Simulating N-Body Orbits

by Jake Hauser for Physics 125

Introduction

Motivation

John Taylor devotes an entire chapter of his Classical Mechanics textbook to the issue of two-body central-force problems, because they can be elegantly reduced to one-body problems that approximate a wide variety of astronomical and quantum systems.[1] While the mathematical machinery built in this chapter is powerful, the approximations used are significant: real-world systems are often impacted by more than two bodies and can be driven by unusual force laws, leading to messy results.

Because two-body models are not universally applicable, an analytical solution to the three-body problems, and even the n-body problem, is very appealing. Unfortunately, neither exists in any generalized form. Although a specific subset of three-body problems – restricted three-body problems, where the third mass is negligible – have analytical solutions, n-body problems in general are often referred to as unsolvable because the method of first integrals cannot be applied.[2] In fact, they are analytically solvable using power series; however, these series converge so slowly that they have little practical use: "one would have to sum up millions of terms to determine the motion of the particles for insignificantly short intervals of time".[3] Therefore, the only reasonable way to model n-body systems is to approximate bodies' trajectories numerically.

Moreover, Bertrand's Theorem dictates that only two central force laws guarantee closed orbits:[4]

$$F(r) = \alpha r \text{ and } F(r) = \frac{\alpha}{r^2}$$

This introduces additional complexity to general n-body systems, further motivating the use of accurate numerical solutions.

The primary motivation for this project is an interest in the orbits observed historically by astronomers, and why the geocentric model of the solar system was once so popular. Scientists aren't offered a top-down view of the solar system, so astronomers can't directly observe the pretty conic sections we discuss in class. Therefore, I built an interface able to model systems from the perspective of any body in the system, in order to illustrate the genius that would have been required to theorize a heliocentric model and to help build mental models for n-body motion from various perspectives.

---

[1] John Taylor, *Classical Mechanics,* (University Science Books, 2005), 293.
[2] Eric Weisstein, "Restricted Three-Body Problem", *WolframResearch*, last modified 2007, http://scienceworld.wolfram.com/physics/RestrictedThree-BodyProblem.html.
[3] Florin Diacu, *The Solution of the* n*-body Problem**, (The Mathematical Intelligencer Vol. 18, No. 3, 1996).
[4] Porter Johnson, *Classical Mechanics with Applications*, (World Scientific, 2010), 149.

As a proof of concept, the program I built produces similar orbits from the perspective of Earth as Ptolemy's geocentric model of the solar system.

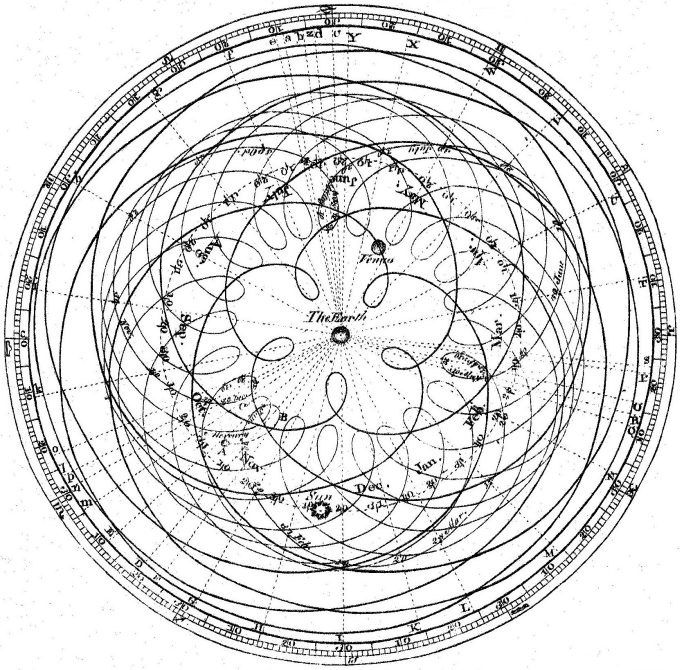

Figure 1 Ptolemaic model of the solar system (a geocentric model)[5]

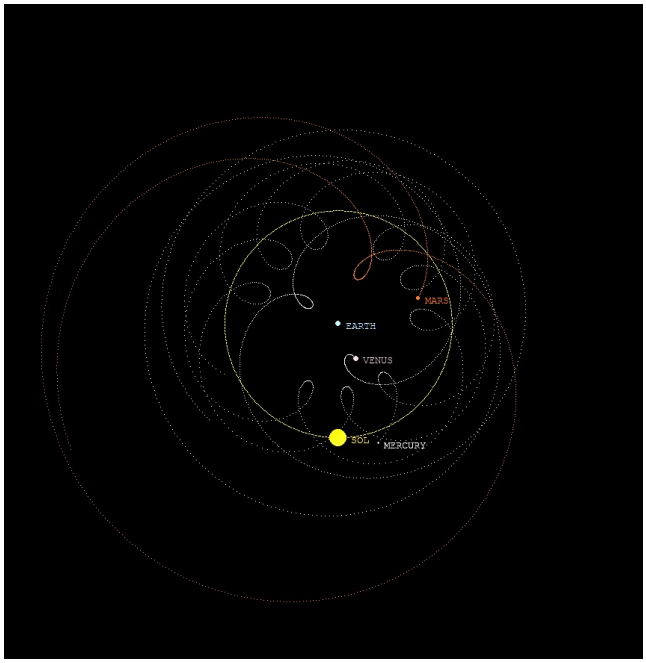

Figure 2 My model of the solar system, from the perspective of the Earth

[5] James Ferguson, (Encyclopaedia Britannica, 1st Edition, 1771), https://commons.wikimedia.org/wiki/File:Cassini_apparent.jpg.

Code Outline

My project, coded in Python using a graphical library called Pygame:

1. models n-body systems using any body as a reference frame, drawing trajectories with dots indicating uniform time intervals between points;
2. allows the user to scroll and zoom around the frame, to centre the system, and to reset to initial conditions;
3. implements our solar system with proper initial conditions sourced from JPL ephemerides on 27 March 2017;
4. can generate gravitational potential lines, of varying step sizes and resolutions, surrounding the system's bodies;
5. creates systems with randomly generated initial conditions; and
6. supports user-written custom systems.

