Package 'orbitr'

May 3, 2026

Type Package

Title A Tidy Physics Engine for Building and Visualizing Orbital Simulations

Version 0.3.0

Description A lightweight, fully vectorized N-body physics engine built for the R ecosystem. Simulate and visualize complex orbital mechanics, celestial trajectories, and gravitational interactions using tidy data principles. Features multiple numerical integration methods, including the energy-conserving velocity Verlet algorithm (Verlet (1967) <doi:10.1103/PhysRev.159.98>), to ensure highly stable orbital propagation. Gravitational N-body methods follow Aarseth (2003, ISBN:0-521-43272-3).

URL https://orbit-r.com/, https://github.com/DRosenman/orbitr

BugReports https://github.com/DRosenman/orbitr/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.3

Imports dplyr, ggplot2, Rcpp, tibble

Suggests plotly, gganimate, gifski, magick, knitr, rmarkdown, pkgdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

LinkingTo Rcpp

NeedsCompilation yes

Author Dave Rosenman [aut, cre]

Maintainer Dave Rosenman <dave.rosenman.data@gmail.com>

Depends R (>= 4.1.0)

Repository CRAN

Date/Publication 2026-05-03 06:30:02 UTC

Contents

`add_body` *Add a physical body to the system*

Description

This function introduces a new celestial or physical body into your ‘orbit_system‘. You must provide a unique identifier and its mass. By default, the body will be placed at the origin (0, 0, 0) with zero initial velocity unless specified.

Usage

```
add_body(system, id, mass, x = 0, y = 0, z = 0, vx = 0, vy = 0, vz = 0)
```

Arguments

`system`	An ‘orbit_system‘ object created by ‘create_system()‘.
`id`	A unique character string to identify the body (e.g., "Earth", "Apollo").
`mass`	The mass of the object in kilograms.
`x`	Initial X-axis position in meters (default 0).
`y`	Initial Y-axis position in meters (default 0).
`z`	Initial Z-axis position in meters (default 0).

vx	Initial velocity along the X-axis in meters per second (default 0).
vy	Initial velocity along the Y-axis in meters per second (default 0).
vz	Initial velocity along the Z-axis in meters per second (default 0).

Value

The updated ‘orbit_system‘ object containing the newly added body.

Examples

```
my_universe <- create_system() |>
  add_body(id = "Earth", mass = 5.97e24) |>
  add_body(id = "Moon", mass = 7.34e22, x = 3.84e8, vy = 1022)
```

---

add_body_keplerian *Add a body using Keplerian orbital elements*

Description

A convenience wrapper around [add_body()] that lets you specify an orbit using classical Keplerian elements instead of raw Cartesian state vectors. The elements are converted to position and velocity in the reference frame of the parent body, which must already exist in the system.

Usage

```
add_body_keplerian(
  system,
  id,
  mass,
  a,
  e = 0,
  i = 0,
  lan = 0,
  arg_pe = 0,
  nu = 0,
  parent
)
```

Arguments

system	An ‘orbit_system‘ object created by [create_system()].
id	A unique character string to identify the body.
mass	The mass of the body in kilograms.
a	Semi-major axis in meters.
e	Eccentricity (0 = circle, 0 < e < 1 = ellipse). Default 0.

i	Inclination in degrees. Default 0.
lan	Longitude of ascending node in degrees. Default 0.
arg_pe	Argument of periapsis in degrees. Default 0.
nu	True anomaly in degrees. Default 0 (body starts at periapsis).
parent	Character id of the parent body (must already exist in ‘system‘). The orbital elements are defined relative to this body.

Value

The updated ‘orbit_system‘ with the new body added.

Keplerian Elements

Six numbers fully describe a Keplerian orbit:

‘a‘ (semi-major axis) The size of the orbit — half the longest diameter of the ellipse, in meters.

‘e‘ (eccentricity) The shape of the orbit. 0 is a perfect circle; values between 0 and 1 are ellipses.

‘i‘ (inclination) The tilt of the orbital plane relative to the reference plane, in degrees.

‘lan‘ (longitude of ascending node) The angle from the reference direction to where the orbit crosses the reference plane going "upward," in degrees. Sometimes written as Ω.

