrog method only needs position, velocity, and acceleration of the last time step. The `Integrator` then delegates the calculation of the acceleration for the current time step to its `AccelerationAccumulator` object, by calling the `applyComponents` method.

As the last step, the velocity can be updated because it depends on the position and acceleration of the current time step. The algorithm is illustrated in Figure 4.2.

Algorithm 2: Integrator Step

Data: container

foreach $d \in container$ **do**
  $d.acc\_t0 \leftarrow d.acc\_t1$
  $d.position \leftarrow d.position * \Delta t * d.velocity + \frac{1}{2}\Delta t^2 * s.acc\_t0$
$applyComponents()$
foreach $d \in container$ **do**
  $d.velocity \leftarrow d.velocity + \frac{1}{2}\Delta t * (d.acc\_t0 + d.acc\_t1)$

Figure 4.2.: Integration algorithm per time step

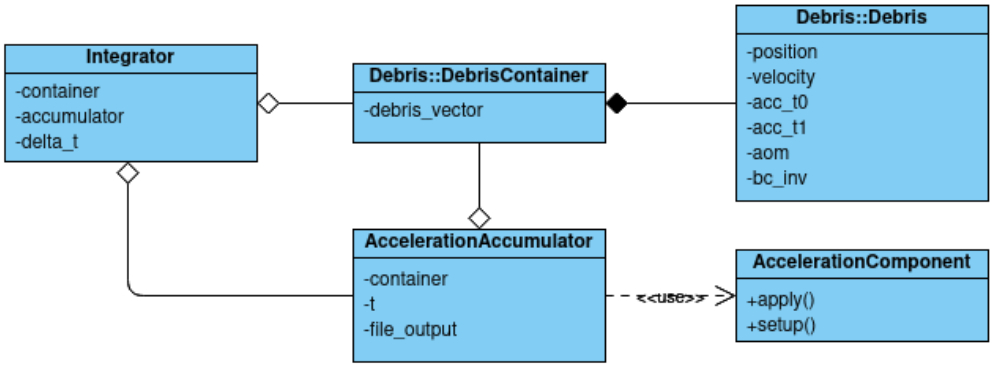


Figure 4.3.: Classes involved in physics calculations

4.4. I/O

To load the starting conditions of a simulation and to write the current state of the simulation at a given timestep, the developed software uses file input and output classes. At the beginning of the program, the user provides an input file name and an output file name via the command line.

The class diagram in Figure 4.4 shows the classes associated with input and output.

4.4.1. File Input

To define the starting conditions of a simulation an input text file is used. This file specifies the start time, the end time, the time step for the simulation and a time step for output of the complete state of the simulation. The configuration information to use for the `AccelerationAccumulator` is also given as a list of boolean values specifying if the corresponding `AccelerationComponent` should be applied. As part of this input file, an arbitrary number of particles to simulate can be listed. For these particles, all properties of the `Debris` class can be set to their starting conditions.

To load the simulation specifications into the software the `FileInput` class is used. A `FileInput` object holds a reference to the central `DebrisContainer` to add particles while

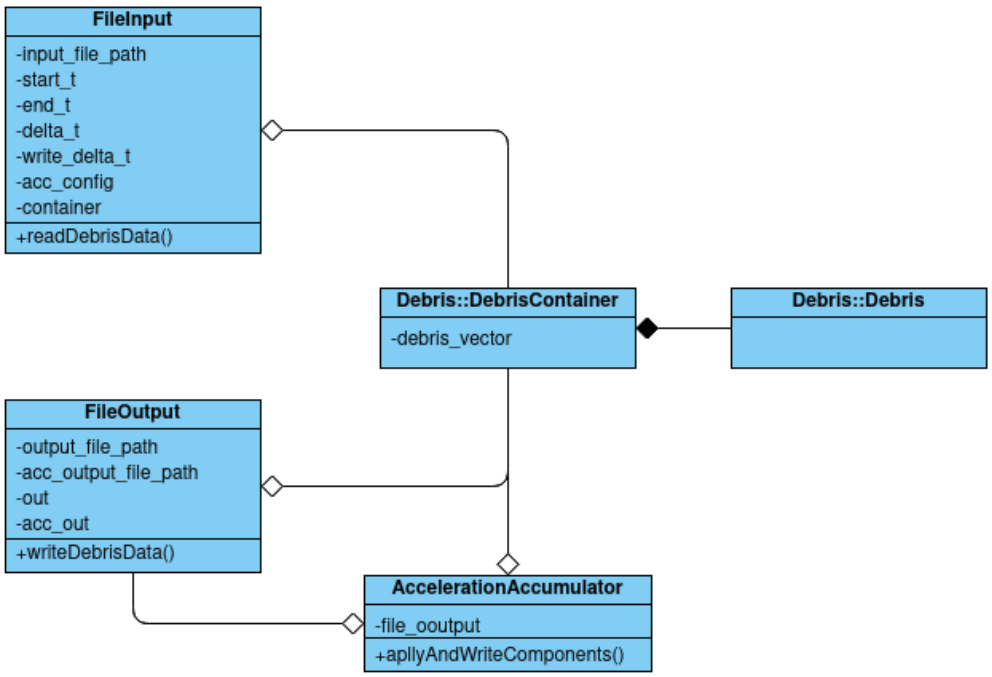


Figure 4.4.: Classes involved in I/O

reading the particles starting conditions. Additionally, it provides an interface to give access to the time and configuration variables.

4.4.2. File Output

The output of the simulation state is realized using the `FileOutput` class. As output files, comma-separated values (CSV) files are used.

