Sonnensystem Simulation

Svenson Han Göthel, 25 Januar 2025

(Lloyd Gymnasium Bremerhaven, 6c)

Inhaltsverzeichnis

Motivation

In unserem Informatikkurs (Programmierkurs) haben wir eine Bibliothek (Framework) inklusive Beispiele entwickelt, welche ich schon in dem Projekt Visualisierung von π nutzte und erweiterte. In dieser Bibliothek – quasi ein Physiklabor - habe ich auch mit Gravitationsberechnungen in Spielen experimentiert, z.B. spacewars (webapp) und canonball (webapp).

Dieses Projekt soll zeigen, ob eine Gravitations-Simulation der Planetenumlaufbahnen unseres Sonnensystem im Vergleich zur Keplersche Bahnbestimmung ausreichend genau ist.

Webanwendung und Source Code

Der Source Code ist in dem Projekt gfxbox2 verfuegbar und kann auf Linux, FreeBSD oder anderen Unixen kompiliert werden.

Dieses Programm kann auch im Webbrowser über diesen Link ausgeführt werden.

Der komplette Source Code dieses Programms kann hier direkt angeschaut werden.

Übersicht

Dieses Computerprogramm simuliert den Orbit aller Planeten unseres Sonnensystems. Die Simulation berechnet die Umlaufbahn der Planeten. Hier werden mindestens einmal pro simulierter Stunde die Geschwindigkeitsvektoren aller Planeten durch die Gravitation aller Himmelskörper verändert. Diese Simulation nutzt die newtonschen Gravitationsgesetze und nicht die auf Beobachtung beruhende Keplersche Bahnbestimmung. Zusätzlich wird das simulierte System visualisiert. Außerdem können zur Verifikation Referenzpunkte mit der Simulation verglichen werden. Die Horizon JPL Datenbank wird benutzt. Relevante Datenpunkte werden in diesem Dokument festgehalten, als auch Position und Geschwindigkeitsvektor jedes Planeten für den Simulationsstart und für die Referenzpunkte in dem Computerprogramm genutzt.

Hintergrund und theoretische Grundlagen

Jeder Planet hat eine eigene Umlaufbahn in der er um die Sonne kreist. Die Umlaufbahn ist kein Ding was man anfassen oder sehen kann. Man stellt sie sich nur vor, weil zum Beispiel die Erde jedes Jahr fast an die gleiche Stelle ankommt. Die Planetenumlaufbahnen entstehen durch den bleibenden Einflüssen der Massen anderer Himmelskörper auf den Geschwindigkeitsvektor. Eine Masse steht in Wechselwirkung mit anderen Massen, sie ziehen sich räumlich an. Diese Erscheinung wird Gravitation genannt. Die Gravitation ist eine Beschleunigung. Die Berechnung des Gravitationseinflusses wird in den folgenden Kapiteln beschrieben.

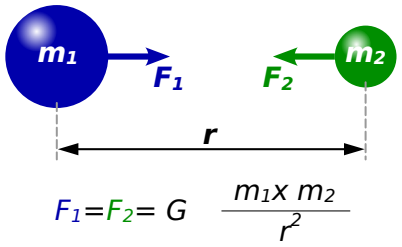
Implementation

Berechnung des Gravitationseinflusses

Unsere Simulation nutzt die Horizon JPL Datenbank. Die relevanten Daten sind in diesem Dokument festgehalten, als auch für die 3D Position und Geschwindigkeitsvektor der Start- und Referenzpunkte genutzt.

Methode 1

Diese Methode basiert auf die newtonschen Gravitationsgesetze.



Von I, Dennis Nilsson, CC BY 3.0

Im folgendem wird *d* für die Massendistanz *r* genutzt.

Formel

$F = m * a = [N] = [kg * \frac{m}{s^2}]$, Kraft (allgemein)

$G = [N] * [\frac{m^2}{kg^2}] = [kg * \frac{m}{s^2}] * [\frac{m^2}{kg^2}] = [\frac{m^3}{s^2 * kg}]$, Gravitationskonstante

$\vec{d} = p_1 - p_2$, Planetendistanzvektor von p_2 nach p_1, Zentrum der Planeten 2 & 1

$F = F_1 = F_2 = G * \frac{m_1 * m_2}{|\vec{d}|^2} = [N] = [\frac{m^3}{s^2 * kg}] * \frac{[kg^2]}{[m^2]} = [kg * \frac{m}{s^2}]$,
Gravitationskraft mit m_1 und m_2, Masse von den Planeten 1 & 2

