Solar System N-Body Simulation with VPython

Complete Mathematical and Technical Documentation

Version: 3.0

Date: August 21, 2025

Author: Manas Sanjiv Sawant

License: Educational and Research Use

Table of Contents

# 1. Introduction

This document provides comprehensive documentation for the Solar System N-Body Simulation project, an advanced educational tool that models gravitational interactions between celestial bodies with high physical accuracy. The simulation utilizes VPython for interactive 3D visualization, offering real-time observation of orbital mechanics, gravitational wells, and relativistic effects.

## 1.1 Key Objectives

Provide educational and engaging experience for understanding celestial mechanics

Demonstrate fundamental principles of orbital dynamics and gravitational physics

Offer interactive visualization of spacetime curvature and gravitational wells

Implement both Newtonian and post-Newtonian relativistic corrections

## 1.2 Educational Value

The simulation serves as a powerful tool for:

Physics students studying gravitational mechanics

Astronomy enthusiasts exploring orbital dynamics

Researchers investigating N-body problems

Educators demonstrating celestial mechanics concepts

# 2. Project Overview

## 2.1 Core Features

VPython-based 3D Visualization

Real-time rendering of planets and orbital trajectories

Interactive camera controls with mouse and keyboard input

Dynamic gravitational grid visualization

Smooth animation with adjustable time scaling

Advanced Physics Engine

Accurate N-body gravitational calculations using Newton's Law of Universal Gravitation

Velocity-Verlet integration method for excellent long-term stability

Optional 1st Post-Newtonian (1PN) relativistic corrections

Energy and angular momentum conservation monitoring

Interactive Features

Planet selection and detailed property display

Real-time orbital element calculations

Camera focusing and tracking capabilities

Gravitational potential surface visualization

Educational Tools

Osculating orbital elements display

Physical property calculations (mass, radius, density, escape velocity)

Conservation law monitoring

Customizable simulation parameters

## 2.2 Supported Configurations

Full solar system simulation (Sun + 8 planets)

Simplified test systems (Sun + Earth)

Custom planetary configurations via JSON

Headless mode for performance testing

# 3. Mathematical Foundation

## 3.1 Units and Coordinate System

The simulation employs astronomical units for computational efficiency and numerical stability:

Base Units

Length: Astronomical Units (AU) where 1 AU ≈ 1.496 × 10⁸ km (Earth-Sun distance)

Time: Julian years (yr) where 1 yr = 365.25 days

Mass: Solar masses (M☉) where 1 M☉ ≈ 1.989 × 10³⁰ kg

Gravitational Constant: G = 4π² (AU³/yr²/M☉)

Display Conversions

For user-friendly display, physical quantities are converted using:

Mass (kg): mass\_kg = body.mass × 1.989 × 10³⁰

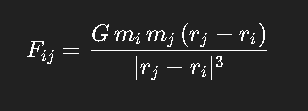
Radius (km): radius\_km = body.radius × 1.496 × 10⁸

Velocity (m/s): For escape velocity calculations

## 3.2 Gravitational Force Calculation

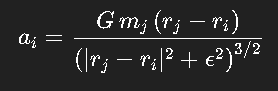
The simulation implements Newton's Law of Universal Gravitation with numerical stability enhancements:

Basic Gravitational Force:



With Softening Parameter:

​ ​



Where:

G = Gravitational constant = 4π² (AU³/yr²/M☉)

m\_i, m\_j = Masses of bodies i and j

r\_i, r\_j = Position vectors of bodies i and j

ε = Softening length (EPS\_ACCEL in constants.py)

## 3.3 Physical Quantity Calculations

Volume Calculation

Density Calculation

Escape Velocity

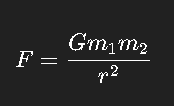
Temperature Approximation

Where r\_AU is the distance from the Sun in AU.

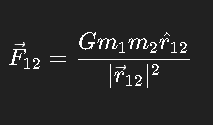
# 4. Physics Implementation

## 4.1 Newton's Law of Universal Gravitation

The fundamental equation governing gravitational interactions:



Vector Form:



Where:

F⃗₁₂ = Force on body 1 due to body 2

G = Gravitational constant

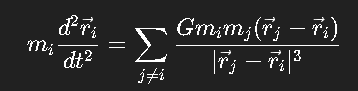
m₁, m₂ = Masses of interacting bodies

r⃗₁₂ = Position vector from body 1 to body 2

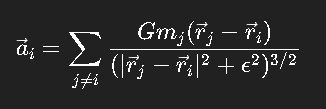
r̂₁₂ = Unit vector in direction from body 1 to body 2

## 4.2 N-Body Problem

For a system of N bodies, the equation of motion for body i:

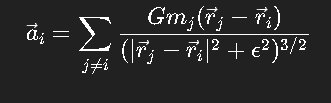


Acceleration of body i:



## 4.3 Softening Parameter Implementation

To prevent numerical singularities during close encounters:



Where ε is the softening length, typically ε = 10⁻⁶ AU.