A `FileOutput` object manages two file output streams, `out` and `acc_out`, and appends the state of the simulation, to the files at each writing time step. The `out` stream is used to safe the state of all `Debris` objects of the simulation and `acc_out` to write the accelerations of all `AccelerationComponents` calculated. It holds references to the central `DebrisContainer` object and is by itself referenced by the `AcclerationAccumulator` to provide output functionality during the acceleration calculations.

The time step for writing the state of the simulation by the `FileOutput` object is defined in the input text file. If the time since the last output is greater than this time step the main simulation loop calls the `writeDebrisData` method of the `FileOutput` object and triggers the use of the `AccelerationAccumulator`'s `applyAndWriteComponents` method. This method performs the same calculations described in Section 4.2, but after calculating the acceleration of a component it uses the `FileOutput` object to write the result to the output file.

Part III.

Results

5. Results

5.1. Used Hardware and Software

The following data from the developed software is produced by running simulations on the CoolMUC-2 segment of the Linux cluster at the Leibniz-Rechenzentrum der Bayerischen Akademie der Wissenschaft.[1] Some Key specifications of this Hardware can be found in Table 5.1

CoolMUC-2	
CPU	Intel Xeon E5-2697 v3[a]
Number of nodes	812
Cores per node	28
Hyperthreads per core	2
Core nominal frequency	2.6 GHz
Memory (DDR4) per node	64 GB (Bandwidth 120 GB/s - STREAM)

Table 5.1.: CoolMUC-2 Hardware specifications

Since the Developed software does not use parallelization the simulations are run only on one node of the cluster. The used compiler is clang version 10.0.0 using the O3 optimization level.

5.2. Error Analysis

One of the important properties of a numerical integration application, like the one developed for this thesis, is the achieved error convergence.

For this reason, the *heyoka* tool described in Section 3 is used to simulate one particle to generate synthetic ground-truth data. Due to *heyoka*'s configurable maximum error, the accuracy of this data can be ensured to be at the same level as machine precision ($1e-23$ for this analysis). The error metric for this analysis is the relative error in the single components of the velocity vector, averaged over all three components.

$$f = \frac{1}{3}\left(\left|\frac{v_{heyoka,x} - v_x}{v_{heyoka,x}}\right| + \left|\frac{v_{heyoka,y} - v_y}{v_{heyoka,y}}\right| + \left|\frac{v_{heyoka,z} - v_z}{v_{heyoka,z}}\right|\right) \tag{5.1}$$

Because the leapfrog method described in Section 4.3 is a numerical integration method of order 2, the accumulated global error should be $O(\Delta t^2)$. To analyze the error f for decreasing Δt, a system with the same starting conditions is simulated multiple times. After each simulation, the used Δt is halved. The simulated particles orbit at an altitude of 500

[1] `https://doku.lrz.de/display/PUBLIC/CoolMUC-2`

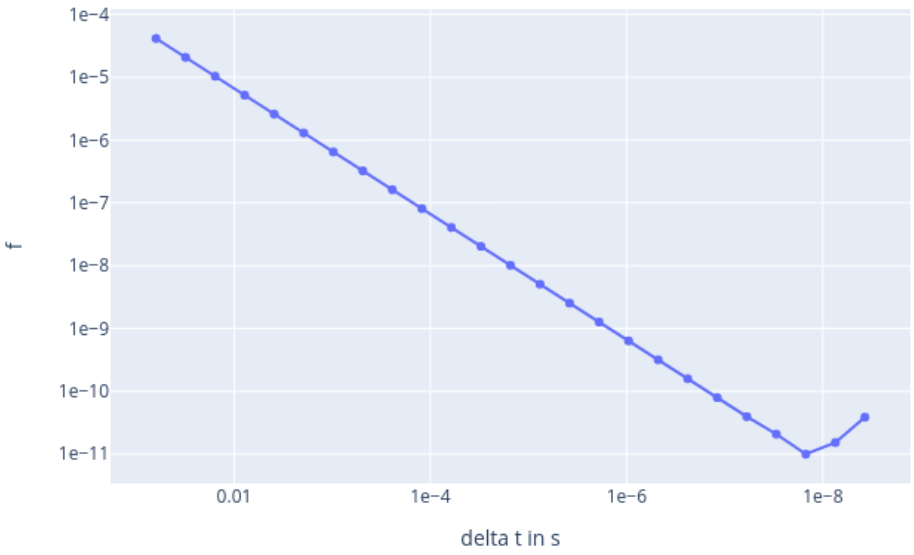


Figure 5.1.: Relative velocity error average of x,y,z components

km above ground at a velocity of 7.61 km/s. In Figure 5.1 the accumulated global error f at the end of these simulations is shown for decreasing Δt. As expected the error decreases with smaller Δt, but after reaching its minimum of $f \sim 1e-11$ at $\Delta t \sim 1e-8$ the error starts to increase again.

To get a better understanding of the error over time, the development of the resulting accumulated global errors over time, for $2^{-21} >= \Delta t >= 2^{-27}$, are shown in Figure 5.2. As can be seen in the figure a value of $\Delta t < 2^{-25}$ leads to oscillations in the error, exceeding even the errors for greater Δt. This behavior can be explained by rounding errors, due to the use of floating-point numbers for representing the state of an object. Because a decreasing time step size leads to decreasing magnitude of the updates to apply to the position and velocity vectors, if the update steps become small enough relative to these vectors, the rounding of the update result towards the nearest valid floating-point representation deletes more significant bits of the integration step than for bigger time steps.