Method

Approximation

At each frame, the program calculates the net force $\vec{F}$ on each body, divides this force by the body's mass m to get an acceleration, and multiplies this acceleration by a small time step dt to find an instantaneous change in velocity. The body's total velocity $\vec{v}$ is adjusted accordingly, and then multiplied by dt to find an instantaneous change in position. Finally, the body's position $\vec{s}$ is adjusted accordingly. This is known as the Euler method, and can be mathematically expressed in this case as:

$$\begin{aligned}\vec{v}_{n+1} &= \vec{v}_n + \frac{\vec{F}_n}{m} \cdot dt \\ \vec{s}_{n+1} &= \vec{s}_n + \vec{v}_{n+1} \cdot dt \\ &= \vec{s}_n + \vec{v}_n \cdot dt + \frac{\vec{F}_n}{m} \cdot dt^2\end{aligned}$$

The larger the value of dt, the greater the disparity between the actual trajectory and the calculated approximation. However, a larger dt also corresponds to a faster running program. Therefore, one must balance speed and accuracy (as in most projects), weighting the two differently for each simulation, depending on the speeds and distances involved. For example, the speed at which the outer planets of the solar system move at reasonable speeds is too fast for the inner planets to be simulated accurately.

Figure 3 illustrates the increasing error that is propagated by Euler method approximations. *Figures 4-6* show the trajectories of the inner planets for three possible time steps, demonstrating the increasing error created.

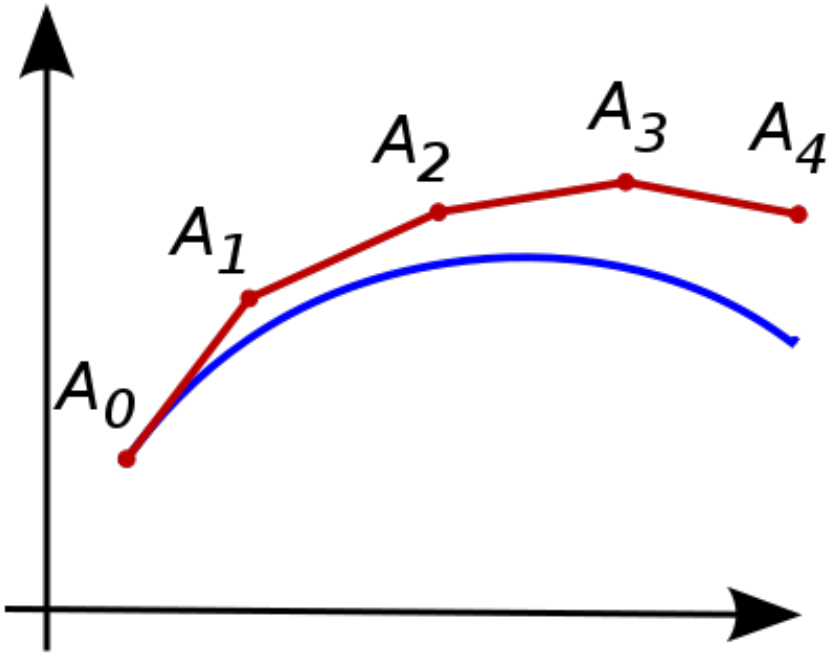

Figure 3 The red points represent the Euler approximation of the blue curve

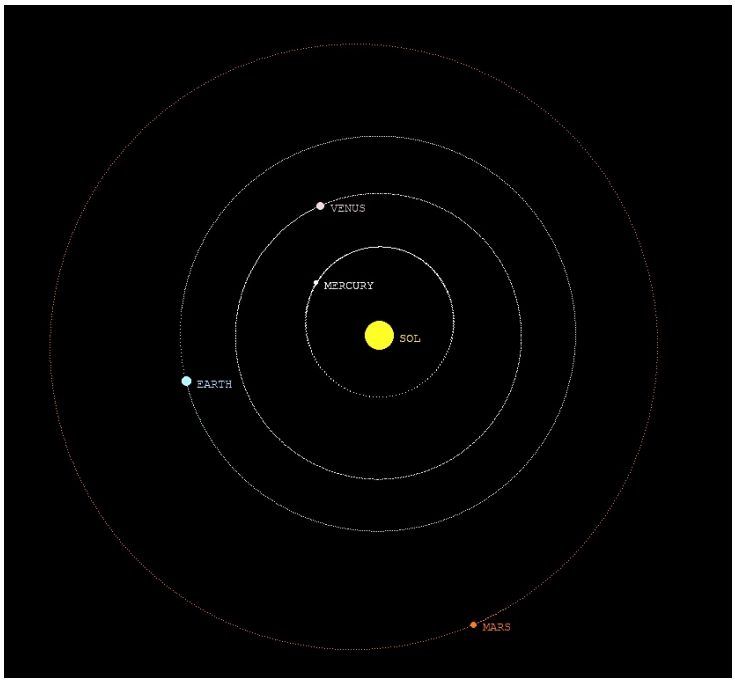

Figure 4 The inner planets of the solar system with dt = 1 day.
Note the degree to which the bodies follow smooth, closed orbits.

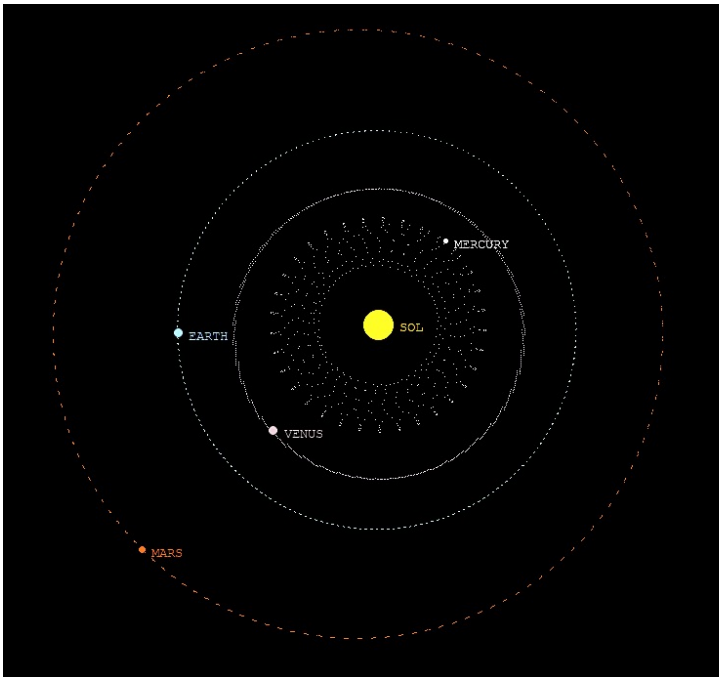

Figure 5 The inner planets of the solar system with dt = 10 days.
Note that Mercury, the planet which orbits fastest, no longer has a stable elliptical orbit.