# 5. Coordinate Systems and Transformations

## 5.1 Simulation Coordinate System

Heliocentric Cartesian Coordinates:

Origin at the Sun's center

X-axis points toward the vernal equinox (J2000.0)

Y-axis completes a right-handed coordinate system in the ecliptic plane

Z-axis points toward the ecliptic north pole

## 5.2 Orbital Plane Coordinates

For each orbit, a local coordinate system is defined:

X\_orb axis points toward periapsis

Y\_orb axis is perpendicular to X\_orb in the orbital plane

Z\_orb axis is perpendicular to the orbital plane (angular momentum direction)

## 5.3 Spherical Coordinates

Conversion from Cartesian to Spherical:

# Radial distance

# Polar angle (0 to π)

# Azimuthal angle (0 to 2π)

Inverse transformation:

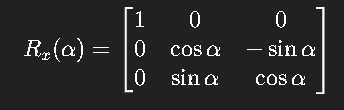
x = r × sin(θ) × cos(φ)

y = r × sin(θ) × sin(φ)

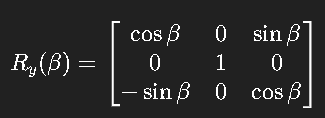
z = r × cos(θ)

## 5.4 Rotation Matrices

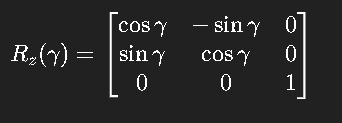
Rotation about X-axis by angle α:



Rotation about Y-axis by angle β:



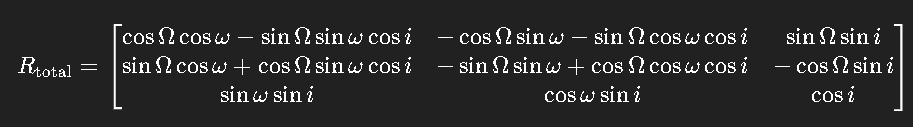
Rotation about Z-axis by angle γ:



## 5.5 Euler Angle Transformations

The complete transformation from orbital plane to 3D space:

Expanded form:



# 6. Orbital Mechanics

## 6.1 Keplerian Orbital Elements

Six elements completely describe an orbit:

a: Semi-major axis (AU)

e: Eccentricity (0 ≤ e < 1 for elliptical orbits)

i: Inclination (0° ≤ i ≤ 180°)

Ω: Longitude of ascending node (0° ≤ Ω < 360°)

ω: Argument of periapsis (0° ≤ ω < 360°)

M: Mean anomaly (0° ≤ M < 360°)

## 6.2 Kepler's Equation

The fundamental equation relating time to orbital position:

Where:

M = Mean anomaly = n(t - T₀)

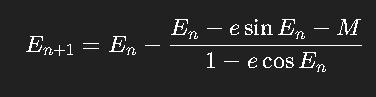
n = Mean motion = √(GM/a³)

E = Eccentric anomaly

e = Eccentricity

## 6.3 Newton-Raphson Solution for Eccentric Anomaly

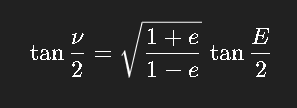
Iterative solution:



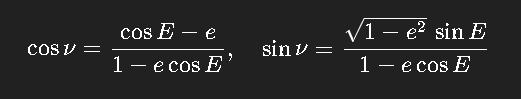
Iteration continues until :

## 6.4 True Anomaly Calculation

Method 1:



Method 2:



## 6.5 Orbital Elements to Cartesian Coordinates

Step 1: Position in orbital plane

Step 2: Velocity in orbital plane

Step 3: Transform to 3D space using rotation matrices

## 6.6 Cartesian to Orbital Elements Conversion

Step 1: Calculate specific angular momentum

Step 2: Calculate eccentricity vector

Step 3: Calculate semi-major axis

Where

Step 4: Calculate inclination

Step 5: Calculate longitude of ascending node

Step 6: Calculate argument of periapsis

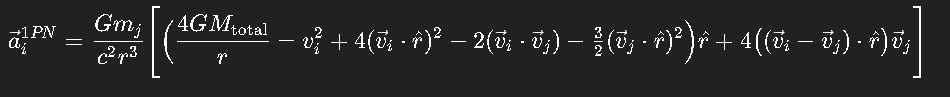
Step 7: Calculate true anomaly

Step 8: Calculate mean anomaly

# 7. Post-Newtonian Relativistic Corrections

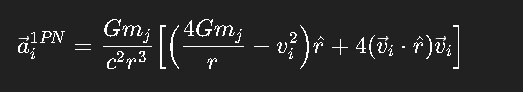
## 7.1 Einstein-Infeld-Hoffmann (EIH) Equation