In practice, this behavior at small Δt can be ignored, because when using an integration time step of $\Delta t = 1e-8$ the time to run the simulation exceeds the time simulated by magnitudes. The leapfrog method is used for the numerical integration in the developed software, because it is a simple to implement second-order integration method, that only requires one evaluation of the acceleration per time step. Unfortunately, this results in the need to choose a small Δt to minimize the global integration error. E.g. to achieve a relative error smaller than $1e-8$, Δt has to be smaller than $2^{-16} = 1.525878e-5$. Such small time steps slow down the simulation. To address this issue another, higher-order, integration method could be used.

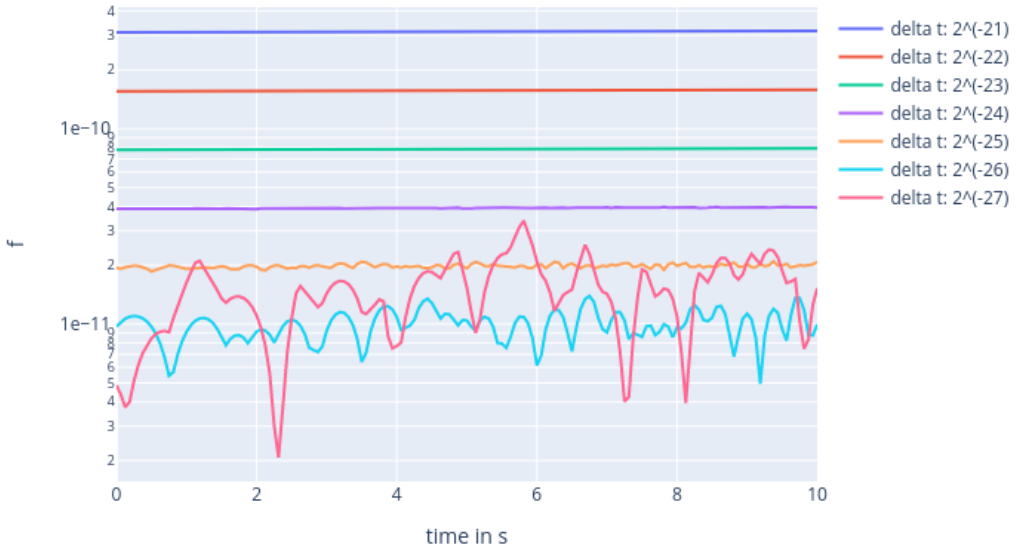


Figure 5.2.: Relative velocity error average of x,y,z components

5.3. Calculation Speed

To investigate the performance of the developed software in terms of calculation speed, the number of particle updates per second is used. This data is created running simulations with 1 to $2^{19} = 524288$ particles, by doubling the number of particles for each simulation.

Because only the number of particles is important all these particles have the same starting condition, i.e. they are in an orbit at an altitude of 500 km above ground at a velocity of 7.61 km/s. After measuring the seconds t_{sim} to run a simulation with n particles for 10000 time steps, the number of particle updates per second

$$\frac{\#updates}{second} = \frac{n \cdot 10000}{t_{sim}} \tag{5.2}$$

can be calculated. For each particle count, an average of 10 such simulations is used. The simulations are performed without file output.

Figure 5.3 shows the behavior of the updates per second with an increasing number of simulated particles. As can be seen in the graph, for a small number of particles, the number of updates per second is rapidly increasing. This initial behavior can be explained by the particle independent calculations described in Section 4.2 done at the beginning of each time step, that are performed, regardless of the number of particles. If these calculations take x seconds, the fraction $\frac{x}{n}$ seconds must be added to the time needed to calculate the acceleration of the particle, to get the time spend updating one particle. Because this overhead per particle decreases with increasing n, for large simulations, it can be neglected

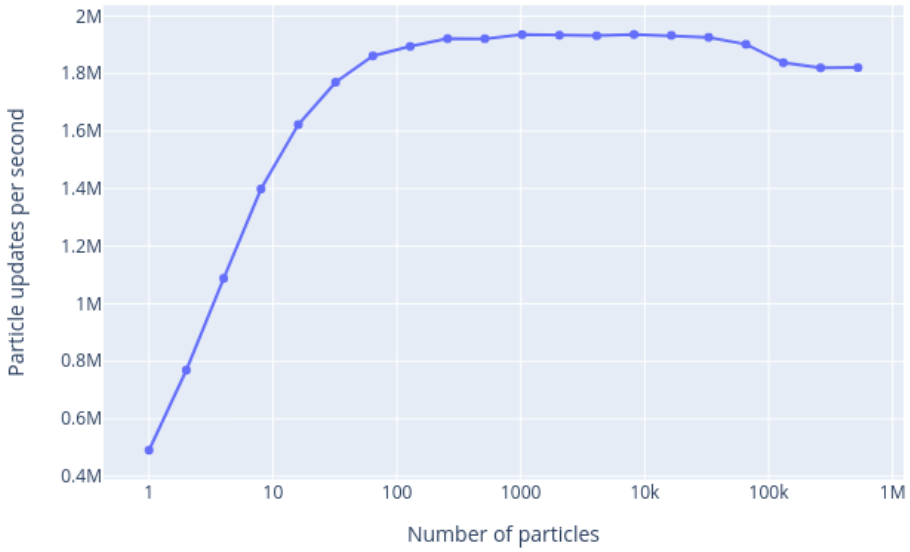


Figure 5.3.: Particle updates per second for different number of particles

and the update time per particle is the time needed to calculate its acceleration. For the given graph this point is reached at a simulation size of 128 particles. At this simulation size a plateau, around 1.9 million particle updates per second, is reached.

The update rate starts to decrease slightly if n gets greater than 100000 particles. Probably this decrease at greater n can be explained by cache effects. If the number of particles becomes high enough, the particles already updated during the current time step will be removed from the cache, to load the new particle data. When the next time step is starting, the particle data then has to be loaded from slower memory, resulting in a loss of update speed per particle.