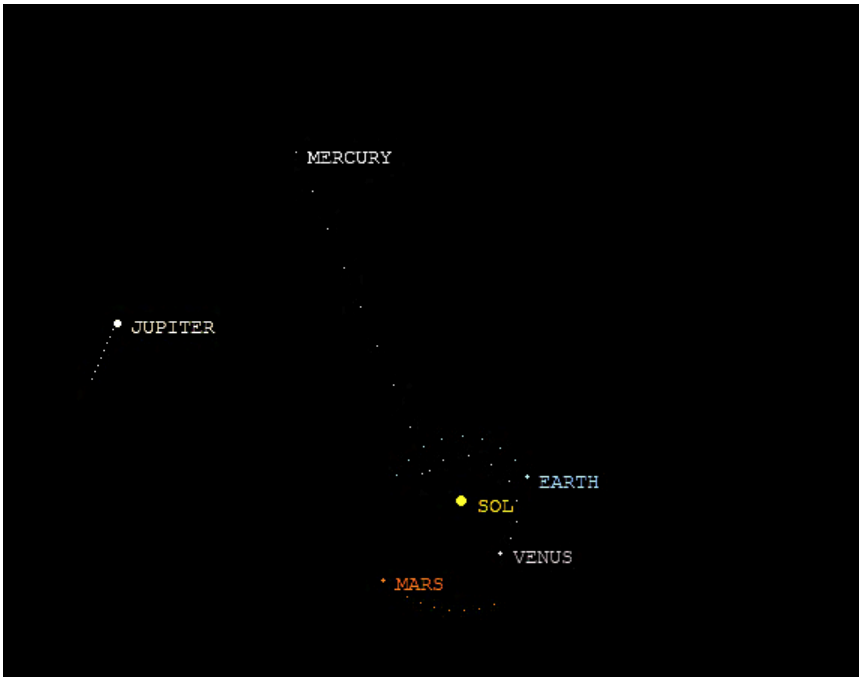

Figure 6 The inner planets of the solar system with dt = 15 days
Note that Mercury immediately exits the solar system due to approximation inaccuracies

Input

All of a system's key characteristics are communicated to the program by input files. These files have the header:

```
kg;x10^x;km;(AU,AU);(AU/day,AU/day);(R,G,B);NAME
```

This header row separates sections by semicolons and provides information about units. Each subsequent row describes an individual mass in the system. The first two columns describe a body's mass: if the number in the first column is a and the number in the second is b, then the body's mass is $a \cdot 10^b \ kg$.

The third column is the body's radius in kilometres. The fourth column is a tuple representing the body's position, in astronomical units (AU), projected onto the xy-plane (where the first coordinate is the x-position and the second is the y-position). The fifth column is the equivalent tuple representing the body's velocity in AU per day. The sixth column is a colour in RGB format. The seventh column is the body's name, drawn on the screen when the user moves their cursor over the body; if a name is not specified then the body's mass is reformatted and used instead.

For example the sun's input line looks like this:

```
1.99;30;696342;(0.00313,0.00450);(−0.00000350,−0.00000657);(253,184,19);SOL;
```

Simulation Loop

Once the program has progressed past the instruction screen, it begins the main simulation loop, which continues until the program is exited. The loop executes once each frame, holding the program to a maximum framerate of forty frames per second.

First, the program reprints the background. Next, it checks for events, which can be user keystrokes, mouse movements, or requests to resize the simulation window (when not in full screen mode). When the program is not paused, it next goes through the process of Euler method approximations, checking for collisions between bodies along the way.

After combining and deleting bodies as appropriate for a frame's collisions, the program moves the user's viewpoint according to arrow key movement. Next, it moves and draws each body and its respective list of trace points, adjusting movements according to the chosen reference frame body. This is also the stage where the loop tests for mouse collisions with bodies in the system, to determine whether to switch reference frames and whether to draw a body's label on the screen.

Finally, Boolean flags corresponding to arrow key movements and tracing are reset and the display is updated, ready for another frame.

Auxiliary Code

While I include summaries of key elements here, the reader should refer to the code itself for relevant comments in proper context.

The `vector` class supports convenient objects used throughout this program: vectors. This class improves the brevity and readability of code by making it simple to perform vector operations on tuple-like datatypes using natural operations. By inheriting base python operations, the `vector` class allows the use of `+`, `-`, `*`, `/`, and `^` operations on vectors. The `vector` class interprets a number times a vector as a scalar multiplication and a vector times a vector as a dot product.

The `body` class is designed to support easy implementation of bodies moving through space and interacting via forces. `body` objects keep their own key information (that defined by relevant input files) as well as their ongoing traces. Moreover, `body` methods make it easy to change a body's velocities and positions.

This project uses several auxiliary functions to perform key modeling tasks. `transform`, `detransform`, and `radScale` all convert information about bodies to forms compatible with pixels and Pygame drawing functions. `drawCircle` and `drawText` are wrappers on Pygame drawing functions that allow easy function calls and coordinate transformations. `gravity`, `collision`, and `averageColour` all contribute to the running of the program, the former by carrying out force calculations and the latter two by helping with mass and mouse collisions. Finally, `potential` and `contourLines` facilitate gravitational potential lines and `parse` properly reads in text input.

Results

The strongest feature of my program is its versatility. At this stage, the user is restricted to models with only one force (which must be central and proportional to the two masses involved), but otherwise the user has absolute freedom with regards to the system setup and force law power. This means that although the following sections provide example simulations, the best way to explore this project is to play with the program itself.

With that in mind, some instructions currently configured for Macintosh operating systems – will hopefully update with Windows information soon: to run the program at any time, simply run the application itself. The only nuance here is in changing the run configuration. To make changes, right click the application and click Show Package Contents. From there you should go into Contents and then Resources, from which you can open "variables" with your preferred text editor. This file allows you to edit the run configuration.

If you change "variables" to run the "custom" system information, you'll likely want to edit this file as well. It can also be found in Resources, and you should edit it following the system input format described earlier in this report.

Solar System

The following diagrams illustrate both the accuracy of this project's simulations – producing recognizable closed orbits for our solar system – and the degree of insight required to hypothesize a heliocentric model. At first glance, although the planets experience retrograde motion with respect to Earth, their orbits seem generally geocentric, and only Mercury seems to orbit the Sun over time scales like a few Earth years. Graduating to a heliocentric model therefore requires an inspired examination of how distances between the Sun and other planets vary over time.