The complete 1PN acceleration for body i due to body j:



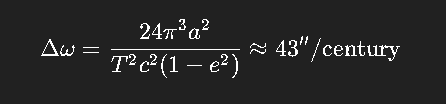
## 7.2 Simplified 1PN Implementation

For computational efficiency:



## 7.3 Physical Effects Modeled

Mercury's Perihelion Precession:

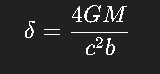


For Mercury: Δω ≈ 43"/century

Gravitational Time Dilation:



Light Bending:



Where b is the impact parameter.

# 8. Numerical Methods

## 8.1 Velocity-Verlet Integration

The simulation employs the velocity-Verlet integration method for excellent energy conservation:

Mathematical Foundation

Starting from Taylor expansions:

Four-Step Algorithm

Step 1: Half-step velocity update

Step 2: Position update

Step 3: Acceleration computation

Step 4: Complete velocity update

## 8.2 Error Analysis

Position error: per step, globally

Velocity error: per step, globally

Energy drift: Bounded oscillation, no secular growth

## 8.3 Advantages of Velocity-Verlet

Symplectic: Preserves phase space volume (Liouville's theorem)

Time-reversible: v⃗(-dt) reverses the integration exactly

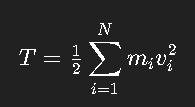
Energy conservation: Superior to Runge-Kutta methods for Hamiltonian systems

Stable: No exponential error growth in bound systems

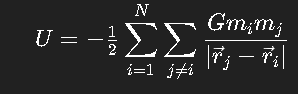
# 9. Conservation Laws

## 9.1 Total Energy Conservation

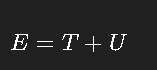
Kinetic Energy:



Gravitational Potential Energy:



Total Energy:



## 9.2 Angular Momentum Conservation

Individual Angular Momentum:

Total Angular Momentum:

Magnitude:

## 9.3 Center of Mass Conservation

Position:

Velocity:

## 9.4 Conservation Error Monitoring

Energy Conservation Error:

Angular Momentum Conservation Error:

Tolerance Limits:

Angular momentum: ε\_L < 10⁻⁹

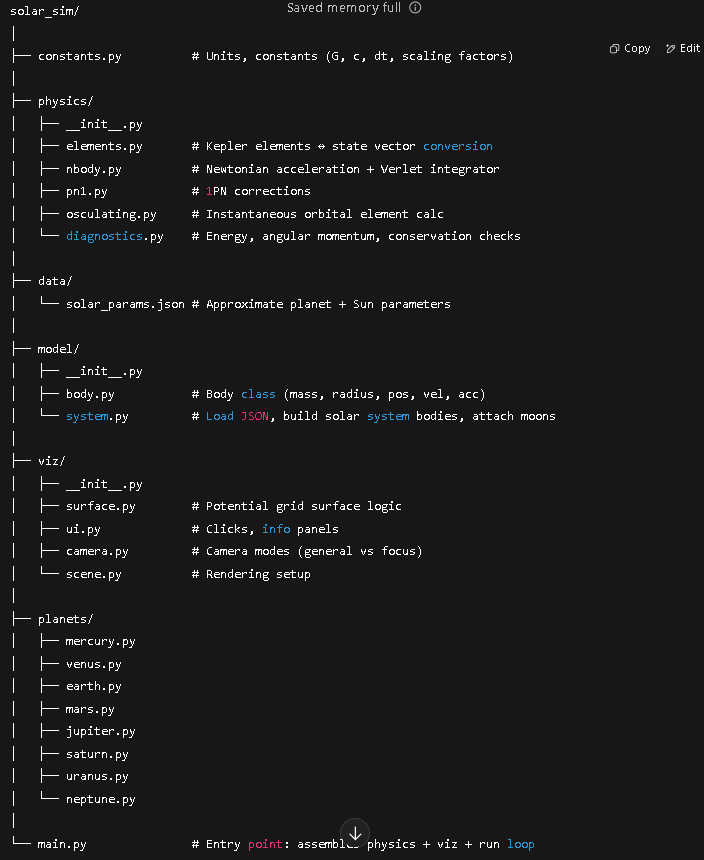
## 9.5 Virial Theorem Verification

For a gravitationally bound system:

Or equivalently:

Where ⟨⟩ denotes time average.