‘arg_pe‘ (argument of periapsis) The angle within the orbital plane from the ascending node to the closest-approach point, in degrees. Sometimes written as ω.

‘nu‘ (true anomaly) Where the body currently sits along its orbit, measured as an angle from periapsis in degrees. 0 = at periapsis (closest), 180 = at apoapsis (farthest).

Examples

```
# Earth orbiting the Sun with real orbital elements
system <- create_system() |>
  add_sun() |>
  add_body_keplerian(
    "Earth", mass = mass_earth,
    a = distance_earth_sun, e = 0.0167, i = 0.00005,
    parent = "Sun"
  )

# Mars with its notable eccentricity
system <- system |>
  add_body_keplerian(
    "Mars", mass = mass_mars,
    a = distance_mars_sun, e = 0.0934, i = 1.85,
    lan = 49.6, arg_pe = 286.5, nu = 0,
    parent = "Sun"
  )
```

add_planet *Add a known solar system body by name*

Description

A convenience wrapper around [add_body_keplerian()] that looks up real orbital elements for well-known solar system bodies. Instead of typing out mass, semi-major axis, eccentricity, and inclination by hand, just give the name and parent:

Usage

```
add_planet(
  system,
  name,
  parent,
  nu = 0,
  a = NULL,
  e = NULL,
  i = NULL,
  lan = NULL,
  arg_pe = NULL,
  mass = NULL
)
```

Arguments

system	An ‘orbit_system‘ object.
name	The name of the body. Must be one of: ‘"Mercury"‘, ‘"Venus"‘, ‘"Earth"‘, ‘"Mars"‘, ‘"Jupiter"‘, ‘"Saturn"‘, ‘"Uranus"‘, ‘"Neptune"‘, ‘"Moon"‘, or ‘"Pluto"‘. Case-sensitive.
parent	Character id of the parent body, which must already exist in the system. For planets and Pluto this is typically ‘"Sun"‘; for the Moon it is ‘"Earth"‘.
nu	True anomaly in degrees (default 0, body starts at periapsis). This is the most commonly overridden element — use it to spread planets around their orbits instead of starting them all at periapsis.
a	Override semi-major axis (meters).
e	Override eccentricity.
i	Override inclination (degrees).
lan	Override longitude of ascending node (degrees).
arg_pe	Override argument of periapsis (degrees).
mass	Override mass (kg).

Details

“‘ create_system() |> add_sun() |> add_planet("Earth", parent = "Sun") |> add_planet("Moon", parent = "Earth") “‘

Any Keplerian element can be overridden to explore "what if" scenarios while keeping the rest of the real values:

“‘ # What if Mars had zero eccentricity? add_planet("Mars", parent = "Sun", e = 0) “‘

Value

The updated ‘orbit_system‘ with the body added.

Examples

```
# Build the inner solar system
create_system() |>
  add_sun() |>
  add_planet("Mercury", parent = "Sun") |>
  add_planet("Venus",   parent = "Sun") |>
  add_planet("Earth",   parent = "Sun") |>
  add_planet("Mars",    parent = "Sun") |>
  simulate_system(time_step = seconds_per_day, duration = seconds_per_year) |>
  plot_orbits()

# Earth-Moon system
create_system() |>
  add_body("Earth", mass = mass_earth) |>
  add_planet("Moon", parent = "Earth") |>
  simulate_system(time_step = seconds_per_hour, duration = seconds_per_day * 28) |>
  plot_orbits()

# What if Jupiter were twice as massive?
create_system() |>
  add_sun() |>
  add_planet("Jupiter", parent = "Sun", mass = mass_jupiter * 2)
```

---

add_sun *Add the Sun to the system*

Description

A convenience function that adds the Sun as the central body of a simulation. By default it is placed at the origin with zero velocity, which is the natural choice for a heliocentric reference frame. Position and velocity can be overridden for advanced use cases such as barycentric coordinates.