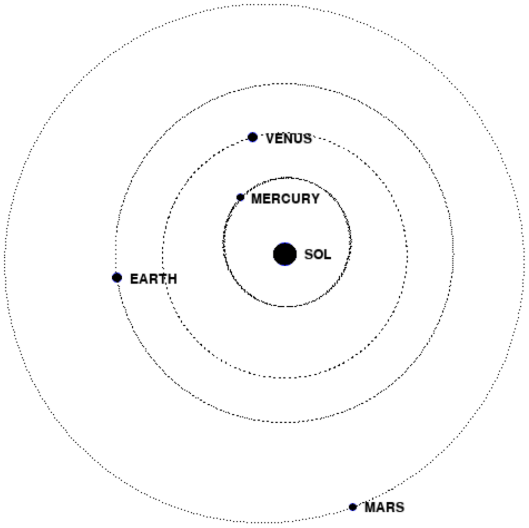

Figure 7 Inner solar system from Sun's perspective

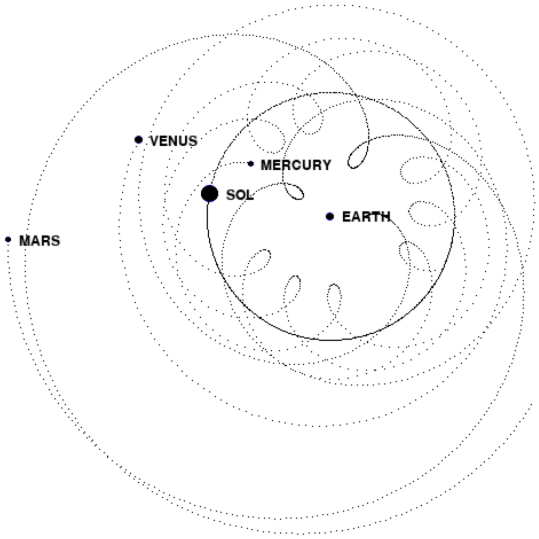

Figure 8 Inner solar system from Earth's perspective

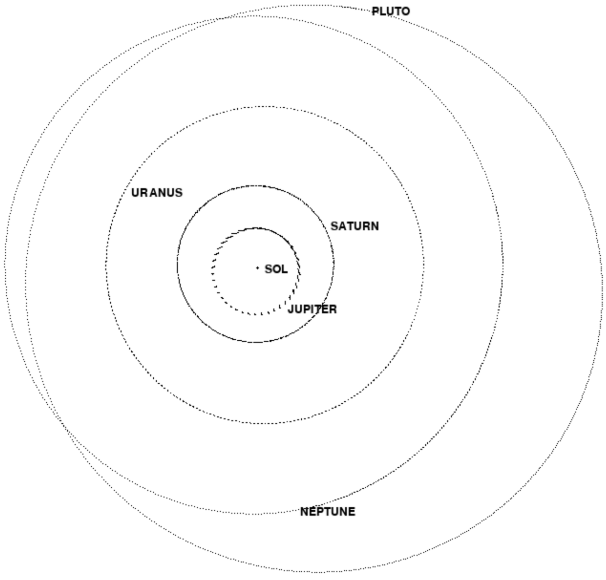

Figure 9 Outer solar system from Sun's perspective

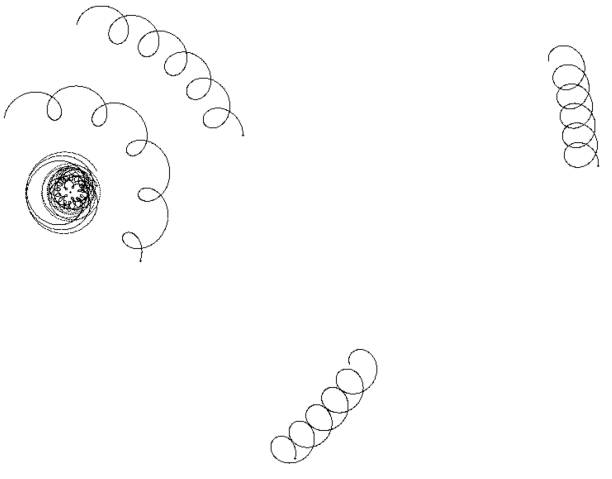

Figure 10 Outer solar system from Earth's perspective

Force Laws

The transformed radial equation used to solve the two-body problem isn't limited to inverse-square force laws. Likewise, it's useful for an n-body orbital simulator to support any central, conservative force law; such forces exist in nature, and examining them in this setting can yield interesting comparisons between force laws. For example, the force law in *Figure 12* is equivalent to Hooke's Law, and the planets' trajectories follow what you would expect for this situation. Moreover, Earth seems to survive the force laws of *Figures 11, 13,* and *14*, at least for the time ranges depicted.

