Simulating the Orbital Dynamics of Lagrange Points

Xavier Schrage

Problem

There are few programs that simulate the physics of Lagrange points using only gravity and inertia. This lack of visualization tools means students have a hard time understanding the mechanics of Lagrange points. For this project, a program was created to simulate the three-dimensional orbital dynamics of a solar system and a drifting spacecraft. The Lagrange points and their effects appear as a consequence of orbital mechanics.

Background

- Lagrange points are places where the gravitational forces of two large orbiting bodies cancel out the centrifugal force on a small, third object in the system. This allows a body with a negligible mass to sit at that point without being disrupted by either of the two larger bodies' gravitational pulls.
- There are 5 Lagrange points in each system of two massive bodies. See *Figure 1* for their positions.
- L4 and L5 are stable equilibriums, so an object can orbit the points as if there were a mass there.
- The other three Lagrange points are technically unstable, but a spacecraft can remain in these locations with minimal propulsion.
- Lagrange points are used as places to put spacecraft. Notably, the James Webb Space Telescope is located at the L2 Lagrange point, however, since L2 is unstable, the JWST must make correction burns frequently to stay at L2.
- Earth has two stable Lagrange points, L4 and L5, however only L4 is known to have natural satellites.
- Jupiter has so many objects at its L4 and L5 points, that they have names for each type.
  - The asteroids that orbit Jupiter's L4 point are called "Greeks"
  - The asteroids that orbit Jupiter's L5 point are called "Trojans"
  - Jupiter has over 15,000 known Trojan and Greek asteroids, and is estimated to have almost a million of them.

Design

- The program's physics system and its rendering system are separate, which allows for both the number of timesteps taken and the size of each timestep to be modified independently from the actual rendering of the program.
- Every time a frame in the program is rendered, the function to handle the physics system is called from a separate translation unit, which allows the physics or rendering system to be more easily reused for different projects in the future.
- The physics function calls sub-functions which each handle a different aspect of the system. Each of those functions iterate for as many physics timesteps per frame as the user sets, each with a delta time, which the user also sets.
- The framerate is set to have a maximum of 60 frames per second in order to improve consistency across devices.
- The user can select how many timesteps they want the program to calculate each frame.