Usage

```
add_sun(system, mass = mass_sun, x = 0, y = 0, z = 0, vx = 0, vy = 0, vz = 0)
```

Arguments

system	An ‘orbit_system‘ object created by [create_system()].
mass	Mass of the Sun in kilograms. Defaults to [mass_sun] (1.989 x 10^30 kg).
x	Initial X-axis position in meters (default 0).
y	Initial Y-axis position in meters (default 0).
z	Initial Z-axis position in meters (default 0).
vx	Initial velocity along the X-axis in m/s (default 0).
vy	Initial velocity along the Y-axis in m/s (default 0).
vz	Initial velocity along the Z-axis in m/s (default 0).

Details

This pairs naturally with [add_planet()]:

“‘ create_system() |> add_sun() |> add_planet("Earth", parent = "Sun") |> add_planet("Mars", parent = "Sun") “‘

Value

The updated ‘orbit_system‘ with the Sun added.

Examples

```
# Typical usage — Sun at the origin
create_system() |>
  add_sun()


# Full solar system in three lines
create_system() |>
  add_sun() |>
  add_planet("Earth", parent = "Sun") |>
  add_planet("Mars",  parent = "Sun") |>
  simulate_system(time_step = seconds_per_day, duration = seconds_per_year) |>
  plot_orbits()
```

---

animate_system *Animate the System Over Time (Smart 2D/3D Dispatch)*

Description

Plays the simulation forward as an animation. Bodies move through their orbits frame by frame, optionally leaving a fading wake behind them. This is the animated counterpart to [plot_system()] — a moving snapshot rather than a single frozen one.

Usage

```
animate_system(
  sim_data,
  fps = 20,
  duration = 10,
  trails = FALSE,
  three_d = NULL
)
```

Arguments

sim_data	A tibble output from [simulate_system()].
fps	Frames per second of the rendered animation. Default ‘20‘.
duration	Length of the animation in seconds. Default ‘10‘. Together with ‘fps‘, this determines how many simulation time steps are sampled into frames (‘fps * duration‘). If your simulation has fewer steps than that, every step becomes a frame.
trails	Logical. If ‘TRUE‘ (the default), each body leaves a fading wake of its recent positions behind it. Set ‘FALSE‘ for naked moving dots.
three_d	Logical. If ‘TRUE‘, forces a 3D animation even for planar data.

Details

If any body has non-zero motion in the Z dimension (or ‘three_d = TRUE‘), [animate_system_3d()] is used; otherwise a 2D ‘gganimate‘ animation is returned.

The 2D path requires the ‘gganimate‘ package, which is in ‘Suggests‘. Install it with ‘install.packages("gganimate")‘. Rendering a 2D animation is much slower than a static plot — expect tens of seconds for typical simulations, since every frame is drawn and encoded as a GIF (or MP4).

The 3D path uses ‘plotly‘’s built-in ‘frame‘ aesthetic, which produces an interactive HTML widget with a play button and time slider. No GIF encoding is involved, so 3D animations render essentially instantly.

Value

A rendered ‘gganimate‘ animation (2D) or a ‘plotly‘ HTML widget with built-in play/pause controls (3D). The 2D return value can be saved to disk with [gganimate::anim_save()].

Examples

```
sim <- create_system() |>
  add_sun() |>
  add_body("Earth", mass = mass_earth, x = distance_earth_sun, vy = speed_earth) |>
  simulate_system(time_step = seconds_per_day, duration = seconds_per_year)

# 2D fading-wake animation (requires gganimate)
anim <- animate_system(sim, fps = 20, duration = 8)
anim
```

```
# Save to disk
gganimate::anim_save(file.path(tempdir(), "earth_orbit.gif"), anim)
```

`animate_system_3d` *Animate the System Over Time in Interactive 3D*

Description

The 3D counterpart to [animate_system()]. Builds a ‘plotly‘ 3D scene with the bodies as moving markers and an interactive Play / Pause control plus a time slider. Optionally shows the full orbit paths drawn faintly behind.

Usage

```
animate_system_3d(sim_data, fps = 20, duration = 10, trails = FALSE)
```

Arguments

`sim_data`	A tibble output from [simulate_system()].
`fps`	Frames per second target for playback. Default ‘20‘. Combined with ‘duration‘, controls how many time steps are sampled into frames.
`duration`	Total playback length in seconds. Default ‘10‘.
`trails`	Logical. If ‘TRUE‘ (the default), the full orbit paths are drawn faintly behind the animated markers.