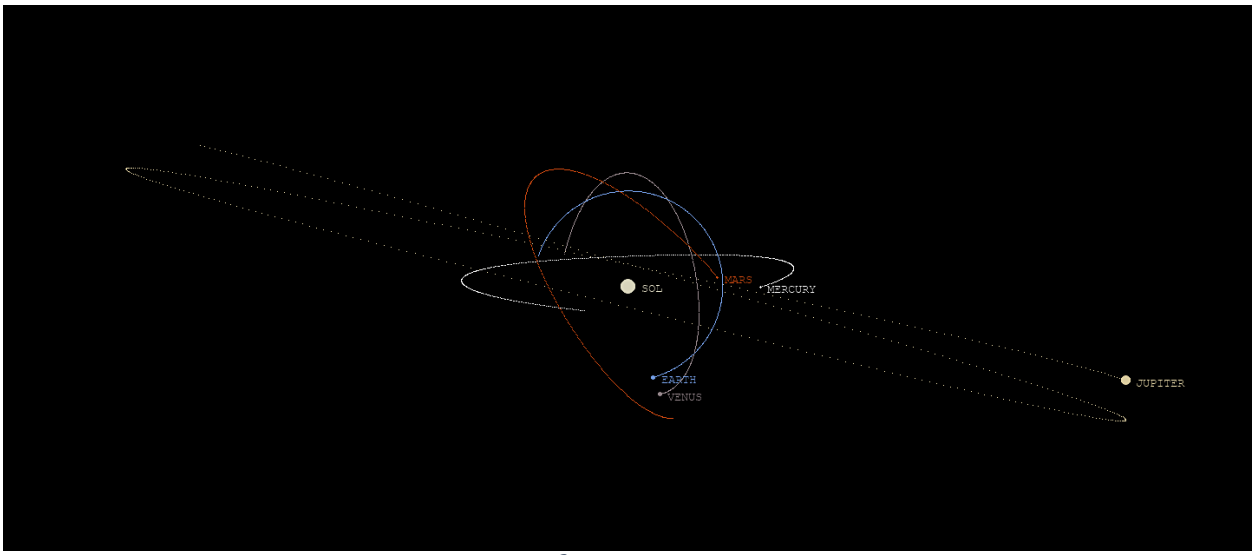

Figure 11 Trajectories of Solar System bodies when $F \propto r^2$

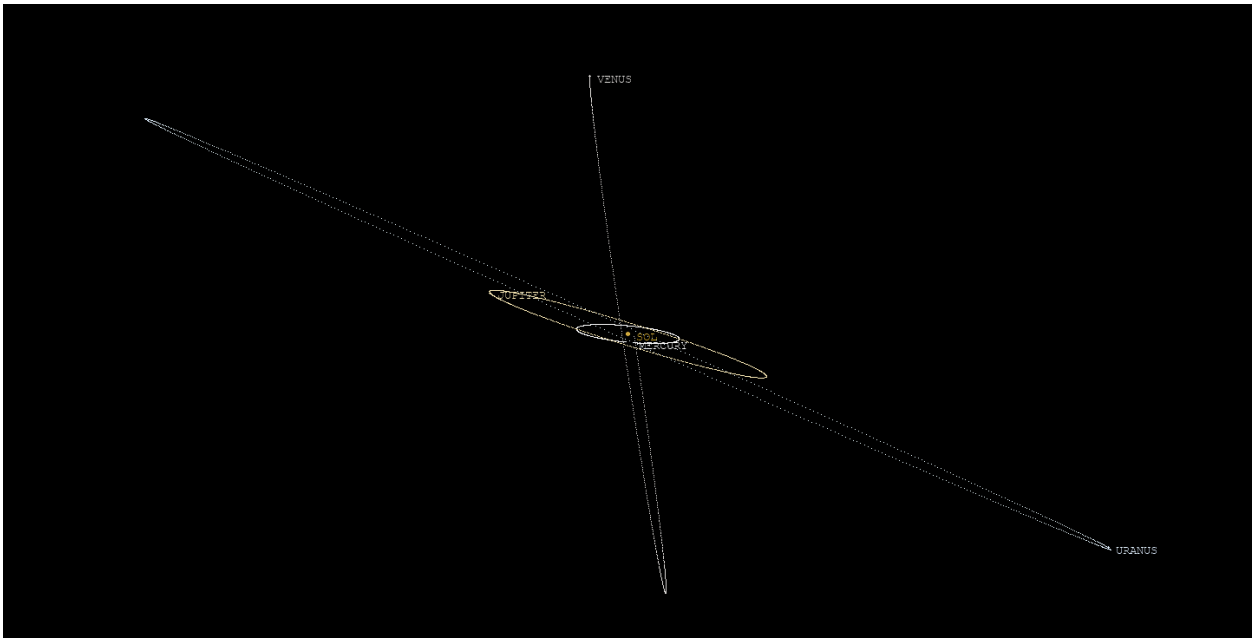

Figure 12 Trajectories of Solar System bodies when $F \propto r$; note the closed orbits predicted by Bertrand