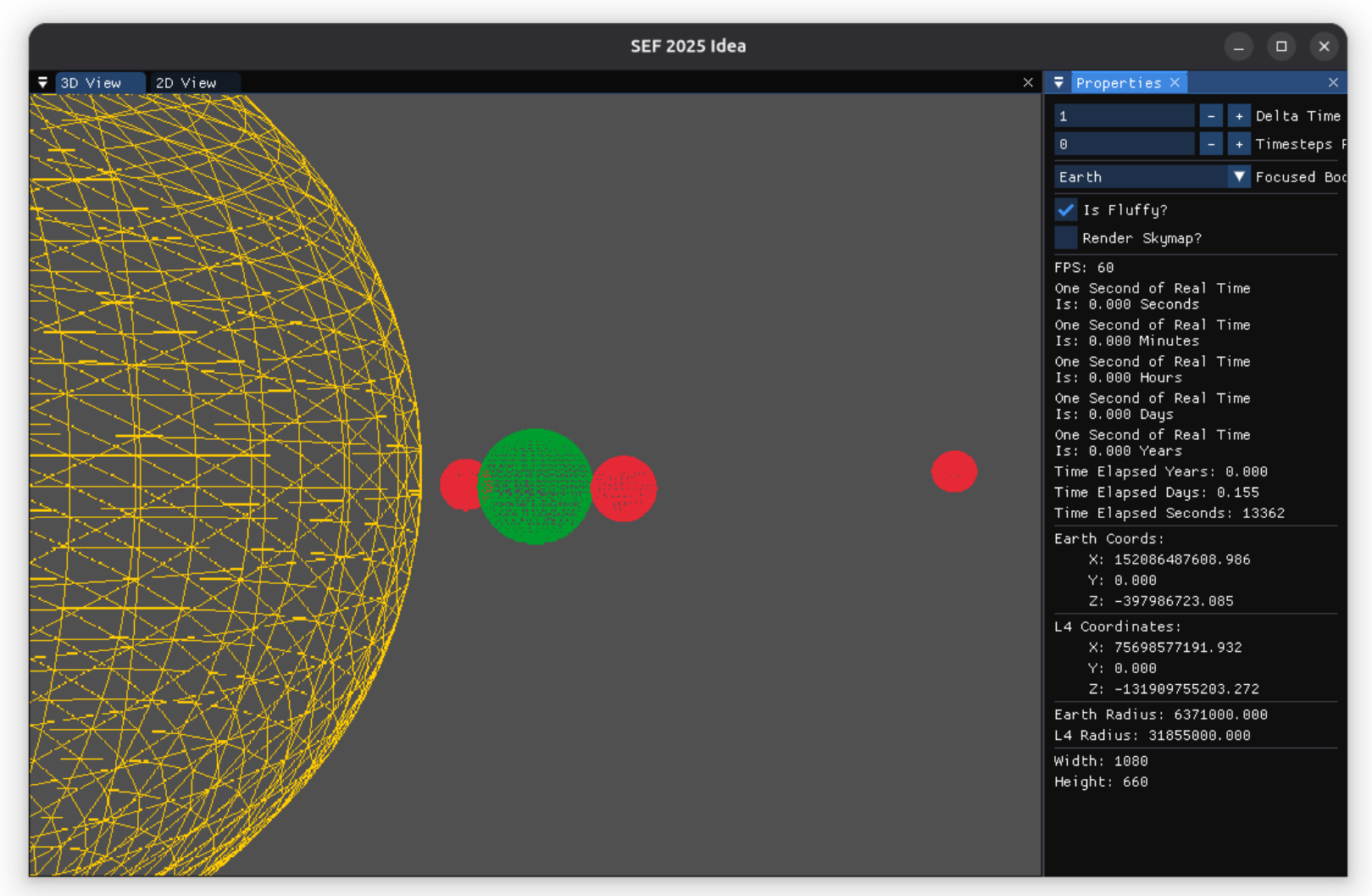
Physics Equations

```
For Each Timestep
    For Each Gravitational Body
        $F_{obj} = \frac{G m_{ship} m_{obj}}{distance^2}$
        $F_{x,obj} = -F_{obj} \frac{distance_x}{distance}$
        $F_{y,obj} = -F_{obj} \frac{distance_y}{distance}$
    $F_{x,net} = \sum_{obj} F_{x,obj}$
    $F_{y,net} = \sum_{obj} F_{y,obj}$
    $a_{x,net} = \frac{F_{x,net}}{m_{ship}}$, $a_{y,net} = \frac{F_{y,net}}{m_{ship}}$   } Newton's Second Law
    $V_{x,1} = V_{x,0} + a_{x,net} * \Delta t$, $V_{y,1} = V_{y,0} + a_{y,net} * \Delta t$   } Compute new velocity based on existing velocity
    $x_1 = x_0 + v_{x,1} * \Delta t$, $y = y_0 + v_{y,1} * \Delta t$   } Compute new approximate location of the ship
```

$$F = m * a$$

$$F = \frac{G m_1 m_2}{r^2}$$

Every 60th timestep is rendered to the screen



```
1          - + Delta Time
0          - + Timesteps P
Earth      ▼   Focused Bod
[✓] Is Fluffy?
[ ] Render Skymap?
FPS: 60
One Second of Real Time
Is: 0.000 Seconds
One Second of Real Time
Is: 0.000 Minutes
One Second of Real Time
Is: 0.000 Hours
One Second of Real Time
Is: 0.000 Days
One Second of Real Time
Is: 0.000 Years
Time Elapsed Years: 0.000
Time Elapsed Days: 0.155
Time Elapsed Seconds: 13362
Earth Coords:
    X: 152086487608.986
    Y: 0.000
    Z: -397986723.085
L4 Coordinates:
    X: 75698577191.932
    Y: 0.000
    Z: -131909755203.272
Earth Radius: 6371000.000
L4 Radius: 31855000.000
Width: 1080
Height: 660
```