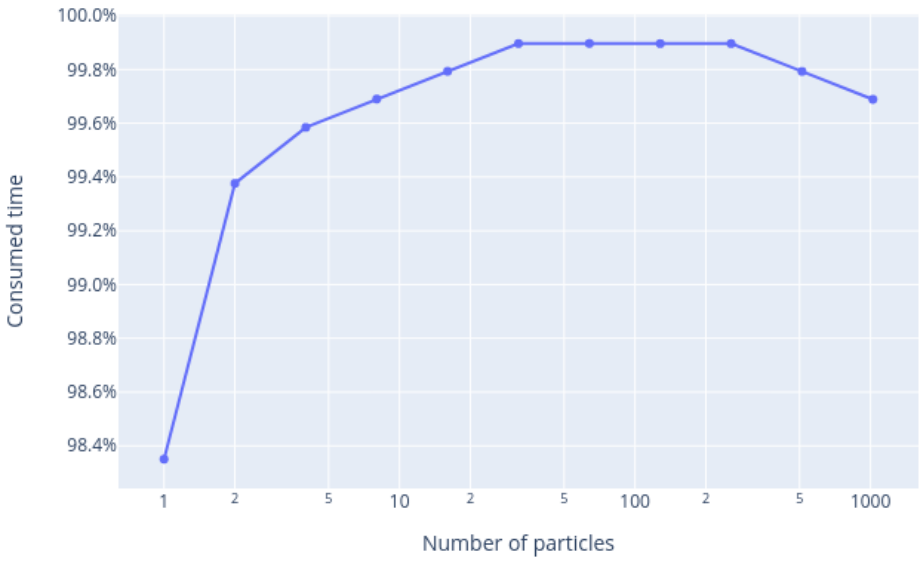


Figure 5.4.: Time consumption of `integrate` function relative to the `runSimulation` function

5.4. Integration profiling

Based on the results of the last Section 5.3, the same simulations up to $n = 1024$ are profiled using the version 5.4.44 of the perf[2] tool. The results of these profiling runs are used to analyze the computation time needed for different simulation steps.

5.4.1. Integration Time Compared to Main Simulation Loop

First, the time consumption of the `integrate` method of the `Integrator` class is compared to the run time of the main simulation loop. The result is depicted in Figure 5.4. As expected the time spent in the `integrate` method is almost the complete time spent inside the simulation loop. The minimum of 98.35% for a single particle and the maximum of 99.9% for 32 to 256 particles differ only by 1.55% of the total run time.

5.4.2. Computation Time Distribution Inside `integrate`

Because of this result, analyzing the time consumption inside the `integrate` method is a good way to find computationally intensive parts of the software overall.

The result of profiling the `integrate` method can be seen in Figure 5.5. For the `integrate` method the distribution of the time consumption of the different methods called are roughly

[2] https://perf.wiki.kernel.org/index.php/Main_Page

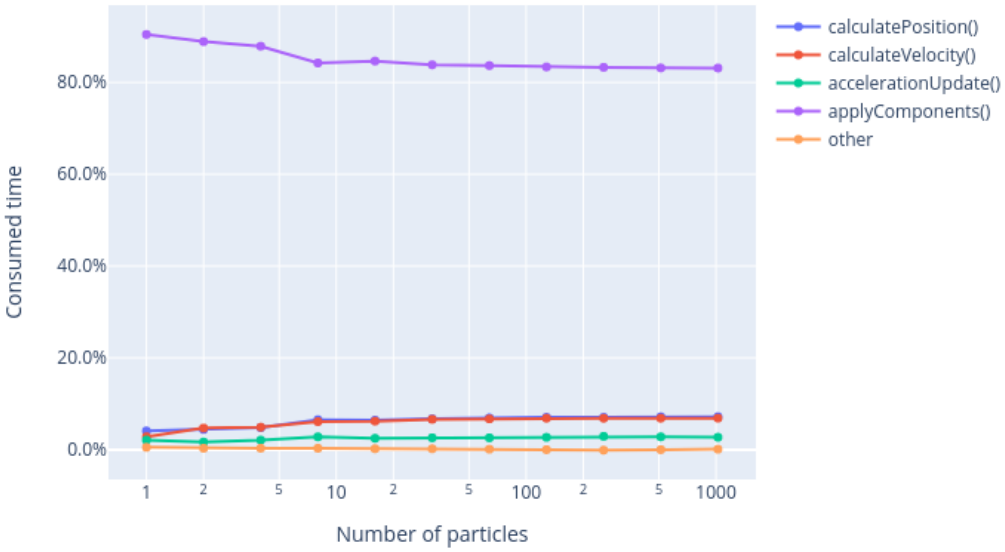


Figure 5.5.: Time consumption of integration sub steps relative to the `integrate` function

the same for increasing n. The calculation of the particles' accelerations starts at 90% for $n = 1$ and slightly decreases with higher n till it stabilizes around 83%. This higher time consumption for smaller n can be explained with the same, particle independent calculations, that explains the lower update rate at small n discussed in Section 5.3. The time consumption of the rest of the called methods inside `integrate`, increases slightly with higher n, but none of them disproportionately compared to each other.

Inside the `integrate` method, there are two main types of actions, that use up 99.8% of the computation time spent inside the `integrate` method. The first one and by far the more time-consuming action is the calculation of the accelerations of the particles. With an increasing number of particles the `applyComponetns` method needs around 83% of the consumed time. The other type of action is the update of the particles' state. Adding up the times needed to write the acceleration of the last time step from the `acc_t1` to the `acc_t0` variable, updating the `position` and the `velocity` of the `Debris` objects, results in a time consumption of $2.8\% + 7.1\% + 6.9\% = 16.8\%$ for large n.