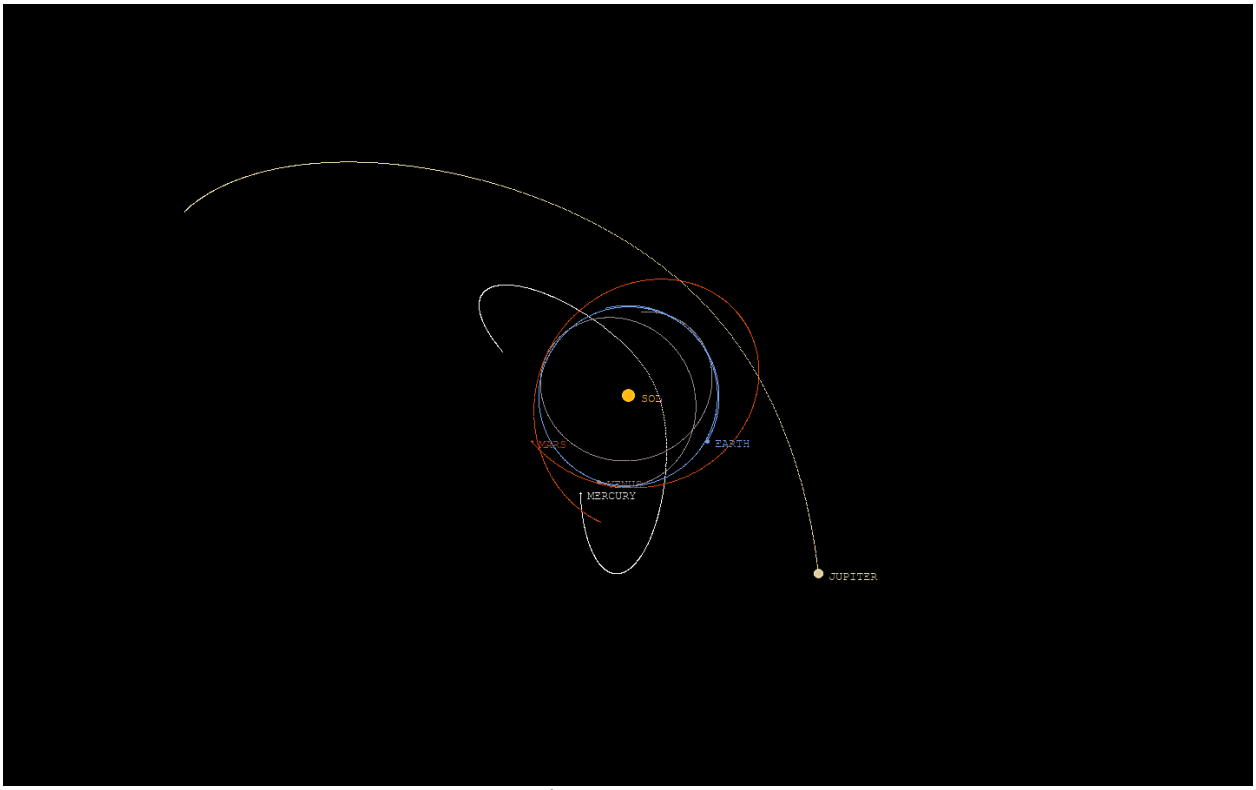

Figure 13 Trajectories of Solar System bodies when $F \propto \frac{1}{r}$

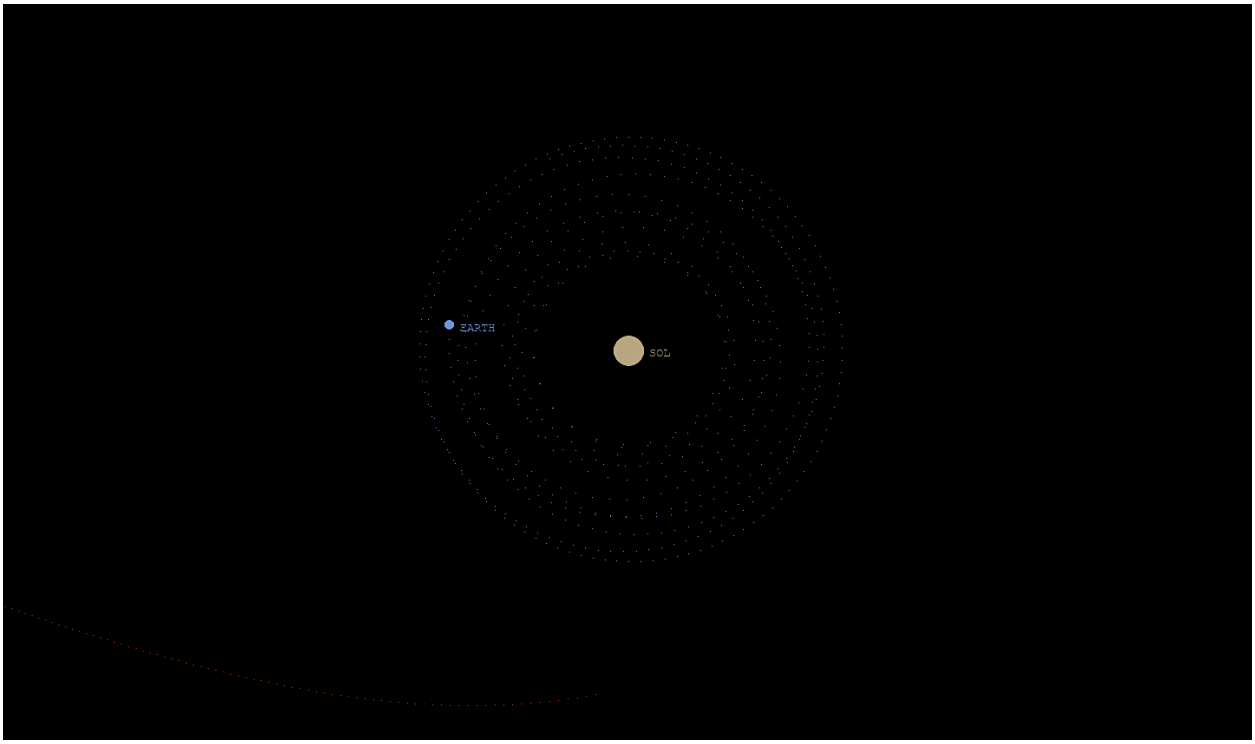

Figure 14 Trajectories of Solar System bodies when $F \propto \frac{1}{r^3}$; note that of the planets in the Solar System, only Earth seems to survive this force law, although its orbit is not closed

Other Systems

Various initial conditions lend themselves to insightful simulations using gravitational potential lines, varying force laws, and even normal gravity. The following section uses a two-body, a four-body, and an eight-body system, and the reader is encouraged to play with more models besides these.

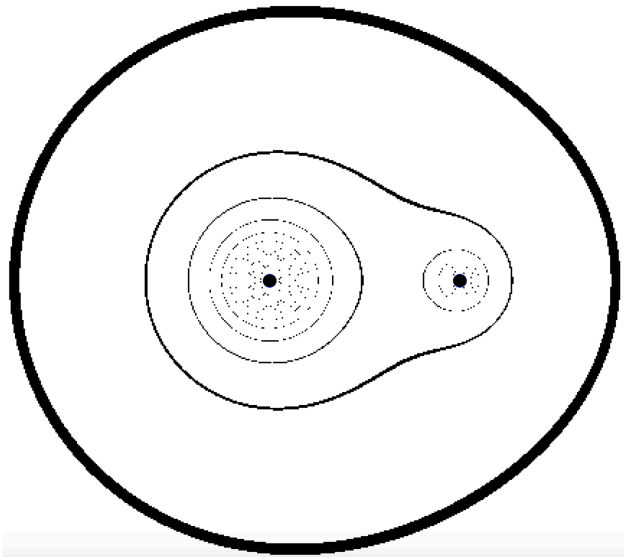

Figure 15 Gravitational potential lines for a binary star system

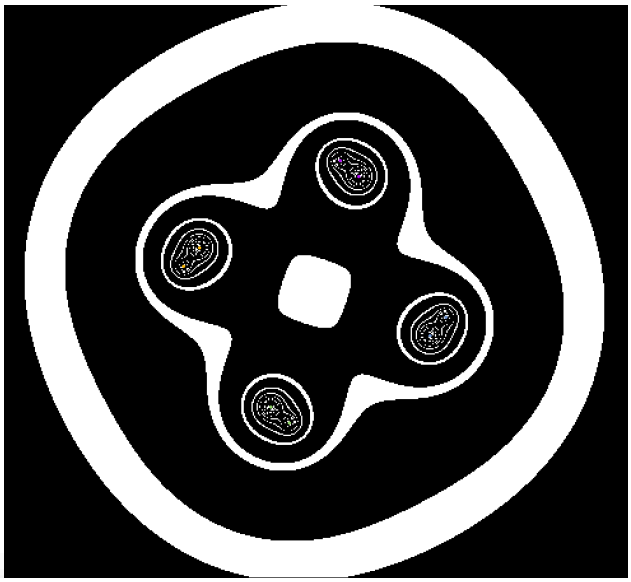

Figure 16 Gravitational potential lines for an eight-body system; thick white areas are due to a higher tolerance for potential