$g_2 = \frac{F}{m_2} = G * \frac{m_1}{|\vec{d}|^2} = [\frac{m^3}{s^2 * kg}] * \frac{[kg]}{[m^2]} = [\frac{m}{s^2}]$,
Betrag der Gravitationbeschleunigung für Planet 2

$\vec{g_2} = \frac{\vec{d}}{|\vec{d}|} * g_2 = [\frac{\vec{m}}{s^2}]$,
Gravitationbeschleunigung für Planet 2 in Richtung Planet 1

Source Code

```
// Returns gravity [m/s^2] acceleration body `b` towards body `a`
static pixel::f3::vec_t gravity1(const CBody& a, const CBody& b) {
    pixel::f3::vec_t v_d = a.center - b.center;
    const double d = v_d.length();
    pixel::f3::vec_t v_g; // Gravitationsbeschleunigung (Ergebnis)
    if( !is_zero(d) ) {
        // normal-vector: v_d / d (einheitsvektor, richtung)
        v_g = ( v_d / d ) * (G * a.mass / ( d * d ));
    }
    return v_g;
}
```

Methode 2

Diese Methode basiert ebenfalls auf die newtonschen Gravitationsgesetzen und nutzt den Standard-Gravitationsparameter *GM*. Dies ist die Standard Methode und benutzt die bekannte Definition für F und G aus Methode 1.

Formel

$GM = G * m = [\frac{m^3}{s^2 * kg}] * [kg] = [\frac{m^3}{s^2}]$, Standard Gravitationsparameter

$\vec{d} = p_1 - p_2$, Planetendistanzvektor von p_2 nach p_1, Zentrum der Planeten 2 & 1

$F = F_1 = F_2 = G * \frac{m_1 * m_2}{|\vec{d}|^2} = GM_1 * \frac{m_2}{|\vec{d}|^2} = [N] = [\frac{m^3}{s^2}] * \frac{[kg]}{[m^2]} = [kg * \frac{m}{s^2}]$,
Gravitationskraft mit GM_1 und m_2, GM von Planet 1 und Masse von Planet 2

$g_2 = \frac{F}{m_2} = G * \frac{m_1}{|\vec{d}|^2} = \frac{GM_1}{|\vec{d}|^2} = [\frac{m^3}{s^2}] * [m^{-2}] = [\frac{m}{s^2}]$,
Betrag der Gravitationbeschleunigung für Planet 2

$\vec{g_2} = \frac{\vec{d}}{|\vec{d}|} * g_2 = [\frac{\overrightarrow{m}}{s^2}]$,
Gravitationbeschleunigung für Planet 2 in Richtung Planet 1

Source Code

```
// Returns gravity [m/s^2] acceleration body `b` towards body `a`
static pixel::f3::vec_t gravity2(const CBody& a, const CBody& b) {
    pixel::f3::vec_t v_d = a.center - b.center;
    const double d = v_d.length();
    pixel::f3::vec_t v_g; // Gravitationsbeschleunigung (Ergebnis)
    if( !is_zero(d) ) {
        // normal-vector: v_d / d (einheitsvektor, richtung)
        v_g = ( v_d / d ) * ( a.GM / ( d * d ) );
    }
    return v_g;
}
```

Zeit und Raum

Animation

Um eine flüssige Animation zu erhalten, wird das aktuelle Bild gezeigt und das nächste wird gemalt. Sobald der Monitor bereit ist das nächste Bild anzuzeigen, werden die beiden Bilder ausgetauscht. Bei einem Monitor mit 60Hz Bildwiederholrate ist die Periode bis zum nächsten Bild $\frac{1}{60}s = 16,\overline{66}ms$. Dies ist die maximale Dauer, um das nächste Bild zu erstellen (rendering).

Bei Einhaltung von 60Hz, ist die Periode zwischen den rendering Aufrufen im *mainloop* ~16,66 ms.

Planetenumlaufbahn

Es gibt eine Echtzeit, d.h. die Zeit vom Betrachter, und eine Simulationszeit, d.h. die Zeit des simulierten Sonnensystems. In der Regel wird ein Vielfaches der Echtzeit, z.B. 1 Monat in einer Sekunde simuliert. Das Verhältnis von Simulationszeit zur Echtzeit kann verändert werden.