5.4.3. Computation Time Distribution Inside `applyComponents`

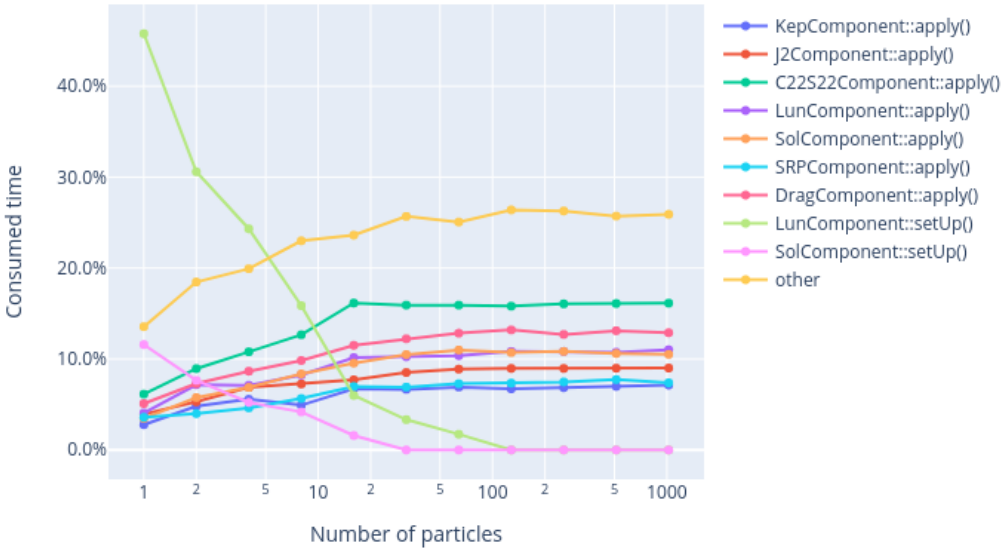


Figure 5.6.: Time consumption of acceleration calculation sub steps relative to the `applyComponents` function. The point "other" mainly combines different kinds of data access

The last method to investigate with respect to the consumption of computation time is the `applyComponents` method of the `AccelerationAccumulator` class. The profiling result of this method is shown in Figure 5.6

As can be seen in the graph, the distribution of computation time consumption inside the `applyComponents` method changes significantly with increasing particle number n. For small n the computation time spent inside the `setUp` functions of the `LunComponent` and the `SolComponent` dominate the run time of the whole method. For a single particle, the two functions consume $45.8\% + 11.6\% = 57.4\%$ of the complete computation time. This is caused by the complex calculations to determine the Sun's and the Moon's position described in Section 2.2.2. However, as described above, for an increasing number of particles the calculations of the particle's accelerations start to dominate the required computation time. This leads to a decreasing portion of the complete computation time inside the `applyComponents` method. The `SolComponent.setUp` functions portion vanishes at $n = 32$ and `LunComponent.setUp` at $n = 128$, i.e. they need less than 1% of the complete computation time.

The `apply` functions of the seven `AccelerationComponents` start with portions of the spent computation time in a range between 2.8% for the `KepComponent` and 6.1% for the `C22S22Component` for a single simulated particle. The sum of all `apply` functions results in

a total portion of 29.1% of the complete computation time. For $n = 1024$ these values span from 7.1% for the `KepComponent`, 16.1% for the `C22S22Component` and a total of 74.0% for all `AccelerationComponents`.

In contrast to Figures 5.4 and 5.5, a great portion of computation time is spent to access data. The different types of data access during the `applyComponents` method are responsible for almost the complete portion of computation time labeled "other" in Figure 5.6. For an increasing number of particles this fraction makes up between 13.6% for $n = 1$ and around 26% for $n > 128$. The fact that data access is responsible for such a great amount of computation time, makes it a promising starting point to optimize the used data structures for faster data access. This approach would also reduce the time needed to access the particle data during updating their state inside the `integrate` method analyzed in the last section.

5.5. Acceleration Component Influences

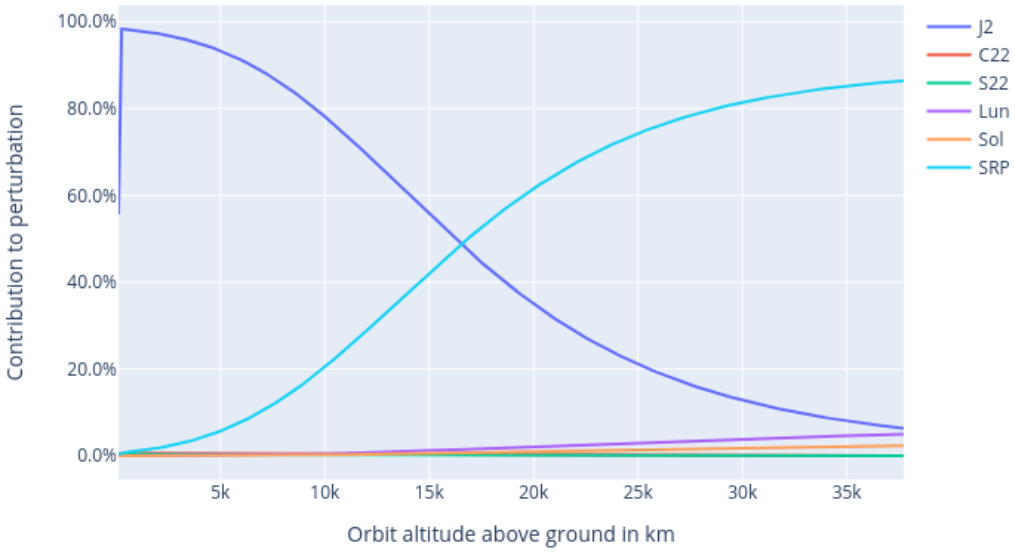


Figure 5.7.: Contributions of perturbation accelerations for different orbit altitudes

After analyzing the computation time needed to calculate the acceleration of a particle, due to the different `AccelerationComponents`, this section analyzes the influence of these components compared to each other depending on the orbit altitude of the particle.