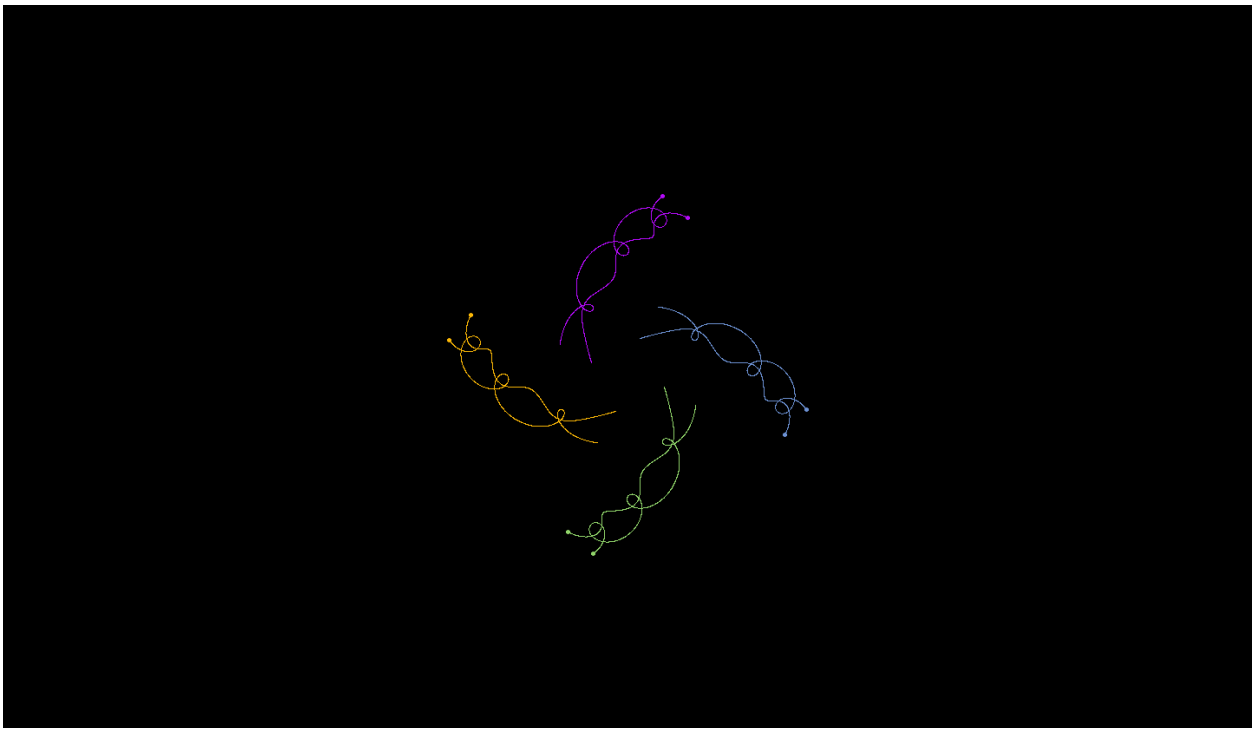

Figure 17 Trajectories of objects in the same eight-body system as in Figure 16. the degree to which the movements of these bodies are intertwined with each other motivates the insolvability of n-body problems

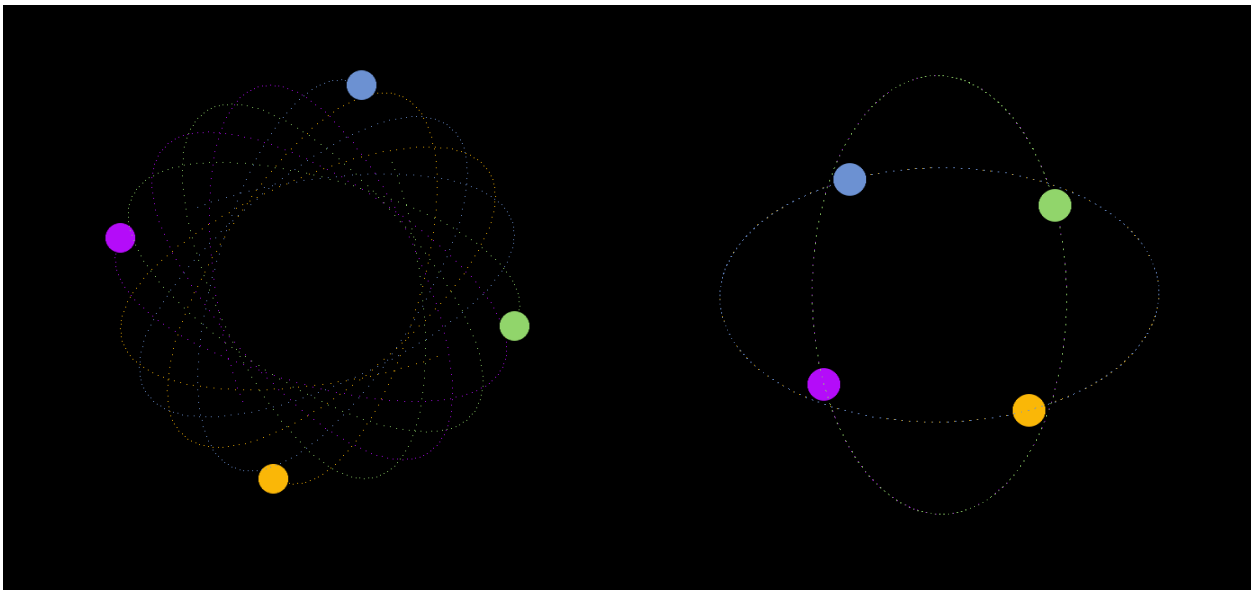

Figure 18 Trajectories of the system's bodies when $F \propto r^2$

Figure 19 Trajectories of the system's bodies when $F \propto r$; note the closed orbits predicted by Bertrand

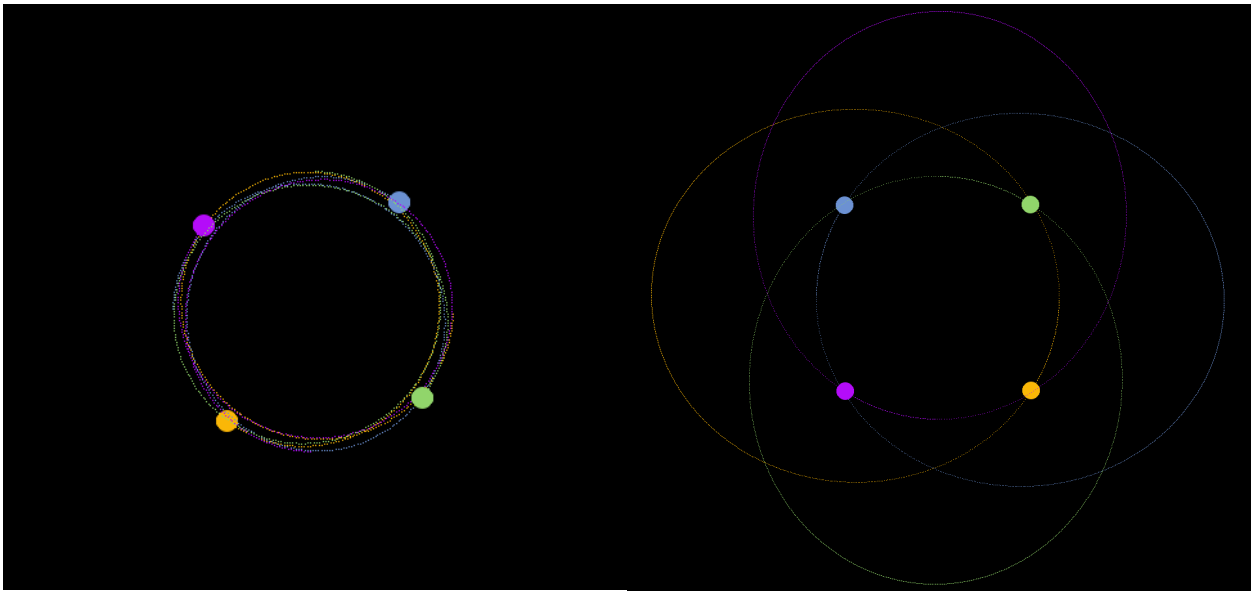

Figure 20 Trajectories of the system's bodies when $F \propto \frac{1}{r}$

Figure 21 Trajectories of the system's bodies when $F \propto \frac{1}{r^2}$; note the closed orbits predicted by Bertrand

Randomized Systems

Because of the wide variety of possible initial conditions for randomized systems, I'm reticent to claim any deep insight into these systems at this time. To make scientific claims in this realm would require more significant and systematic research than I've conducted with randomly-generated gravitational systems. However, I can make a qualitative (and unsurprising) observation: it seems that systems with more highly constrained initial positions and velocities have more interaction between the bodies.