Die Bewegung der Planeten erfolgt vom *mainloop* aus für jeden Planet. Nun wird für jeden Planeten *p* in dem Sonnensystem der Geschwindigkeitsvektor durch die Gravitationsbeschleunigung jeden anderen Planeten *q* verändert wird. Die Gravitationsbeschleunigung wirkt sich wie folgt auf den Geschwindigkeitsvektor und somit der Position aus:

$\vec{g} = [\frac{\overrightarrow{m}}{s^2}]$, Gravitationsbeschleunigung (siehe oben)

$\vec{v} = [\frac{\overrightarrow{m}}{s}]$, Geschwindigkeitsvektor mit erstem Wert aus Datenbank

p, Position mit erstem Wert aus Datenbank

$dt_{real} = t = [s]$, Dauer einer Periode in Echtzeit, vom rendering Aufruf zum nächsten

$timescale$, Verhältnis von Simulationszeit zur Echtzeit

$dt_{world} = dt_{real} * timescale = [s]$, Dauer einer Periode in Simulationszeit

$\vec{g} * dt_{world} = \overrightarrow{v_{add}} = [\frac{\overrightarrow{m}}{s^2}] * [s] = [\frac{\overrightarrow{m}}{s}]$

$\vec{v} = \vec{v} + \overrightarrow{v_{add}}$

$\vec{v} * dt_{world} = \vec{s} = [\frac{\overrightarrow{m}}{s}] * [s] = [\overrightarrow{m}]$

$p_n + \vec{s} = p_{n+1}$, Position + Strecke = neue Position

Animationsgenauigkeit

dt_{real} = Zeitdifferenz zwischen einem Bild malen und dem nächsten

dt_{world} = dt in Simulationszeit

Da das Produkt von Gravitationsbeschleunigung und dt_{world} (Geschwindigkeitsvektorsummand) zu ungenau ist, wurde der Animationsfunktion *tick* die Unterfunktion *sub_tick* eingeführt. Bei einem dt_{world} von ~6 Tagen (ein Jahr pro Sekunde) hat der Merkur das Sonnensystem verlassen. Ein solcher Simulationsschritt überspringt die feingranulare Wechselwirkung von Gravitation zu Geschwindigkeitsvektor und Position. Wir sprechen hier von der Auflösungsgenauigkeit der Simulation.
Um die Genauigkeit zu erhöhen, wird *n*-mal *sub_tick* mit $n = dt_{world} / 1h$ für einen Simulationstag aufgerufen.

Source Code

```
void tick(const fraction_timespec& dt, const int64_t time_scale) {
    constexpr fraction_timespec zero;
    constexpr fraction_timespec max_time_step(1_h);
    const fraction_timespec dt_world = dt * time_scale; // world [s]
    _time_scale_last = time_scale;

    fraction_timespec wts = _world_time;
    for(fraction_timespec i=dt_world; i > zero; i-=max_time_step ) {
        const fraction_timespec step = jau::min(i, max_time_step);
        sub_tick(step, wts);
        wts += step;
    }
    _d_sun = _center.length();

    _world_time += dt_world;
}
void sub_tick(const fraction_timespec& dt, const fraction_timespec& wts) {
    if( _id == cbodyid_t::sun && !with_oobj) {
        return;
    }
    const float dt_f = (float)dt.to_us()/1000000.0f;
    for( const CBodyRef& cb : cbodies ) {
        if( this != cb.get() ) {
            f3::vec_t g;
            switch( gravity_formula ) {
                case 1:  g = gravity1(*cb, *this); break;
                default: g = gravity2(*cb, *this); break;
            }
            _velo += g * dt_f;
        }
    }
    _center += _velo * dt_f;

    const fraction_timespec orbit_th(color_inverse ? 0 : 1_day);
    if( wts - _orbit_world_time_last > orbit_th ) {
        _orbit_points.emplace_back(_center.x, _center.y);
        _orbit_world_time_last = wts;
    }
}
```

Verifikation

Um die Simulationsgenauigkeit zu überprüfen, wird die Position des ausgewählten Planeten mit dem Referenzwert aus der Datenbank zu jedem Neujahr von 2015 bis 2024 verglichen. Bei erreichen des Zieldatum wird der Fehler in verschiedenen Einheiten angezeigt. Um das Zieldatum mit einer Auflösung von einer Sekunde zu erreichen, wird *time_scale* verringert, siehe Ergebnis.

Visualisierung

2D Visualisierung

Die 3D Planetenpositionen werden auf einer 2D Ebene dargestellt, sozusagen auf der Ekliptikebene. Tatsächlich ist die Z-Komponente der Planetenposition und Geschwindigkeitsvektoren im Verhältnis zum Sonnensystem vernachlässigbar.

Es wird der Orbit eines oder aller Planeten gemalt.

Das Bild kann optional mit weißem Hintergrund für Dokumente gemalt werden, siehe unten.

Maßstab Sonnensystem

Die Größenunterschiede der Planeten sind sehr groß. Würde man alles in einem Maßstab zeigen, würde man nur die Sonne sehen. Deswegen werden die Planeten in unterschiedlicher Skalierung gezeigt.

