CelestialSim

A gravitational N-body simulator implemented in C#, modeling celestial mechanics using Newtonian gravity with numerical integration.

1 Numerical Methods

Let $\Psi(o,t)$ denote the exact state vector of object o at time t, defined as:

$$\Psi(o,t) = \begin{bmatrix} \vec{r}(t) \\ \vec{v}(t) \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \\ \dot{x}(t) \\ \dot{y}(t) \end{bmatrix},$$

where $\vec{r}(t)$ is position and $\vec{v}(t)$ is velocity. The system evolves according to the dynamics function φ:

$$\frac{d}{dt}\Psi(o,t) = \varphi(\Psi(o,t), \mathcal{S}(t)),$$

where $\mathcal{S}(t)$ represents the global state at time t.

1.1 Dynamics Function

The gravitational dynamics are given by:

$$\varphi(\Psi(o_i,t), \mathcal{S}(t)) = \begin{bmatrix} \vec{v}_i(t) \\ G\sum_{o_j \neq o_i} \frac{m_j\left(\vec{r}_j(t) - \vec{r}_i(t)\right)}{\|\vec{r}_j(t) - \vec{r}_i(t)\|^3} \end{bmatrix},$$

where:

- $\mathcal{O}$ = set of all celestial bodies
- G = gravitational constant
- m_j = mass of body o_j
- $\vec{r}_i, \vec{r}_j$ = positions of bodies o_i and o_j

1.2 Euler's Method

This is a simple first-order approximation. We treat the evolution as linear over a small time step Δt:

$$\Psi(o, t + \Delta t) \approx \Psi(o,t) + \Delta t \cdot \varphi(\Psi(o,t), \mathcal{S}(t)).$$

1.3 2nd Order Runge-Kutta

As the name says, this is a second-order approximation. It consists of two steps. We first compute the midpoint state between t and Δt:

$$\Psi_{\text{mid}} = \Psi(o,t) + \frac{\Delta t}{2} \cdot \varphi(\Psi(o,t), \mathcal{S}(t)).$$

Then we update using the midpoint derivative:

$$\Psi(o, t + \Delta t) \approx \Psi(o,t) + \Delta t \cdot \varphi(\Psi_{\text{mid}}, \mathcal{S}(t + \Delta t/2)).$$

2 Simulation Parameters

Time step. Δt is the time step. A smaller time step yields more precise results, but more computational power is required.

Gravitational Constant. G is the universal gravitational constant.

Fast-Forward. Δ represents a "fast-forward" mechanism. Because the simulation is in real time, it might take a long time to observe a significant change. In that case, Δ is added to the total elapsed time since last calculation ϵ, which in turn increases the number of calculation steps n:

$$n = \left\lfloor \frac{\Delta + \epsilon}{\Delta t} \right\rfloor.$$

When you use fast-forwarding, the simulation calculates all the skipped steps while keeping the same small time step (Δt) for accuracy.

3 Controls

- Scroll: Zoom
- Left click + drag: Pan view
- Left click a body: Select
- Middle click: Add new body