POP 11g

Patrick Hartvigsen, Carl Dybdahl, Emil Søderblom

January 18, 2017

We have made an visual simulation of our solar system and compared the simulated data with the data from JPL Ephemeris.

1 Design and Architecture

1.1 Architecture

Figure 1 shows the UML diagram we made for this task.

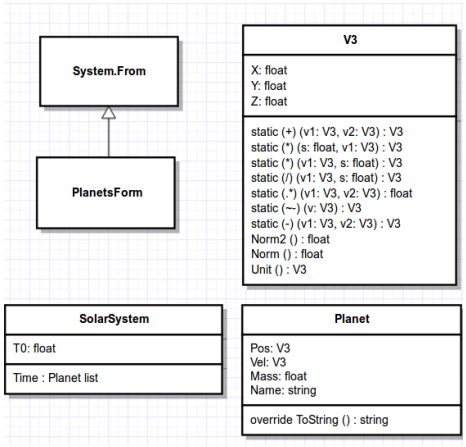


Figure 1: The UML diagram for our architecture.

To easily be able to calculate with 3d-vectors we made an vector class called `V3` with some generic vector functions.

To be able to store and construct planets, we made an Planet class where its posistion and velocity are represented as vectors. The class is immutable, so to simulate the solar system we construct new instances of the planets.

The `PlanetsForm` class inherits from `System.Form` and initilizes a form that we use to display our solar system. This is neccessary to access some attributes of the `System.Form` that we wished to change.

To find the posistion of the planets in a solar system, we made made a class called `SolarSystem`. It is able to simulate the movement of the planets behind the scenes and exposes a nice, simple interface that allows one to look up the position of the planets at any time.

1.2 Algorithms

For the task, we were only asked to do a first-order approximation of the movement of the planets, where we compute the acceleration and update the positions and velocities according to the following rules:

$$p(t_{n+1}) = p(t_n) + \Delta t \cdot v(t_n) \tag{1}$$

$$v(t_{n+1}) = v(t_n) + \Delta t \cdot a(t_n) \tag{2}$$

The precision of the simulation can, however, be significantly improved by using a second-order equation. For example, equation 1 can be improved with the following term:

$$p(t_{n+1}) = p(t_n) + \Delta t \cdot v(t_n) + \frac{1}{2} \cdot \Delta t^2 \cdot a(t_n) \tag{3}$$

The difference between equation 1 and 3 is that 1 approximates the paths of the planets as locally linear, whereas 3 approximates them as locally quadratic. Since the paths are curved, this approximation is much better in the long run.

In practice, however, the improvement does not make much of a difference, as equation 2 has the same imprecision as equation 1. In order to get a true improvement, we compute the jerk, that is, the derivative of acceleration, so that we can also improve equation 2. This gives us the following equation:

$$v(t_{n+1}) = v(t_n) + \Delta t \cdot a(t_n) + \frac{1}{2} \cdot \Delta t^2 \cdot j(t_n) \tag{4}$$

We wanted to compute the jerk using forward-mode automatic differentiation, but due to a limited type system in F♯ this was not easily supported. Instead, we manually did the computations that an automatic differentiation algorithm would do.

In addition to this improvement in precision, we also allowed the sun to move and computed the effects of gravity from all bodies, not just the sun.

2 Comparison with JPL Ephemeris

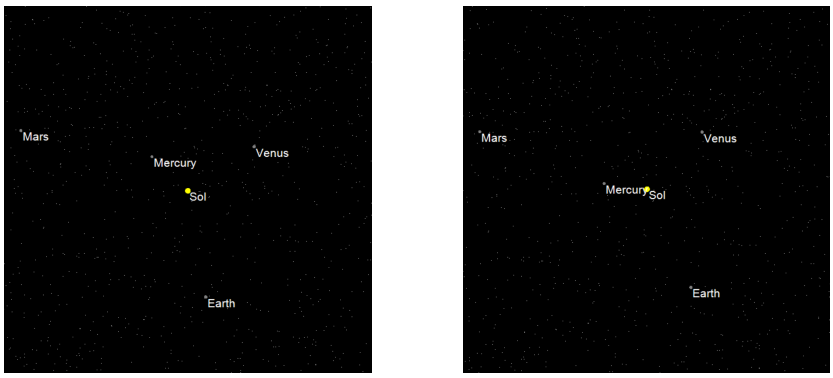
We wrote a program, `JPLECompare`, which makes a table that compares our simulator to the JPL Ephemeris data. It outputs `csv`-formatted data to the standard output. We ran the program and produced the table `jplecompare.csv`.

While our results are not the exact same as JPL Ephemeris's data, only Mercury truly ends up out of sync with the rest of the solar system; some of the other planets end up in slightly different positions than they really should have, but rather than ending up with an entirely different orbit, they seem to merely be at a slightly different point in their orbit.

Figure 2 contains a visual comparison showing that we don't lose much precision after 54 years of simulation.

3 Defects

Because we compute the way each planet affects every other planet, our program is rather slow.



(a) Our results after 54 years of simulation. (b) JPL Ephemeris in 2016

Figure 2: Comparison between our results, having simulated from 1962 to 2016, and JPL Ephemeris' results for the inner solar system. Sol is placed in the middle of each picture.