Maßstab Bildschirm

Ein Maßstab Pixel/Meter wird festgelegt, damit man Teile oder gar das ganze Sonnensystem auf dem Monitor sehen kann.

p, letzter Planet den man sehen soll
s, Distanz zur Sonne von dem Planeten p
r, Radius von p

$$Maßstab = \frac{\text{Bildschirmhöhe in Pixel}}{(s+r)*2}$$

Surrealer Spielmodus

Um die Neugierde auf das Ergebnis zu befriedigen, wenn ein schwerer Himmelskörper das Sonnensystem erreicht, kann dies optional hinzugefügt werden.

Wichtige Konstanten und Variablen

Die Konstante „light_second“(Lichtsekunde) hat den Wert 2,99792458 * 10^8 [m]. Daraus folgend haben die Konstanten „light_minute“(Lichtminute), Lichtstunde, Lichttag, …, Lichtjahr folgende Werte:

$1\,Lichtminute = 2{,}99792458 * 10^8 [m] * 60 = 1{,}798754748 * 10^{10}$
$1\,Lichtstunde = 1\,Lichtminute * 60 = 1{,}0792528488 * 10^{12}$
$1\,Lichttag = 1\,Lichtstunde * 24 = 2{,}59020683712 * 10^{13}$
$1\,Lichtwoche = 1\,Lichttag * 7 = 1{,}81314478598 * 10^{14}$
$1\,Lichtmonat\,(1\,Monat = 30\,Tage = 4{,}28571428571\,Wochen) = 1\,Lichttag * 30 = 7{,}77062051136 * 10^{14}$
$1\,Lichtjahr = 1\,Lichttag * (365 + \frac{1}{4}) = 9{,}46073047258 * 10^{15}$
$G = 6{,}6743015 * 10^{-11}$, G ist die Gravitationskonstante

Ergebnis

Simulationsgenauigkeiten der Methoden bei der Erde

Durchmesser der Erde: 12756 km

Distanz zur Sonne: 147139412 km

Länge des Orbits: 931696868 km

Fehler der Position

Einheit	Methode 1	Methode 2
km	1410275,97	1285863,30
Lichtsekunden	4,70	4,29
In Planetendurchmesser	110.56	100,80

Prozent vom Orbit	0,15%	0,14%
Prozent von der Distanz zur Sonne	0,96%	0,87%

Fehler der Geschwindigkeit

Einheit	Methode 1	Methode 2
km/s	0,29	0,28

Mögliche Gründe der Fehler

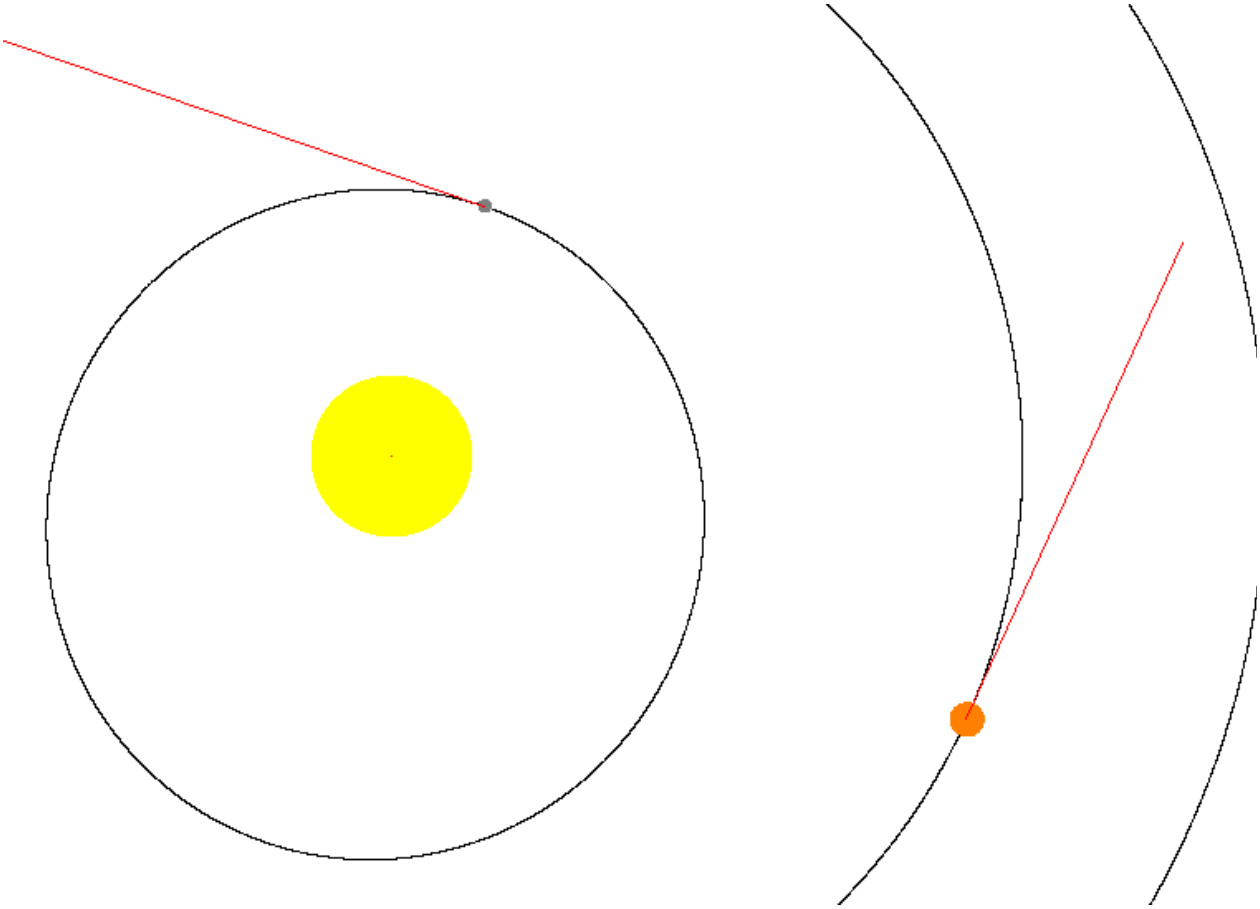
1. Masse ist ungenau
2. dt ist ungenau
3. Satelliten und die Eigenrotation der Planeten fehlt in der Simulation
4. Rotation des Sonnensystems um die Milchstraße :-)