Because the different `AccelerationComponents` can depend on the current time t the data used is averaged over different points in time. For 365 days, starting at $t = 0$, the accelerations for orbit altitudes between 100 km and 377600 km are calculated. For each of the orbit altitudes, 24 particles are taken into account. The velocity used for the particles is

the orbital velocity v_{orbit} determined using the equation[3]

$$v_{orbit} = \sqrt{\frac{398600.5}{R_{\oplus} + altitude}} \tag{5.3}$$

.

These particles cover both protected regions depicted in Figure 1.2

Because the acceleration caused by the `DragComponent` quickly vanishes with the orbit altitude, additional data for orbit altitudes between 10 km and 1100 km is used to analyze the `DragComponent` separately.

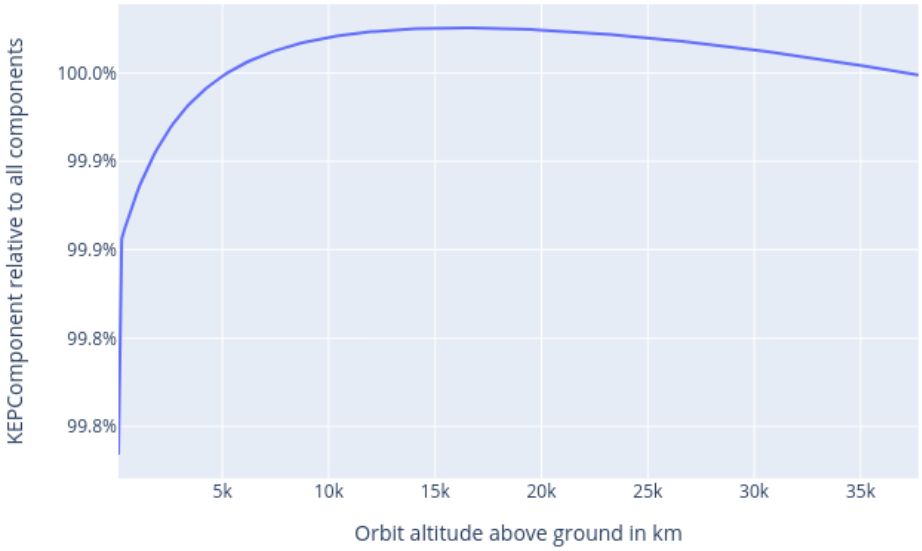
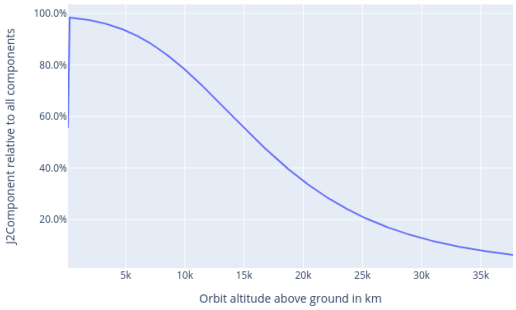


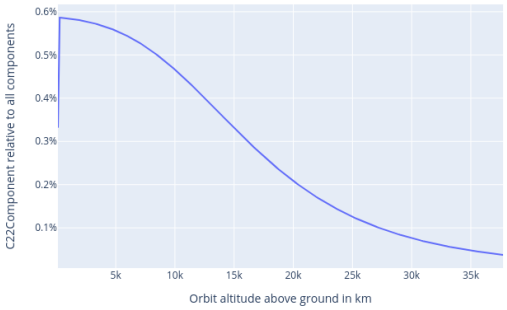
Figure 5.8.: Contribution of `KepComponent` acceleration for different orbit altitudes

At first the relation between a_{kep} and the sum of all perturbing accelerations used in the simulation is analyzed. The contribution of the `KepComponent` depending on the orbit altitude is shown in Figure 5.8. As expected at low orbits the contribution of the `KepComponent` is the smallest of all altitudes, i.e. 99.74% of the complete acceleration is caused by the `KepComponent` and 0.26% are caused by all perturbations. With increasing orbit altitude perturbations caused by the proximity to the Earth, i.e. atmospheric drag and spherical harmonics, contribute less (Figures 5.10, 5.9a, 5.9b and 5.9c). At an orbit altitude of around 17000 km, the maximum contribution of the `KepComponent` is reached. Above 17000 km the gravitational influence due to third bodies, i.e. the Sun and the Moon, as well as the contribution of the solar radiation pressure, become more dominant and the contribution of the `KepComponent` decreases again (Figures 5.9d, 5.9e and 5.9f).