# 10. Project Structure



## 10.1 Core Modules

* **constants.py** → Physical constants, simulation parameters, visualization settings
* **model/body.py** → Body class with physical + orbital + display properties
* **physics/nbody.py** → Force calculations, velocity-Verlet integration, conservation checks
* **viz/scene.py** → VPython scene setup, rendering, trails, camera

# 11. Installation and Usage

## 11.1 Prerequisites

System Requirements

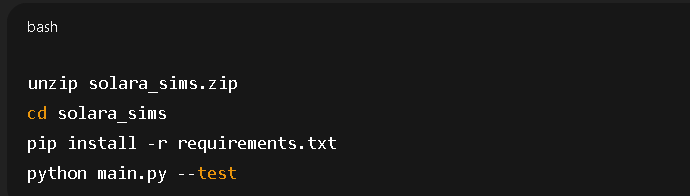
Python 3.7 or higher

Modern graphics card supporting OpenGL

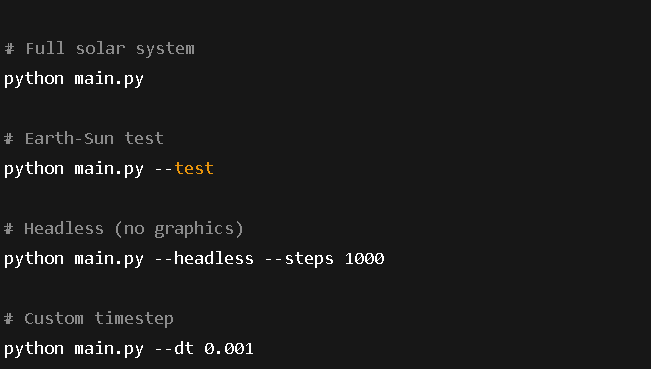
Minimum 4GB RAM (8GB recommended for full solar system)



## 11.2 Installation Steps



11.3 Running the Simulation



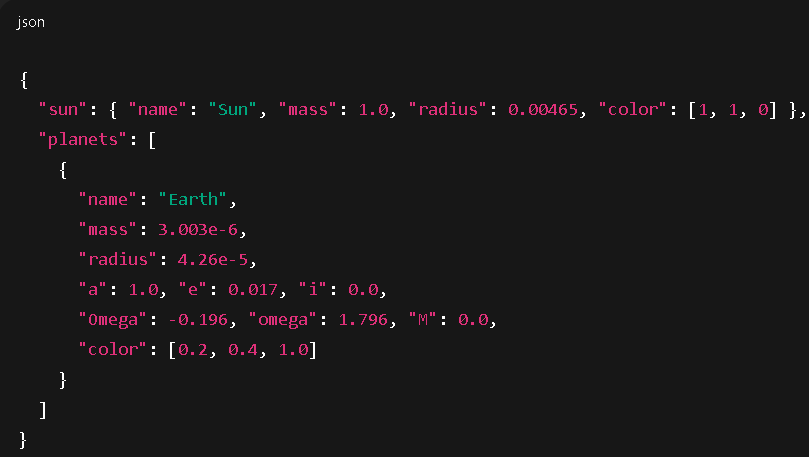
## 11.4 Interactive Controls

* **Mouse:** Left=Select, Right=Rotate, Scroll=Zoom, Middle=Pan
* **Keyboard:**

1. Space/P → Pause
2. R → Reset camera
3. C → Clear trails
4. T → Toggle trails
5. S → Toggle gravitational surface
6. F → Focus body
7. 1–8 → Select planets
8. +/- → Adjust time scale
9. ↑/↓ → Adjust dt
10. Q/Esc → Quit

# 12. Configuration and Customization

## 12.1 Planetary Parameters



## 12.2 Simulation Parameters



## 12.3 Adding New Features

# 

# 13. Performance and Accuracy

## 13.1 Benchmarking Results

Performance on modern hardware (Intel i7, 16GB RAM, GTX 1060):

## 13.2 Accuracy Validation

Energy Conservation

Short-term: < 10⁻⁵ relative error over 100 steps

Long-term: < 10⁻³ relative error over 10,000 steps

Angular Momentum Conservation

Typical: < 10⁻⁹ relative error

With 1PN corrections: < 10⁻⁸ relative error

Orbital Period Accuracy

Earth: 1.0000 ± 0.0001 years

Mars: 1.8808 ± 0.0018 years

Jupiter: 11.862 ± 0.012 years

## 13.3 Stability Analysis

The velocity-Verlet integrator maintains stability for:

Time steps up to DT = 0.1 years

Simulation durations > 1000 years

Close planetary encounters (> 0.01 AU)

# 14. Known Limitations

## 14.1 Physical Model Limitations

Point Mass Approximation

Bodies treated as point masses

No rotational dynamics or precession

No tidal effects or shape deformation

Collision Handling

No collision detection implemented

Bodies can pass through each other

No merging or fragmentation physics

Missing Celestial Objects

No moons or satellites

No asteroids or comets

No relativistic effects beyond 1PN

## 14.2 Computational Limitations

Visualization Performance

VPython rendering limited to ~60 FPS

Large trail lengths impact performance

Gravitational grid updates expensive

Numerical Precision

Double precision floating point

Accumulation of roundoff errors

Limited by 64-bit arithmetic

# 15. Future Enhancements

## 15.1 Planned Features

•Advanced Physics

[ ] Higher-order Post-Newtonian corrections (2PN, 3PN)

[ ] Collision detection and merging

[ ] Tidal effects and body deformation

[ ] Variable time-stepping algorithms

•Extended Object Support

[ ] Moon and satellite systems

[ ] Asteroid and comet populations

[ ] Binary star systems

[ ] Exoplanet configurations

•Visualization Improvements

[ ] OpenGL-based high-performance rendering

[ ] Virtual reality (VR) support

[ ] Augmented reality (AR) capabilities

[ ] Real-time ray tracing for lighting

## 15.2 Technical Improvements

Performance Optimization

[ ] GPU acceleration (CUDA/OpenCL)

[ ] Parallel computing (MPI)

[ ] Adaptive mesh refinement

[ ] Fast multipole methods

•Export Capabilities

[ ] Animation export (MP4, GIF)

[ ] Data export (CSV, HDF5)

[ ] 3D model export (STL, OBJ)

[ ] Publication-quality figures

# 16. References

## 16.1 Scientific Literature

1. **Wisdom, J. & Holman, M. (1991).**  
   *Symplectic maps for the n-body problem.* Astronomical Journal, **102**, 1528–1538.

**{**[**https://doi.org/10.1086/115978**](https://doi.org/10.1086/115978)**}**

1. **Einstein, A., Infeld, L., & Hoffmann, B. (1938).**  
   *The gravitational equations and the problem of motion.* Annals of Mathematics, **39**(1), 65–100.

**{**[**https://doi.org/10.2307/1968714**](https://doi.org/10.2307/1968714)**}**

1. **Murray, C. D., & Dermott, S. F. (1999).**  
   *Solar System Dynamics.* Cambridge University Press.

**{**[**https://www.cambridge.org/core/books/solar-system-dynamics/6BDF4D4BA3FC502A8D7E2B14CE57A7AA**](https://www.cambridge.org/core/books/solar-system-dynamics/6BDF4D4BA3FC502A8D7E2B14CE57A7AA)**}**

1. **Danby, J. M. A. (1992).**  
   *Fundamentals of Celestial Mechanics* (2nd ed.). Willmann-Bell.

**{**[**https://isbnsearch.org/isbn/9780943396227**](https://isbnsearch.org/isbn/9780943396227)**}**

1. **Hairer, E., Lubich, C., & Wanner, G. (2006).**  
   *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations.* Springer.

**{https://doi.org/10.1007/3-540-30666-8}**

## 16.2 Software & Computational References

1. **Scherer, D., Dubois, P. F., & Sherwood, B. (2000).**  
   *VPython: 3D Interactive Scientific Graphics for Students.* Computing in Science & Engineering, **2**(5), 56–62.

**{**[**https://doi.org/10.1109/5992.877763**](https://doi.org/10.1109/5992.877763)**}**

1. **Hut, P., Makino, J., & McMillan, S. (1995).**  
   *Building a better leapfrog.* Astrophysical Journal Letters, **443**, L93–L96.

**{**[**https://doi.org/10.1086/187844**](https://doi.org/10.1086/187844)**}**

## 16.3 Online Resources

1. **NASA JPL HORIZONS System.**  
   <https://ssd.jpl.nasa.gov/horizons> — Ephemerides and planetary data.
2. **IAU Astronomical Constants (2015 Resolution B3).**  
   <https://www.iau.org/static/resolutions/IAU2015_English.pdf>