Gemalte Bilder

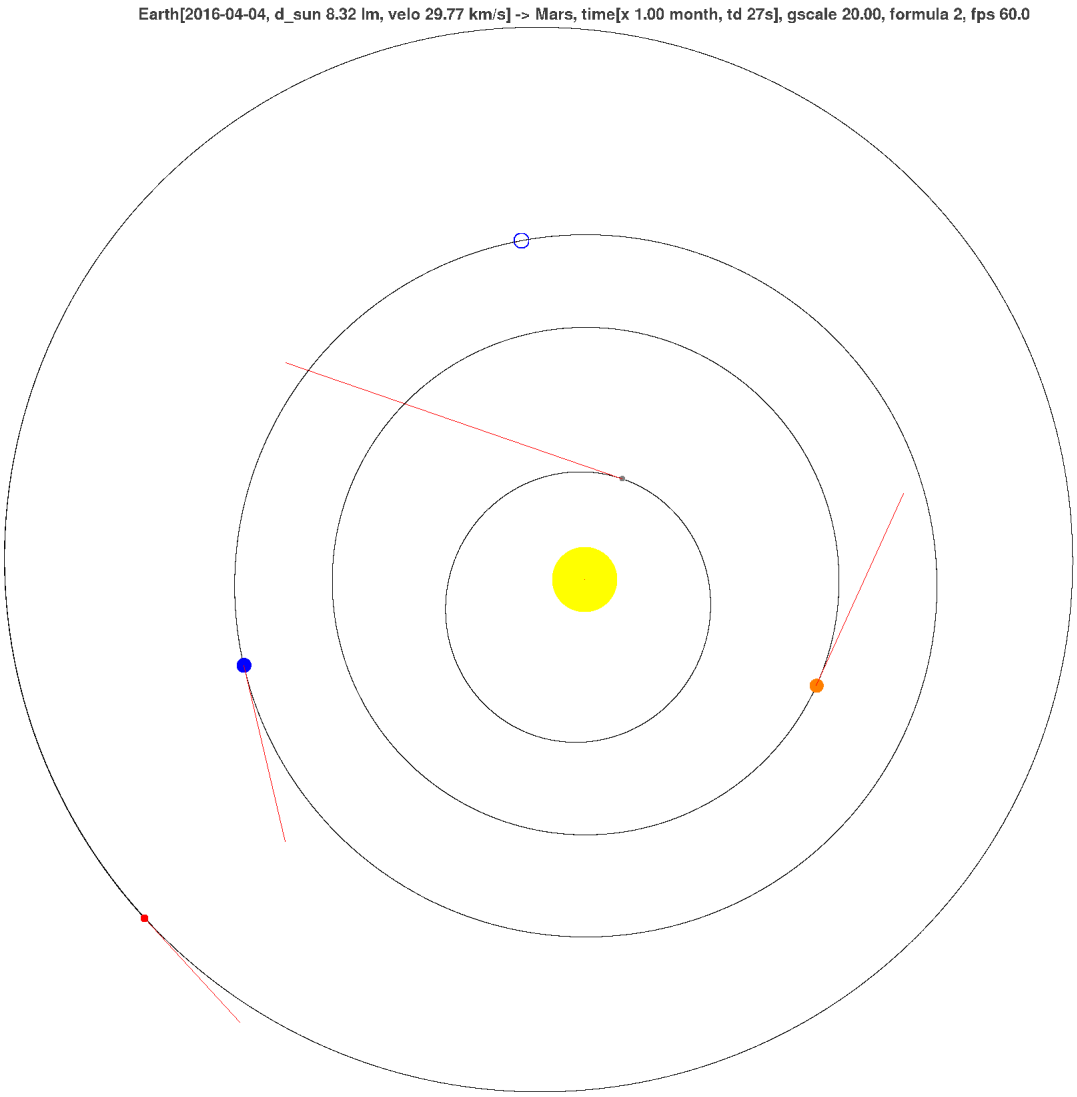
Alle Bilder werden mit dem Orbit aller Planeten gezeigt.