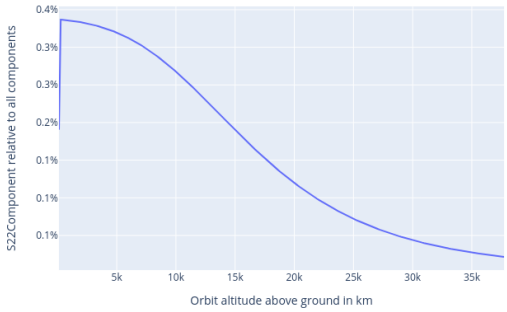
[3] `https://keisan.casio.com/exec/system/1224665242`



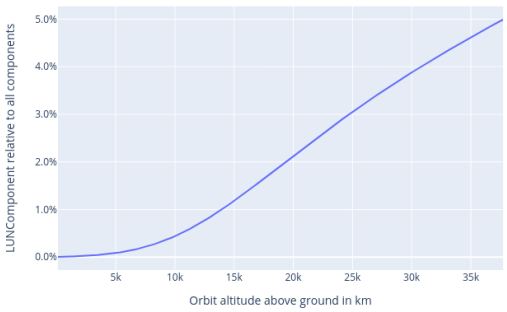
(a) J2Component



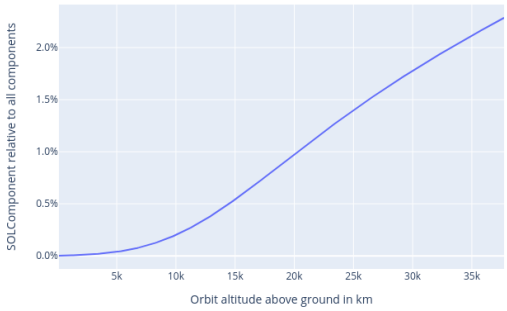
(b) C22Component



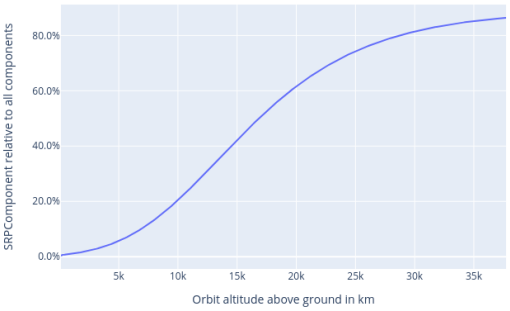
(c) S22Component



(d) LunComponent



(e) SolComponent



(f) SRPComponent

Figure 5.9.: Contributions of perturbation accelerations for different orbit altitudes

In Figure 5.9 the contributions of the `AccelerationComponents` except the `DragComponent` to the sum of all perturbations depending on the orbit altitude are displayed for the single components. Figure 5.7 combines the single-component graphs to get a better impression of the relation between the `AccelerationComponents`. After the `DragComponent` vanishes around 150 km altitude compared to the other perturbations (Figure 5.10) above this altitude the `J2Component` dominates the perturbations, i.e. its contribution is 98.4% of all perturbations at an altitude of a few hundred km. As described above with increasing orbit altitude the spherical harmonics contribute less and less, until around 17000 km altitude the solar radiation pressure starts to dominate the perturbations, i.e. its contribution to all perturbations is greater than 50% above this altitude.

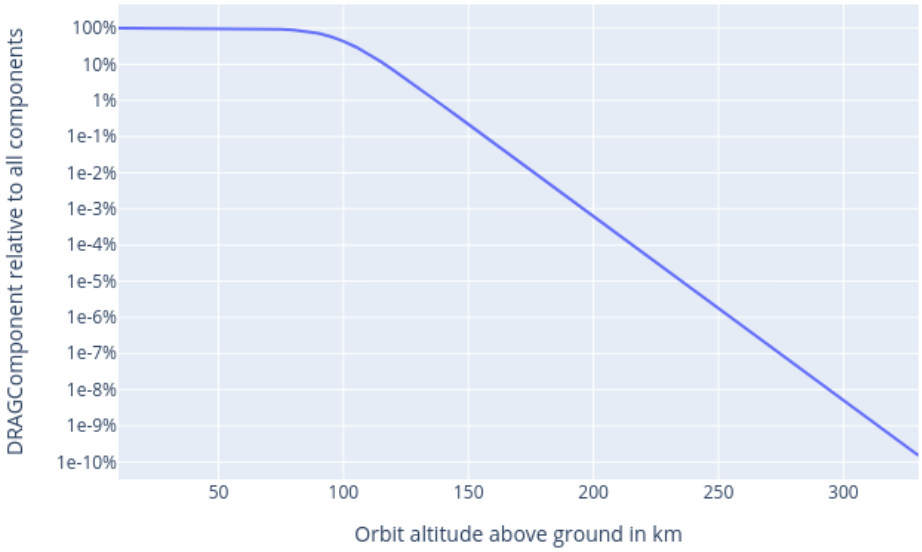


Figure 5.10.: Contribution of DragComponent acceleration for different orbit altitudes

In the end, the `DragComponent`'s behavior is discussed separately. Because the used model of the atmosphere described in Section 2.2.5 results in the rapidly decreasing contribution of the `DragComponent` to all perturbations, the acceleration can be neglected above an orbit altitude of a few hundred km. This behavior is illustrated in Figure 5.10.

If the needed computation time for the `DragComponent.apply` function analyzed in Section 5.4.3 results in an acceleration that can be neglected, a speedup of over 10% could be achieved, for higher altitudes, by introducing an orbit above which the call to `DragComponent.apply` is skipped.

Part IV.

Conclusion

6. Conclusion

As result of this thesis, the developed software can simulate hundreds of thousands of particles, without slowing down the simulation, due to the large number of particles, achieving an update rate of around 2 million particle updates per second. But the number of updates alone doesn't say much about the performance of the software, because if the used integration time step is small, the total simulated time is smaller too. At this point the order of the used integration method must be considered, because a higher-order method allows for bigger time steps. The error of the simulations achieves convergence for reasonable integration time steps. But because the used integration method is only second-order, this convergence is rather slow.

Building upon the software solution developed for this thesis, multiple approaches could be followed, to further improve the performance of the software.

