Internationella Kunskapsgymnasiet **May 12, 2011**

Interactive two-dimensional n-body simulation on gravitational force

Matematik breddning: Mathematical modeling

Author Viktor Qvarfordt
Examiner Charles Sacilotto

1 Description of program

This program is a basic n-body simulator, it simulates the gravitational force exerted between multiple bodies in two dimensions.

New ‘planets’ are produced by pressing and holding down the left mouse button at the location where the planet is to be created. Initial velocity is controlled by dragging the mouse while holding down the left mouse button, this produces the planets velocity vector. The planet is created when the button is released. All planets are cleared by pressing space.

Velocity is measured in distance (pixels) per second, ie the length of the initial user-created velocity vector is the distance the planet will travel per second. A trail is drawn behind all planets, to help visualize the planetary orbits.

Both velocity and acceleration is visually shown as vectors originating in each planets center, drawn in blue and red respectively. (They are also visually scaled down by a factor of 4 and 8.)

If two planets start overlapping each other a perfectly inelastic collision will be simulated; the larger body (the one with the largest mass) will consume the smaller – momentum is conserved.

Further information on the program and technical details are found in the attached `README` file or by running the program with the help flag `--help`.

2 Description of calculations

The calculations are made in steps, that is, the planets are not progressing through their orbits continuously. At each step the current state of the system (mainly position, velocity and acceleration) is identified and used to calculate the next state.

By shortening the interval between each step a more accurate approximation to the motion of the system is acquired.

The calculations are based on Newtons's law of gravitation

$$F = G\frac{m_1 m_2}{d^2} \tag{1}$$

and the equations of motion.

At each step in the calculations, for each planet, the resulting sum of acceleration due to the gravitational force in relation to all other planets is calculated

$$a_{tot} = \sum_{i=0}^{n} G\frac{m_i}{d_i^2} \tag{2}$$

where m_i denotes the mass of the planet by which gravitational force is exerted, and d_i is the distance to the i:th planet, and n is the number of planets. Now that the acceleration is known, the relation

$$\frac{d^2x}{dt^2} = a \tag{3}$$

can be used to get the new position, x, as needed to progress the system one step further. However, because what is being solved is a n-body system, this cannot be solved exactly for all planets, known as the “n-body problem”.[2] Hence the calculations must be performed with numerical methods, this is why the simulation is made up of multiple small steps as earlier mentioned.

Solving second-order differential equations is a very inefficient task compared to methods existing for first order DEs. Thus the integration done when solving for x in (3) is split up into two firs-order DEs,

$$\frac{dv}{dt} = a \tag{4}$$

$$\frac{dx}{dt} = v \tag{5}$$

which will be solved for v and x sequentially.

In its simplest form an approximated solution can be found by using the Euler method, defined as

$$y_{n+1} = y_n + hy'_n \tag{6}$$

This means that for each calculation step the following will be calculated, after the acceleration has been calculated

$$v_{n+1} = v_n + a_n \cdot \Delta t \tag{7}$$

$$x_{n+1} = x_n + v_n \cdot \Delta t \tag{8}$$

If all planets have the internal variables `position, velocity, acceleration` and `mass`, and a list `planets` containing all active planet objects is created, an implementation of the calculation loop in pseudocode could read:

```
for elements in planets as planet1:
    planet1.acceleration = 0
    for elements in planets as planet2:
        if planet1 is not planet2:
            planet1.acceleration += G*planet2.mass/(planet2.pos-planet1.pos)^2
    planet1.velocity += planet1.acceleration * dt
    planet1.position += planet1.velocity * dt
```

Note that this example is incomplete, as for example the acceleration becomes a scalar value. Handling this is too technical for the purpose of demonstrating the method used, see the attached source code for details.

3 Optimization

While the shown method is simple and easy to implement, it suffers from low accuracy as the truncation error per step is very large. To yield acceptable results the step-size has to be far smaller than the computational power on a standard desktop computer allows.

Some improvements to the method can be done by using the mean value for the velocity:

$$v_{mean} = \frac{v(t) + v(t + \Delta t)}{2} = v(t) + \frac{a(t)}{2}\Delta t \tag{9}$$

using this in the expression for the position (8) gives a better approximation to the position

$$x_{n+1} = x_n + v_{mean}\Delta t \tag{10}$$

$$= x_n + v_n\Delta t + \frac{a_n}{2}(\Delta t)^2 \tag{11}$$

Although this modification greatly increases the accuracy of the approximation, too small time-steps are required.

Several methods have been developed for numerically solving differential equations with higher precision. One of the more stable methods is the Runge-Kutta method.

Runge-Kutta is essentially a collection of higher order approximations of solutions to ordinary DEs. (Technically the Euler method can be seen as the first-order Runge-Kutta method.) The fourth-order Runge-Kutta (RK4) is one of the most commonly applied, because of its high convergence relative the step-size.

The concept that the method is based around is that instead of simply making the step-size smaller, the direction of the step will be carefully chosen, so that a higher convergence is achieved. This is the main flaw in the Euler method – the method only evaluates derivatives at the beginning of the interval, at x_n. The RK4 method evaluates the derivatives at four points and calculates a weighted mean value of the derivatives at each of these point, this mean value is then used as the final derivative, the direction in which the next step will be performed.

Letting $y' = f(x, y)$, the standard fourth-order Runge-Kutta method takes the generalized form:

$$\begin{aligned} k1 &= f\left(x_n, y_n\right) \\ k2 &= f\left(x_n + \frac{h}{2}, y_n + \frac{hk_1}{2}\right) \\ k3 &= f\left(x_n + \frac{h}{2}, y_n + \frac{hk_2}{2}\right) \\ k4 &= f\left(x_n + h, y_n + hk_3\right) \\ y_{n+1} &= y_n + \frac{h}{6}(k1 + k2 + k3 + k4) \end{aligned}$$

References

[1] http://en.wikipedia.org/wiki/N-body_simulation, 2011-05-01

[2] http://en.wikipedia.org/wiki/N-body_problem, 2011-05-01

[3] http://farside.ph.utexas.edu/teaching/329/lectures/node35.html, 2011-05-01

[4] http://en.wikipedia.org/wiki/Euler_method, 2011-05-01

[5] http://www.lonesock.net/article/verlet.html, 2011-05-01

[6] http://en.wikipedia.org/wiki/Runge-Kutta_methods, 2011-05-01