Diese Simulation generiert pro Planet einen elliptischen Orbit. Der initiale Geschwindigkeitsvektor ist nicht auf der Tangente zum Umkreis zur Sonne, um einen kreisförmigen Orbit zu erreichen.

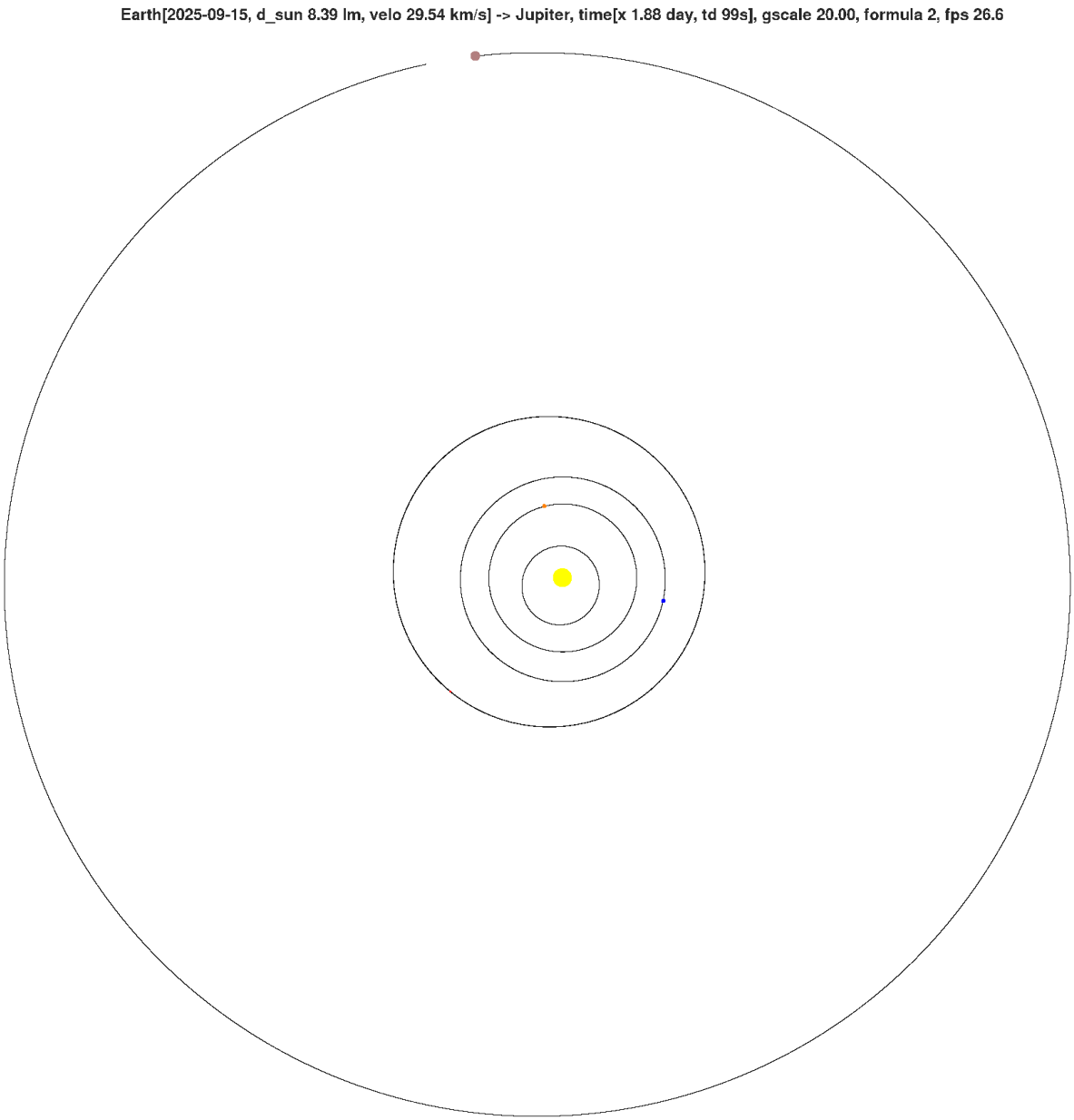
Auch ist die Sonne nicht im Zentrum aller Orbits.