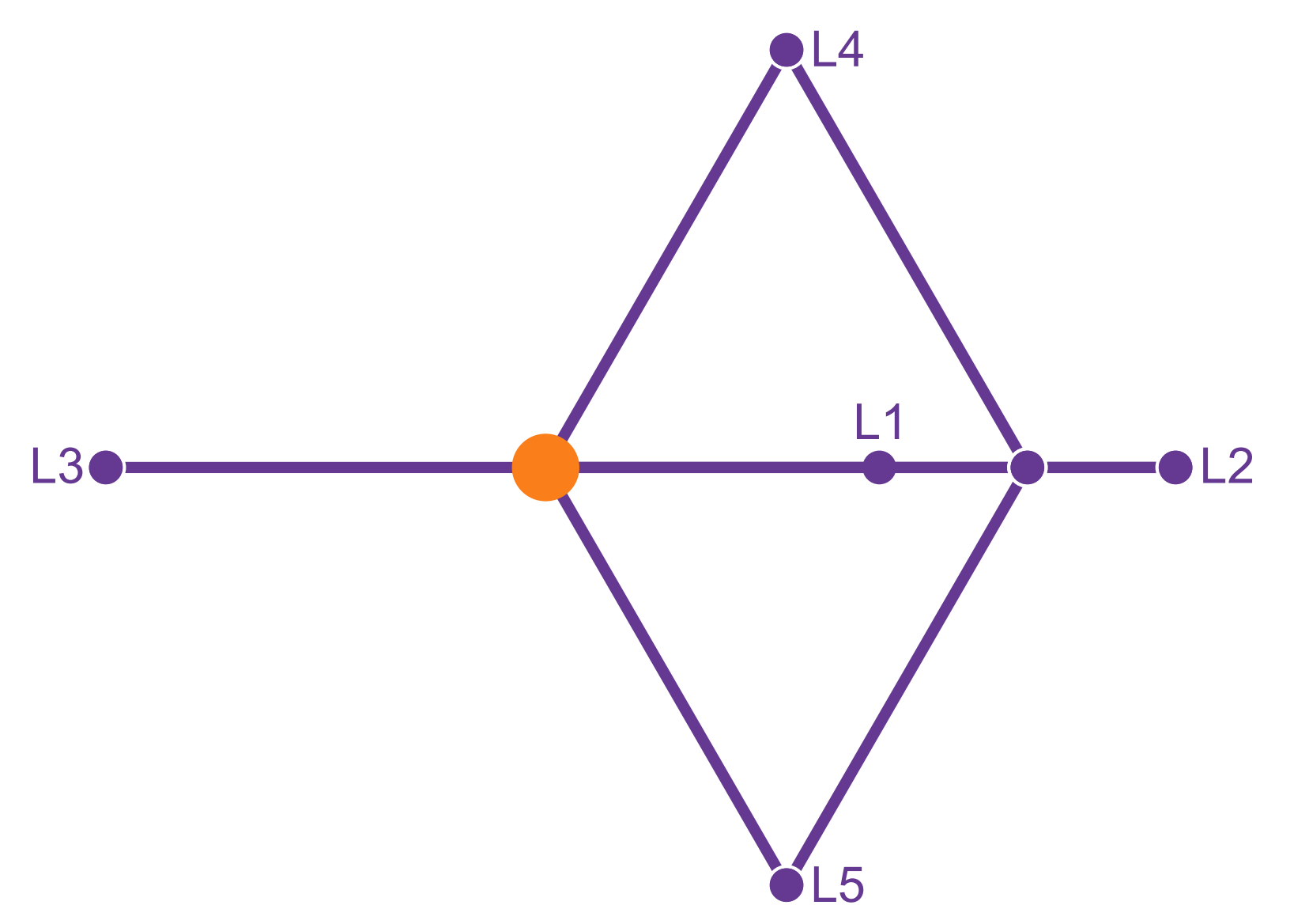


Figure 1.

Citations

- https://en.wikipedia.org/wiki/Lagrange_point
- https://en.wikipedia.org/wiki/Jupiter_trojan
- https://en.wikipedia.org/wiki/Earth_trojan
- Bueche, Frederick J. "Rotation and Orbital Motion." Introduction to Physics for Scientists and Engineers, 4th ed., McGraw Hill, St. Louis, MO, 1986, pp. 143-143.
- https://www.missionjuno.swri.edu/junocam/
- Szebehely, Victor G. "Description of The Restricted Problem" Theory of Orbit: The Restricted Problem of Three Bodies, Academic Press, New York, 1967, pp. 18.

Solution

- C++ was chosen for the program because it is well-suited for large computations.
- The gravitational forces for every object were computed using Newton's laws of gravity and motion.
- The position, velocity, and acceleration of every object is stored in double-precision floats to improve accuracy.
- Visualization of the model can be modified without sacrificing accuracy of the simulation. All units used in computation have SI units and maintain real-world scale.

Results

- The program can successfully simulate the orbital dynamics of a restricted 3-body problem using a computational model.
- Objects positioned near the stable Lagrange points appear to be correctly moving around the libration points, but it is not yet clear if these trajectories would remain stable for astronomical timeframes in the simulation.
- Unfortunately, camera navigation in the program is currently hindered by limitations of the graphics library used.

Conclusion

- A computational model of an n-body system that allows for the existence of Lagrange points has been developed and tested for this project.
- The Lagrange points seem to behave just as they do in real-life, barring computational errors.
- Even though the simulation follows pure Newtonian mechanics, it is still only an approximation of the actual orbital system, as it is computationally impossible to perfectly simulate an n-body system.

Materials

- C++ Compiler (GCC was used)
- Make
- Computer Workstation
- Raylib Graphics Library
- Dear ImGui Library