Value

A ‘plotly‘ HTML widget with a built-in play button and time slider.

Examples

```
create_system() |>
  add_body("Earth", mass = mass_earth) |>
  add_body("Moon",  mass = mass_moon,
           x = distance_earth_moon, vy = speed_moon, vz = 100) |>
  simulate_system(time_step = seconds_per_hour, duration = seconds_per_day * 30) |>
  animate_system_3d()
```

`create_system` *Initialize an orbitr simulation system*

Description

Sets up the foundational data structure for an orbital simulation. Universal N-body gravity is automatically integrated into the system using the specified gravitational constant.

Usage

```
create_system(G = gravitational_constant)
```

Arguments

`G`	The gravitational constant. Defaults to the real-world value (‘gravitational_constant‘, 6.6743e-11 m^3 kg^-1 s^-2). To simulate a zero-gravity environment (inertia only), set ‘G = 0‘.

Value

An empty ‘orbit_system‘ object ready for bodies to be added.

Examples

```
# Creates a system with standard gravity
my_universe <- create_system()

# Creates a universe with 10x stronger gravity
heavy_universe <- create_system(G = gravitational_constant * 10)

# Creates a zero-gravity sandbox
floating_universe <- create_system(G = 0)
```

`export_bodies` *Export body states to CSV*

Description

Writes the body table (id, mass, position, and velocity) from an ‘orbit_system‘ to a CSV file. This is useful for sharing initial conditions with collaborators or loading them into other tools like Python or Excel.

Usage

```
export_bodies(system, path)
```

Arguments

system	An ‘orbit_system‘ object.
path	File path to save to. Should end in ‘.csv‘.

Value

‘system‘, invisibly.

Examples

```
sys <- create_system() |>
  add_sun() |>
  add_planet("Earth", parent = "Sun") |>
  add_planet("Mars",  parent = "Sun")

export_bodies(sys, file.path(tempdir(), "bodies.csv"))
```

---

get_bodies *Extract the body table from a system*

Description

Returns the bodies in an ‘orbit_system‘ as a standalone tibble. Useful when you want to inspect, filter, or save the body states without dealing with the full system object.

Usage

```
get_bodies(system)
```

Arguments

system	An ‘orbit_system‘ object.

Value

A tibble with columns ‘id‘, ‘mass‘, ‘x‘, ‘y‘, ‘z‘, ‘vx‘, ‘vy‘, ‘vz‘.

Examples

```
sys <- create_system() |>
  add_sun() |>
  add_planet("Earth", parent = "Sun") |>
  add_planet("Mars",  parent = "Sun")

# Get the tibble
get_bodies(sys)
```

```
# Use with dplyr

get_bodies(sys) |>
  dplyr::filter(mass > 1e24)
```

`load_solar_system` *Load a pre-built solar system*

Description

A convenience function that creates a complete solar system with the Sun and all eight planets (plus optionally the Moon and Pluto) using real orbital data. Bodies are placed using Keplerian orbital elements from the JPL DE440 ephemeris (J2000 epoch), giving realistic eccentricities, inclinations, and orbital orientations out of the box.

Usage

```
load_solar_system(moon = TRUE, pluto = TRUE)
```

Arguments

`moon`	Logical. If ‘TRUE‘ (the default), include the Moon in orbit around Earth.
`pluto`	Logical. If ‘TRUE‘ (the default), include Pluto.

Details

This is a quick way to get a physically accurate starting point without typing out a dozen [add_body()] calls. The returned system is a normal ‘orbit_system‘ that you can modify further — add bodies, change parameters, or pipe straight into [simulate_system()].

Value

An ‘orbit_system‘ object containing the Sun and planets, ready for simulation.

Examples

```
# Simulate the full solar system for one year
solar <- load_solar_system() |>
  simulate_system(
    time_step = seconds_per_day,
    duration  = seconds_per_year
  )

plot_orbits(solar)

# Just the Sun and planets, no Moon or Pluto
load_solar_system(moon = FALSE, pluto = FALSE)
```

load_system *Load an orbit_system from disk*

Description

Restores an ‘orbit_system‘ previously saved with [save_system()].