Optimizing the used data structures to allow faster data access while iterating over all particles, could enhance simulation speed. One approach could be the use of a structure of arrays in contrast to the used array of structures. This approach could achieve better performance while looping over the particles to simulate, because for most calculations done during the simulation only one property of the particles is needed. Using a structure of arrays to represent the particles could lead to faster memory access because with the first particle loaded of each cache line the needed values for the next particles would be cached and loaded faster in the future.

Incorporating another numerical integration method, that has higher order than the used leapfrog method, could result in a better error behavior of the software. This would allow increasing the time steps used for integration, to speed up simulations, without increasing the integration error of the result.

The developed software solution does not take advantage of possible parallelization. Because the calculations performed per particle are completely independent, parallelization is probably the best place to start to achieve higher simulation speed. As long as the particles do not interact, this approach would result in a simply scalable software solution for simulating large numbers of particles. If the trajectory calculations of the particles are embedded into bigger software solutions, that needs the simulated particles to interact, i.e. collisions must be handled, parallelization can become harder to exploit, but still worth the efforts.

Part V.

Appendix

A. Constants

Constants		
Symbol	Value	Unit
$GM_{\oplus}$	$3.986004407799724 \times 10^5$	km^3/s^2
$GM_{\odot}$	$1.32712440018 \times 10^{11}$	km^3/s^2
$GM_{\mathcal{M}}$	4.9028×10^3	km^3/s^2
$R_{\oplus}$	6378.1363	km
C_{20}	$-4.84165371736 \times 10^{-4}$	
C_{22}	$2.43914352398 \times 10^{-6}$	
S_{22}	$-1.40016683654 \times 10^{-6}$	
θ_G	280.4606	$\circ$
ν_E	$4.178074622024230 \times 10^{-3}$	$\circ/s$
$\nu_{\odot}$	$1.1407410259335311 \times 10^{-5}$	$\circ/s$
ν_{M_a}	$1.512151961904581 \times 10^{-4}$	$\circ/s$
ν_{M_p}	$1.2893925235125941 \times 10^{-6}$	$\circ/s$
ν_{M_s}	$6.128913003523574 \times 10^{-7}$	$\circ/s$
$a_{\odot}$	1.49619×10^8	km
$\varphi_{\odot,0}$	357.5256	$\circ$
$\Omega_{\odot} + \omega_{\odot}$	282.94	$\circ$
P_{SRP}	4.56×10^{-6}	N/m^2
p_0	1.3	kg/m^3
H	8.5	km
$\omega_{\oplus}$	7.292115×10^{-5}	rad/s

Table A.1.: Constants

List of Figures

List of Tables

Bibliography

[Art] Two ways to write the leapfrog, `http://www.artcompsci.org/vol_1/v1_web/node34.html` (Date accessed 22. 08. 2021).

[BI21] Francesco Biscani and Dario Izzo. *Revisiting high-order Taylor methods for astrodynamics and celestial mechanics*, volume 504. 04 2021.

[ESA21] *ESA's annual space environment report.* Number 5.0. ESA, 2021.

[Esc65] P.R. Escobal. *Methods of Orbit Determination.* J. Wiley, 1965.

[Fit12] R. Fitzpatrick. *An Introduction to Celestial Mechanics.* Cambridge University Press, 2012.

[IAD07] *IADC Space Debris Mitigation Guidelines.* Number 1. IADC, 2007.

[KCP78] Donald J. Kessler and Burton G. Cour-Palais. Collision frequency of artificial satellites: The creation of a debris belt. *Journal of Geophysical Research: Space Physics*, 83(A6):2637–2646, 1978.

[LPL+89] A.C. Long, R. Pajerski, R. Luczak, United States. National Aeronautics, Space Administration, and Goddard Space Flight Center. *Goddard Trajectory Determination System (GTDS): Mathematical Theory.* Goddard Space Flight Center, 1989.

[Ros98] D. Rose. *International space station familiarization.* NASA, 1998.

[VCHK06] David Vallado, Paul Crawford, Ricahrd Hujsak, and T.S. Kelso. Revisiting spacetrack report 3. volume 3, 08 2006.

[VM97a] D.A. Vallado and W.D. McClain. *Application: Moon Position Vector*, chapter 3.6. Primis, 1997.

[VM97b] D.A. Vallado and W.D. McClain. *Application: Simplified Acceleration Model*, chapter 7.7.1. Primis, 1997.

[VM97c] D.A. Vallado and W.D. McClain. *Coordinate Systems*, chapter 1.4. Primis, 1997.

[VM97d] D.A. Vallado and W.D. McClain. *Cowell's Formulation*, chapter 7.4. Primis, 1997.

[VM97e] D.A. Vallado and W.D. McClain. *Fundamentals of Astrodynamics and Applications.* Primis, 1997.

[VM97f] D.A. Vallado and W.D. McClain. *Gravity Field of a Central Body*, chapter 7.6.1. Primis, 1997.

[VM97g] D.A. Vallado and W.D. McClain. *Nutation (J2000)*, chapter 1.7.2. Primis, 1997.

[VM97h] D.A. Vallado and W.D. McClain. *Three-body and n-body Equations*, chapter 2.3. Primis, 1997.

[VM97i] D.A. Vallado and W.D. McClain. *Two-body Equation*, chapter 2.2. Primis, 1997.

[VM07] D.A. Vallado and W.D. McClain. *Atmospheric Drag*, chapter 8.6.2. Microcosm Press/Springer, 2007.

[You13] P. Young. The leapfrog method and other "symplectic" algorithms for integrating newton's laws of motion. UC Santa Cruz, 2013.