Orbital Simulator Game – PHYS 3120 Presentation

Alice Heranval, Jami Milliken, Matvey Shakula, Garrett Youngblood
Georgia Institute of Technology

Background

The Science Behind the Game

Orbits in space are governed by gravity—the same force that keeps planets circling the sun. Predicting these motions is a classic physics challenge, especially when multiple objects interact (the n-body problem). While exact solutions are rare, computers can simulate orbits using numerical methods like Runge-Kutta, which calculates step-by-step trajectories.

Why a Game?

Simulations help us understand complex physics, but they're often static or slow. This project asks:
Can we make an orbital simulator fast and interactive enough to feel like a video game?

Educational & Computational Goals

Learn orbital mechanics through hands-on play.
Test numerical methods for speed and accuracy.
Explore challenges like energy conservation and moving bodies.
By blending astrophysics with gaming, this project makes orbital mechanics intuitive, interactive, and fun!

Physics Involved

Our simulator is governed by Newton's law of universal gravitation and the resulting equations of motion for a point mass rocket under the influence of one or more massive bodies.

Gravitational Force and Acceleration

For each fixed body of mass M_i at position r_i, the gravitational force on the rocket (mass m_r) at r is:

$$\boldsymbol{F}_i = -G\frac{M_i m_r}{|r-r_i|^3}(r-r_i)$$

Summing over all ***N*** bodies gives the net force, from which we get the acceleration **a**:

$$\mathbf{a} = \frac{\boldsymbol{F}_{net}}{\boldsymbol{m}_r} = -G\sum_{i=1}^{N}\frac{M_i}{\left|r-r_i\right|^3}(\mathbf{r}-\mathbf{r_i})$$

Mechanical Energy Conservation

The total mechanical energy of the system is:

$$E = \frac{1}{2}m_r v^2 - G m_r \sum_{i=1}^{N}\frac{M_i}{\left|r-r_i\right|}$$

In an isolated two-body central potential, this energy remains constant; deviations in energy over time are numerical artifacts. Using **4th-order Runge-Kutta (RK4)** here does not preserve energy over an extended period of time. This idea led us to implement a simulator to compare orbits calculated from RK4 and Verlet methods.

More Physics

Angular Momentum

In any central potential, the angular momentum $L = r \times p$ is conserved ($dL/dt = 0$), which should give us specific orbital shapes on a 2D plane.

Circular Orbit Conditions

For a perfect circular orbit of radius r about a single mass M, the speed v and period T are:

$$v = \sqrt{\frac{GM}{r}}, \quad T = 2\pi\sqrt{\frac{r^3}{GM}}$$

Model Setup

Physics Engine Core

Gravitational Modeling:

- Gravitational constant G is tunable (default 2000 px³/s² for stable orbits at screen scale)
- Point-mass approximation for all bodies (rocket + celestial objects)

Dimensionality

- 2D simulation (x,y plane) for computational efficiency and gameplay clarity
- Z-axis forces/depth effects omitted for simplicity

Numerical Integration Methods

RK4:

- Time step Δt=0.5Δt=0.5 s balances accuracy and real-time performance
- Derivative evaluations per step: 4 (k1-k4)
- State vector: $[x, y, v_x, v_y]$
- Energy drift ~0.5% over 150 orbits (measured)

Verlet:

- Time-reversible integrator
- Position update: $r_{t+\Delta t} = 2r_t - r_{t-\Delta t} + a_t\Delta t^2$
- Velocity calculated via central difference
- Theoretically conserves energy

Object Dynamics

Rocket:

- Mass: 1 (unit mass, normalized)
- Initial velocity: 75-150 px/s (player-controlled)
- Rotation: Dynamically oriented to velocity vector

Celestial Bodies:

- Planets: Fixed positions (WIDTH/2, HEIGHT/2)
- Asteroids: Randomly generated with:
- Mass: ∝ sprite area (50-150 px → 250-2250 mass units)
- Minimum separation: 100 px buffer + satellite width

Collision & Boundary Handling

Collision Detection:

- If the distance between the rocket and an asteroid is less than the sum of their radii, the game stops as a collision has occurred.
- Planet collision: Hard boundary at surface

Game Boundaries:

- Screen edges (0 ≤ x ≤ WIDTH, 0 ≤ y ≤ HEIGHT)
- Exit condition: Rocket position outside bounds

N-Body Simulation

Launch System:

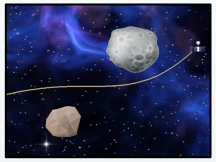
- Players aim a space cannon using mouse controls
- Fixed initial velocity with adjustable launch angle
- Rocket modeled as a point mass with trajectory calculated via RK4 integration

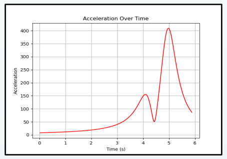
Gravitational Environment:

- 2 randomly generated (size and position) asteroids
- Each asteroid has:
  - Mass proportional to visual size (larger = more gravity)
  - Defined "sphere of influence" affecting the rocket

Game Conditions:

- Success: Take pictures of all asteroids by entering their photo zones (within defined distance of surface)
- Failure: Colliding with any massive body, exiting screen boundaries, failing to photograph all targets





Energy Conservation in Orbit Simulation

Objective: Compare the behavior of RK4 and Verlet integration methods in modeling orbital motion

Setup:

- Rocket launched from a planet's surface with fixed speed
- Rocket treated as a point mass under gravity from a central planet

Gravitational Environment:

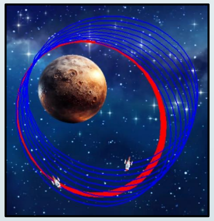
- Single central body with fixed mass and position
- Gravity modeled as an inverse-square law (F = –GM/r²)

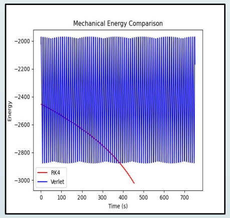
Simulation Goals:

- Complete multiple revolutions around the planet without crashing or drifting

Technical Implementation

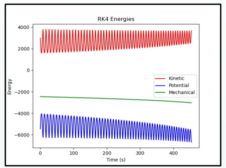
- Central force problem with real-time updates
- Integrators: RK4 and Verlet
- Time evolution visualized frame-by-frame

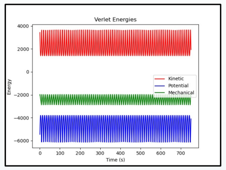




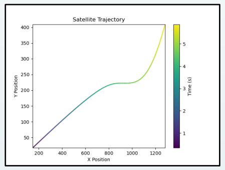
Results

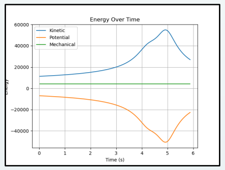
In the RK4 vs Verlet orbital simulation, both methods produced visually accurate and bounded orbits under a central gravitational force. RK4 offered smoother trajectories with high short-term accuracy, while Verlet showed orbital precession due to its reliance on position history. Although Verlet better conserved mechanical energy over time, eliminating its artificial precession required very small time steps (on the order of 1e-4), which are impractical for real-time simulations. The computational cost of such fine resolution makes Verlet unsuitable for interactive applications. In contrast, RK4 tolerates larger time steps and offers greater per-step accuracy, making it more effective for real-time gameplay where performance and visual clarity outweigh long-term stability.





In the satellite asteroid imaging simulation, successful missions depended on the strategic use of mid-course impulse burns. Players were more likely to photograph all asteroids when using posigrade burns to extend orbital arcs and retrograde burns to decelerate for tighter flybys. Failures typically resulted from early collisions or missed targets due to lack of trajectory control. The RK4 integrator handled the complex multi-body gravitational environment effectively, allowing for real-time path corrections in response to nonlinear deflections.





Additional Projects & References

Lagrange Points

- A simulation exists, but due the inherently unstable nature Lagrange points 1-3, preserving realistic physics means significantly reducing numerical error.
- Reduction techniques constrained by simulation runtime and machine epsilon.
- Possible solutions: adaptive timesteps, higher-order integrator, & numerical "nudges".

Atmospheric effects

- Atmospheric drag could be incorporated into another n-body simulation where planets are considered instead of asteroids.
- In a simulation with atmospheric drag, and aero-braking campaign could count as mission success.

Visual effects

- Would like to include an animation for when an impulse is applied
- Need to fix angle of rocket in the energy conservation simulation

References

- *Walter Dehnen and Justin I. Read. "N-body simulations of gravitational dynamics". In: The European Physical Journal Plus 126 (2011), p. 55.*
- *Sverre J. Aarseth. Direct Methods for N-body Simulations. Academic Press, 1985. isbn: 0-12-123420-7.*
- *Mark Newman. Computational Physics. 1st ed. San Francisco, CA: CreateSpace Independent Publishing Platform, 2012. Chap. 8.1.3.*