Usage

```
load_system(path)
```

Arguments

path	File path to an ‘.rds‘ file created by [save_system()].

Value

An ‘orbit_system‘ object.

Examples

```
sys <- create_system() |>
  add_sun() |>
  add_planet("Earth", parent = "Sun")

path <- file.path(tempdir(), "my_system.rds")
save_system(sys, path)
restored <- load_system(path)
```

physical_constants *Physical Constants for Orbital Mechanics*

Description

A curated set of real-world masses and orbital distances for use as convenient starting points in ‘orbitr‘ simulations. All values are in SI units (kilograms and meters).

Usage

```
gravitational_constant

seconds_per_hour

seconds_per_day
```

seconds_per_year

mass_sun

mass_earth

mass_moon

mass_mars

mass_jupiter

mass_saturn

mass_venus

mass_mercury

mass_uranus

mass_neptune

mass_pluto

distance_earth_sun

distance_earth_moon

distance_mars_sun

distance_jupiter_sun

distance_venus_sun

distance_mercury_sun

distance_saturn_sun

distance_uranus_sun

distance_neptune_sun

distance_pluto_sun

speed_earth

speed_moon

`speed_mars`

`speed_jupiter`

`speed_venus`

`speed_mercury`

`speed_saturn`

`speed_uranus`

`speed_neptune`

`speed_pluto`

Format

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

Numeric scalar in kilograms.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

An object of class `numeric` of length 1.

Details

‘gravitational_constant‘: Newton’s gravitational constant (6.6743 x 10^-11 m^3 kg^-1 s^-2). Source: CODATA 2018 recommended value. Use this with ‘create_system()‘ to scale gravity: ‘create_system(G = gravitational_constant * 10)‘.

‘seconds_per_hour‘: 3,600 seconds. Convenient for setting ‘time_step‘ in lunar or close-orbit simulations.

‘seconds_per_day‘: 86,400 seconds. Convenient for setting ‘time_step‘ in planetary-scale simulations.

‘seconds_per_year‘: 31,557,600 seconds (365.25 days, the Julian year). Convenient for setting ‘duration‘ in ‘simulate_system()‘.

‘mass_sun‘: Mass of the Sun (1.989 x 10^30 kg). Source: IAU 2015 nominal solar mass.

‘mass_earth‘: Mass of the Earth (5.972 x 10^24 kg). Source: IAU 2015 nominal Earth mass.

‘mass_moon‘: Mass of the Moon (7.342 x 10^22 kg). Source: JPL DE440 ephemeris.

‘mass_mars‘: Mass of Mars (6.417 x 10^23 kg). Source: JPL DE440 ephemeris.

‘mass_jupiter‘: Mass of Jupiter (1.898 x 10^27 kg). Source: JPL DE440 ephemeris.

‘mass_saturn‘: Mass of Saturn (5.683 x 10^26 kg). Source: JPL DE440 ephemeris.

‘mass_venus‘: Mass of Venus (4.867 x 10^24 kg). Source: JPL DE440 ephemeris.

‘mass_mercury‘: Mass of Mercury (3.301 x 10^23 kg). Source: JPL DE440 ephemeris.

‘mass_uranus‘: Mass of Uranus (8.681 x 10^25 kg). Source: JPL DE440 ephemeris.

‘mass_neptune‘: Mass of Neptune (1.024 x 10^26 kg). Source: JPL DE440 ephemeris.

‘mass_pluto‘: Mass of Pluto (1.309 x 10^22 kg). Source: JPL DE440 ephemeris. Pluto is a dwarf planet but is included for convenience.

‘distance_earth_sun‘: Semi-major axis of Earth’s orbit around the Sun (1.496 x 10^11 m, ~149.6 million km). Earth’s actual distance varies between ~147.1 million km (perihelion) and ~152.1 million km (aphelion).

‘distance_earth_moon‘: Semi-major axis of the Moon’s orbit around Earth (3.844 x 10^8 m, ~384,400 km). The Moon’s actual distance varies between ~363,300 km (perigee) and ~405,500 km (apogee).