Das Sonnensystems bis zum Mars



Das Sonnensystem bis zum Jupiter



Das Sonnensystem bis zum Jupiter mit schwerem Objekt

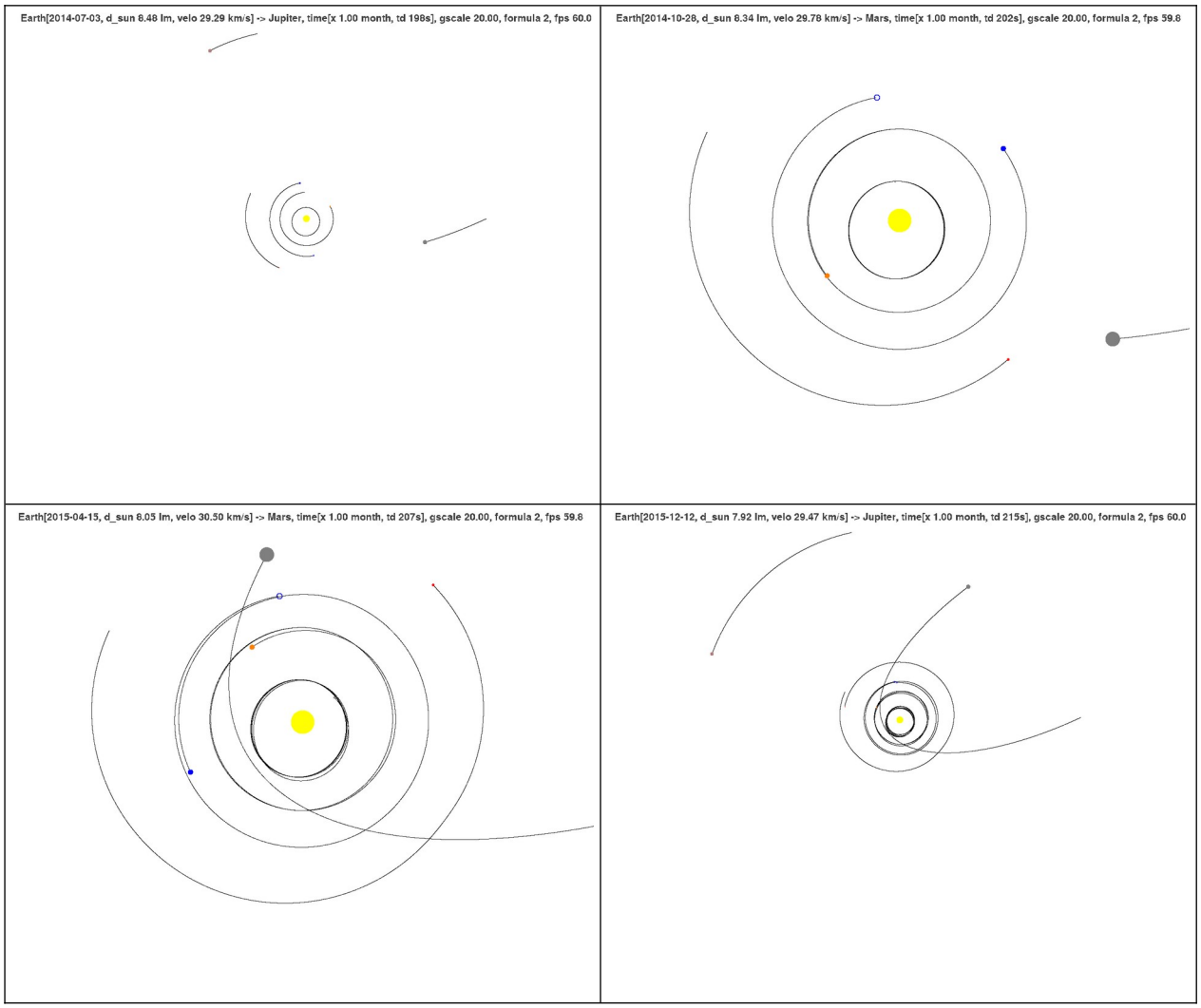
Das schwere Objekt hat …

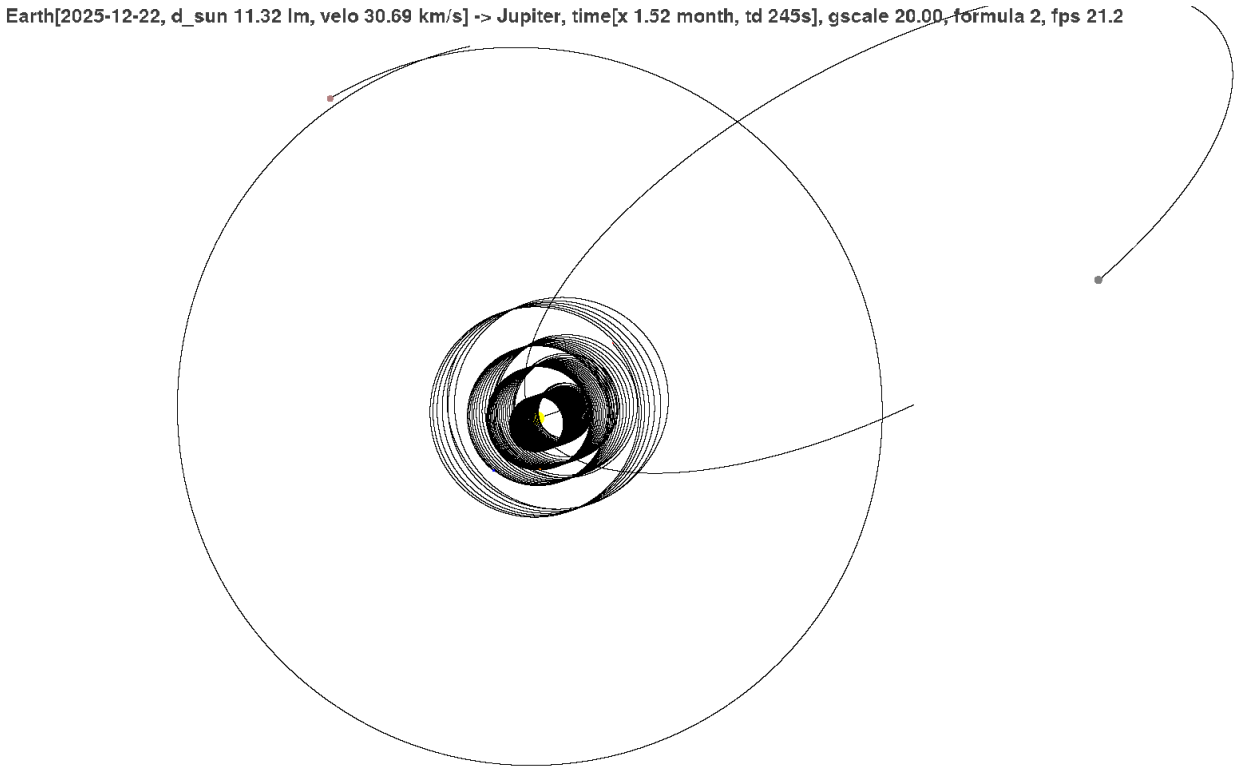
… ein GM: 1327120000 kg^3/s^2

… eine Masse: 1988400*10^{22} kg

… einen Radius: 62110 km

… eine Gravitation: 344,022 m/s^2




Earth[2025-12-22, d_sun 11.32 lm, velo 30.69 km/s] -> Jupiter, time[x 1.52 month, td 245s], gscale 20.00, formula 2, fps 21.2

Diskussion

Es gibt Verbesserungsmöglichkeiten die hereingebracht werden können. Es könnten noch die natürlichen Satelliten und die Eigenrotation der Planeten in die Simulation hinzugefügt werden.

Unterstützer

Ich habe das Programmieren bei meinem Vater, Sven Göthel, in unserem Informatikkurs gelernt. Er hat mir auch bei Programmierproblemen geholfen und mich Beraten.

Meine Eltern, Qun und Sven Göthel, haben mit mir dieses Dokument besprochen und sind mir beim Schreibstil, Struktur und wissenschaftlichen Grundlagen zur Seite gestanden.

Quellenangabe

- Planetendaten kommen aus der Horizon JPL Datenbank
- Wikipedia
    - Newtonsches Gravitationsgesetz
    - Standard gravitational parameter