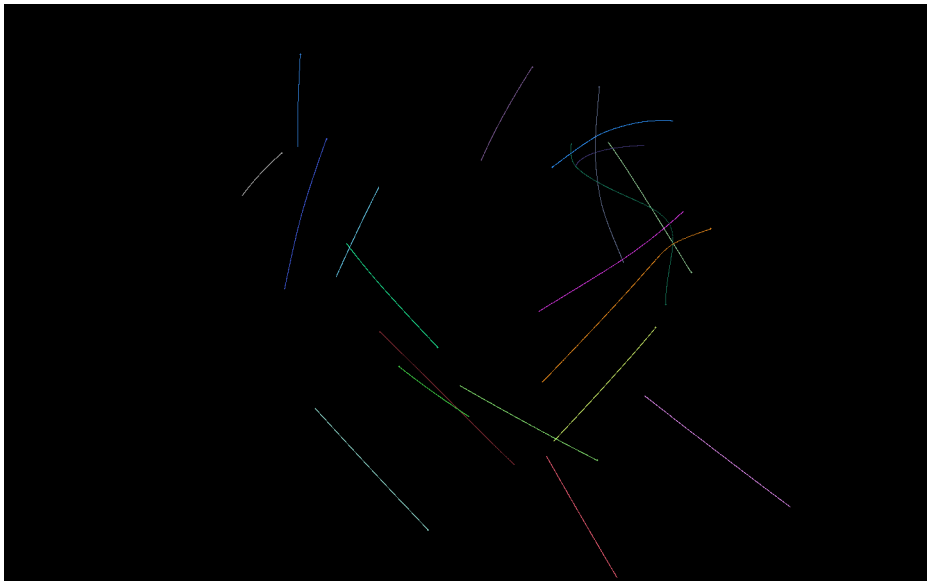

Figure 22 In this randomly generated system, the bodies interact significantly in a couple of areas but otherwise ignore each other

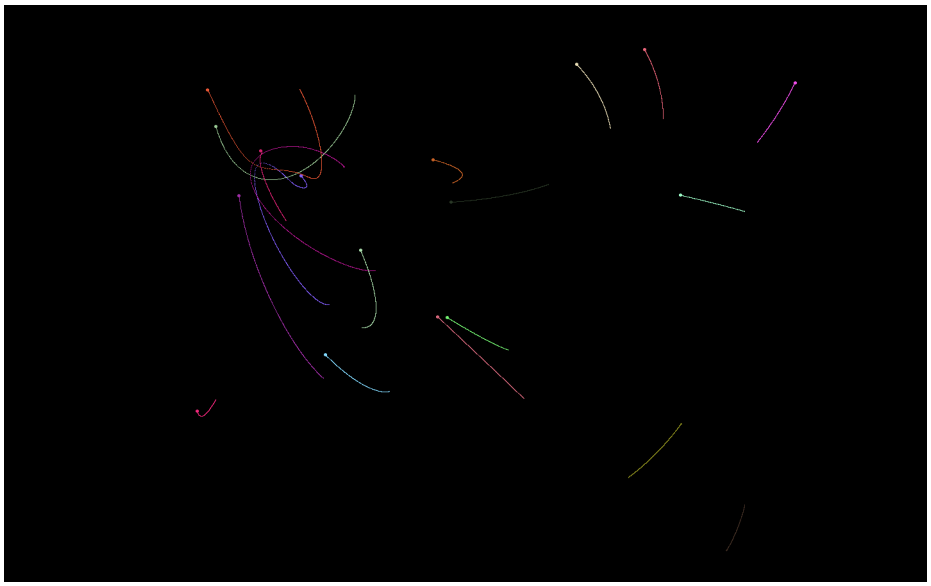

Figure 23 This randomly generated system is constrained to the middle half of position and velocity options used in Figure 22, keeping everything closer together and slower; this appears to have created more bound orbits

Conclusion

This program can effectively model n-body simulations in a wide variety of situations. It supports any central, conservative force laws so long as they don't cause mathematical overflow errors. Furthermore, it allows the user control over the reference frame and viewpoint, and can trace trajectories and gravitation potential lines.

This setup provides a powerful foundation for simulating and conceptualizing complex n-body problems. Although it does not currently lend itself to quantitative analysis – instead providing qualitative visual positions, trajectories, and gravitational potentials – small modifications would likely allow the program to make use of its robust internal mathematics, computed in useful units, in order to provide quantitative results.

The project's greatest strength remains close to its original motivation. Helping users visualize the motion of the solar system from the perspective of a particular planet encourages a better understanding of the history of astronomical models, justifying the physical appeal of a geocentric model and illustrating the ingeniousness of the heliocentric model.

Further Research

While this project covers a wide swathe of orbit modelling, it's still missing a number of key features and can be generally improved in several ways. In order to function as a learning tool, this program needs a significantly improved user interface, potentially incorporating a sandbox mode wherein the user can design their own system while the program is running (avoiding a tedious data input process) and can move these masses around with the cursor. Similarly, supporting a wider variety of pre-set system conditions would make the program more interesting and accessible.

Potential new features include the implementation of rotating bodies and rotating reference frames and a refinement of object collisions. The former would allow more accurate simulations, especially when generated from the perspective of a moving body. The latter would be a physical improvement, since no collisions are truly inelastic like this project assumes them to be.

Overall, some improvements in code structure could allow the faster generation of gravitational potential lines. Moreover, other developments – reducing the number of pixels flipped each frame and reducing the number of nested loops – could improve the overall speed of the simulation, allowing comparable runtimes with smaller values of dt and therefore allowing greater accuracy at the same run-speed.

Finally, randomly generated systems offer an interesting avenue for further research. One could analyze what factors are required to generate bound systems and refine models in order to recreate environments leading to the formations of solar systems.