‘distance_mars_sun‘: Semi-major axis of Mars’s orbit around the Sun (2.279 x 10^11 m, ~227.9 million km). Mars has a notably eccentric orbit (e = 0.093), ranging from ~206.7 million km to ~249.2 million km.

‘distance_jupiter_sun‘: Semi-major axis of Jupiter’s orbit around the Sun (7.785 x 10^11 m, ~778.5 million km).

‘distance_venus_sun‘: Semi-major axis of Venus’s orbit around the Sun (1.082 x 10^11 m, ~108.2 million km). Venus has the most circular orbit of any planet (e = 0.007).

‘distance_mercury_sun‘: Semi-major axis of Mercury’s orbit around the Sun (5.791 x 10^10 m, ~57.9 million km). Mercury has the most eccentric planetary orbit (e = 0.206), ranging from ~46.0 million km to ~69.8 million km.

‘distance_saturn_sun‘: Semi-major axis of Saturn’s orbit around the Sun (1.434 x 10^12 m, ~1.434 billion km).

‘distance_uranus_sun‘: Semi-major axis of Uranus’s orbit around the Sun (2.871 x 10^12 m, ~2.871 billion km).

‘distance_neptune_sun‘: Semi-major axis of Neptune’s orbit around the Sun (4.495 x 10^12 m, ~4.495 billion km).

‘distance_pluto_sun‘: Semi-major axis of Pluto’s orbit around the Sun (5.906 x 10^12 m, ~5.906 billion km). Pluto has a highly eccentric orbit (e = 0.249), ranging from ~4.437 billion km to ~7.376 billion km.

‘speed_earth‘: Mean orbital speed of Earth around the Sun (29,780 m/s).

‘speed_moon‘: Mean orbital speed of the Moon around Earth (1,022 m/s).

‘speed_mars‘: Mean orbital speed of Mars around the Sun (24,070 m/s).

‘speed_jupiter‘: Mean orbital speed of Jupiter around the Sun (13,060 m/s).

‘speed_venus‘: Mean orbital speed of Venus around the Sun (35,020 m/s).

‘speed_mercury‘: Mean orbital speed of Mercury around the Sun (47,360 m/s).

‘speed_saturn‘: Mean orbital speed of Saturn around the Sun (9,680 m/s).

‘speed_uranus‘: Mean orbital speed of Uranus around the Sun (6,800 m/s).

‘speed_neptune‘: Mean orbital speed of Neptune around the Sun (5,430 m/s).

‘speed_pluto‘: Mean orbital speed of Pluto around the Sun (4,740 m/s).

A Note on "Distance" Constants

Orbital distances are not truly constant. Every orbit is an ellipse, so the separation between two bodies changes continuously. The distances provided here are **semi-major axes** — the average of the closest approach (periapsis) and farthest point (apoapsis). The semi-major axis is the single most characteristic length scale of an elliptical orbit: it determines the orbital period via Kepler’s Third Law, and when paired with the circular velocity at that distance, it produces a near-circular orbit that closely approximates the real trajectory.

For example, the Earth-Sun distance varies from about 147.1 million km (perihelion in January) to 152.1 million km (aphelion in July). The semi-major axis of 149.598 million km sits right in the middle and gives the correct orbital period of one year.

plot_orbits *Plot Orbital Trajectories (Smart 2D/3D Dispatch)*

Description

Plot Orbital Trajectories (Smart 2D/3D Dispatch)

Usage

```
plot_orbits(sim_data, three_d = NULL)
```

Arguments

sim_data	A tibble output from ‘simulate_system()‘
three_d	Logical. If TRUE, forces a 3D plot even for 2D data.

Value

A ‘ggplot‘ object (2D) or a ‘plotly‘ HTML widget (3D) showing the orbital trajectories of all bodies in the simulation.

plot_orbits_3d *Plot 3D Interactive Orbital Trajectories*

Description

Generates an interactive 3D visualization of the orbital system using plotly. You can click, drag to rotate, and scroll to zoom in on the trajectories.

Usage

```
plot_orbits_3d(sim_data)
```

Arguments

sim_data	A tibble containing the simulation output from ‘simulate_system()‘.

Value

A plotly HTML widget displaying the 3D orbits.

Examples

```
create_system() |>
  add_body("Earth", mass = mass_earth) |>
  add_body("Moon", mass = mass_moon,
           x = distance_earth_moon, vy = speed_moon, vz = 150) |>
  simulate_system(time_step = seconds_per_hour, duration = seconds_per_day * 30) |>
  plot_orbits_3d()
```

`plot_system` *Plot System Snapshot at a Single Time (Smart 2D/3D Dispatch)*

Description

Plots the position of every body in the system at a single time step, optionally with the full orbital trajectories drawn faintly behind. This is the snapshot counterpart to [plot_orbits()], which draws full trajectories.

Usage

```
plot_system(sim_data, time = NULL, trails = FALSE, three_d = NULL)
```

Arguments

`sim_data`	A tibble output from [simulate_system()].
`time`	Time (in simulation seconds) to snapshot. Defaults to the last time step. The function snaps to the closest available time in the data.
`trails`	Logical. If 'TRUE' (the default), the full orbit paths are drawn faintly behind the snapshot points. Set 'FALSE' for a pure snapshot showing only the body positions at the chosen time.
`three_d`	Logical. If 'TRUE', forces a 3D plot even for planar data.

Details

If any body has non-zero motion in the Z dimension (or 'three_d = TRUE'), [plot_system_3d()] is used; otherwise a 2D 'ggplot2' plot is returned.

Value

A 'ggplot' object (2D) or a 'plotly' HTML widget (3D).

Examples

```
sim <- create_system() |>
  add_sun() |>
  add_body("Earth", mass = mass_earth, x = distance_earth_sun, vy = speed_earth) |>
  simulate_system(time_step = seconds_per_day, duration = seconds_per_year)

# Final state with faint orbit trails
plot_system(sim)

# State at day 100, no trails
plot_system(sim, time = seconds_per_day * 100, trails = FALSE)
```

`plot_system_3d` *Plot 3D Interactive System Snapshot at a Single Time*

Description

The 3D counterpart to [plot_system()]. Draws every body's position at a chosen time as a sphere in an interactive plotly scene, optionally with the full orbital trajectories shown faintly behind.

Usage

```
plot_system_3d(sim_data, time = NULL, trails = FALSE)
```

Arguments

`sim_data`	A tibble output from [simulate_system()].
`time`	Time (in simulation seconds) to snapshot. Defaults to the last time step. Snaps to the closest available time in the data.
`trails`	Logical. If 'TRUE' (the default), the full orbit paths are drawn faintly behind the snapshot points.

Value

A 'plotly' HTML widget.

Examples

```
create_system() |>
  add_body("Earth", mass = mass_earth) |>
  add_body("Moon",  mass = mass_moon,
           x = distance_earth_moon, vy = speed_moon, vz = 100) |>
  simulate_system(time_step = seconds_per_hour, duration = seconds_per_day * 30) |>
  plot_system_3d()
```

print.orbit_system *Print an orbit_system*

Description

Displays a compact, human-readable summary of an ‘orbit_system‘ showing the gravitational constant and a tibble of body states.

Usage

```
## S3 method for class 'orbit_system'
print(x, ...)
```

Arguments

x	An ‘orbit_system‘ object.
...	Additional arguments (ignored).

Value

‘x‘, invisibly.

Examples

```
create_system() |>
  add_sun() |>
  add_planet("Earth", parent = "Sun") |>
  add_planet("Mars",  parent = "Sun")
```

remove_body *Remove one or more bodies from the system*

Description

Drops bodies by name from an ‘orbit_system‘. This is the counterpart to [add_body()] — use it to strip out bodies you no longer need before simulating, or to prune a system built with [load_solar_system()].

Usage

```
remove_body(system, id)
```

Arguments

system	An ‘orbit_system‘ object.
id	A character vector of body names to remove. All names must exist in the system.

Details

‘‘‘ load_solar_system() |> remove_body(c("Pluto", "Moon")) |> simulate_system(time_step = seconds_per_day, duration = seconds_per_year) ‘‘‘

Value

The updated ‘orbit_system‘ with the specified bodies removed.

Examples

```
# Remove a single body
create_system() |>
  add_sun() |>
  add_planet("Earth", parent = "Sun") |>
  add_planet("Mars",  parent = "Sun") |>
  remove_body("Mars")


# Remove multiple bodies from the full solar system
load_solar_system() |>
  remove_body(c("Pluto", "Moon")) |>
  simulate_system(time_step = seconds_per_day, duration = seconds_per_year) |>
  plot_orbits()
```

---

save_system *Save an orbit_system to disk*

Description

Saves the full ‘orbit_system‘ object (bodies, forces, and time) to an ‘.rds‘ file so it can be restored later with [load_system()]. This preserves everything — the gravitational constant, body states, and class — exactly as it was.

Usage

```
save_system(system, path)
```

Arguments

system	An ‘orbit_system‘ object.
path	File path to save to. Should end in ‘.rds‘.

Value

‘system‘, invisibly.

Examples

```
sys <- create_system() |>
  add_sun() |>
  add_planet("Earth", parent = "Sun")

save_system(sys, file.path(tempdir(), "my_system.rds"))
```

`shift_reference_frame` *Shift the coordinate reference frame of the simulation*

Description

Recalculates the positions and velocities of all bodies relative to a specific target body. This effectively "anchors the camera" to the chosen body, placing it at the origin (0, 0, 0) for all time steps.

Usage

```
shift_reference_frame(sim_data, center_id, keep_center = TRUE)
```

Arguments

`sim_data`	A tidy ‘tibble‘ containing the output from ‘simulate_system()‘.
`center_id`	The character string ID of the body to use as the new origin.
`keep_center`	Logical. Should the central body remain in the dataset (it will have 0 for all coordinates) or be removed? Default is ‘TRUE‘.

Value

A tidy ‘tibble‘ with updated ‘x‘, ‘y‘, ‘z‘, ‘vx‘, ‘vy‘, and ‘vz‘ columns.

Examples

```
# Simulate Sun-Earth-Moon
orbit_data <- create_system() |>
  add_sun() |>
  add_body("Earth", mass = mass_earth, x = distance_earth_sun, vy = speed_earth) |>
  add_body("Moon", mass = mass_moon, x = distance_earth_sun + distance_earth_moon,
           vy = speed_earth + speed_moon) |>
  simulate_system(time_step = seconds_per_hour, duration = seconds_per_year)

# Shift view to Earth and plot
orbit_data |>
  shift_reference_frame(center_id = "Earth") |>
  plot_orbits()
```

simulate_system *Simulate kinematics for an orbitr system*

Description

Propagates the physical state of an ‘orbit_system‘ through time using numerical integration. This engine supports multiple mathematical methods, defaulting to the energy-conserving Velocity Verlet algorithm to ensure highly stable orbital trajectories.

Usage

```
simulate_system(
  system,
  time_step = seconds_per_hour,
  duration = seconds_per_year,
  method = "verlet",
  softening = 0,
  use_cpp = TRUE
)
```

Arguments

system	An ‘orbit_system‘ object created by ‘create_system()‘.
time_step	The time increment per frame in seconds (default 3600s / 1 hour). For planetary orbits around a star, daily steps (‘86400‘) are usually sufficient. For lunar-scale or tighter orbits, hourly steps (‘3600‘) work well.
duration	Total simulation time in seconds (default 31557600s / 1 year).
method	The numerical integration method: "verlet" (default), "euler_cromer", or "euler".
softening	A small distance (in meters) added to prevent numerical singularities when bodies pass very close to each other. The gravitational distance is computed as ‘sqrt(r^2 + softening^2)‘ instead of ‘r‘. Default is 0 (no softening). A value like 1e4 (10 km) is reasonable for planetary simulations.
use_cpp	Logical. If ‘TRUE‘ (default), uses the compiled C++ acceleration engine for better performance. Falls back to vectorized R if the C++ code is not available.

Value

A tidy ‘tibble‘ containing the physical state (time, id, mass, x, y, z, vx, vy, vz) of every body at every time step.

Examples

```
my_universe <- create_system() |>
  add_body("Earth", mass = mass_earth) |>
  add_body("Moon", mass = mass_moon, x = distance_earth_moon, vy = speed_moon) |>
```

```
simulate_system(time_step = seconds_per_hour, duration = seconds_per_day * 28)
```

Index