The three-body problem simulation

Zhang Liang
DUT

1. Brief introduction

The three-body problem is a well-known nonlinear dynamic problem. Unlike two-body problem, there is no closed-form solution, so numerical methods are generally required. It is interesting and challenging to simulate the three-body motion because of its chaotic property.

2. Problem statement

Simulate the classic three-body problem by numerical method and render an animation of their motion. To give reference, the description of the three-body problem is quoted from Wikipedia:
The three-body problem is the problem of taking the initial positions and velocities of three point masses and solving for their subsequent motion as dictated by Newton's laws of motion and of universal gravitation

3. Theoretic model

3.1. The Three-body problem

The gravity of object j on object i can be calculated by $\overrightarrow{F_{ij}} = Gm_i m_j \frac{\overrightarrow{x_i}-\overrightarrow{x_j}}{|\overrightarrow{x_i}-\overrightarrow{x_j}|^3}$.

And the net force on object i is $\overrightarrow{F_i} = \overrightarrow{F_{ij}} + \overrightarrow{F_{ik}}$.

So the Newtonian equations of motion in this system are of the form:

$$\ddot{\overrightarrow{x_i}} = -Gm_j \frac{\overrightarrow{x_i} - \overrightarrow{x_j}}{\left|\overrightarrow{x_i} - \overrightarrow{x_j}\right|^3} - Gm_k \frac{\overrightarrow{x_i} - \overrightarrow{x_k}}{|\overrightarrow{x_i} - \overrightarrow{x_k}|^3}$$

3.2. The 4th order Runge-Kutta method

The initial value problem is specified as follow:

$$\begin{cases} \ddot{\vec{x}} = f(\vec{x}) \\ \dot{\vec{x}}(t_0) = \dot{\overrightarrow{x_0}} \\ \vec{x}(t_0) = \overrightarrow{x_0} \end{cases}$$

$\vec{x}$ and $\dot{\vec{x}}$ are the unknown function of time *t*, which we would like to approximate;

$\ddot{\vec{x}}$, the 2nd derivative at $\vec{x}$, is a function of $\vec{x}$;

At the initial time t_0, the corresponding $\vec{x}$ value is $\overrightarrow{x_0}$, and $\dot{\vec{x}}$ value is $\dot{\overrightarrow{x_0}}$;

The RK4 approximation of $\vec{x}(t_0 + h)$ and $\dot{\vec{x}}(t_0 + h)$ is determined as follow:

$$\overrightarrow{x_1} = \overrightarrow{x_0} + \frac{h}{6} \cdot (\overrightarrow{k_{11}} + 2\overrightarrow{k_{12}} + 2\overrightarrow{k_{13}} + \overrightarrow{k_{14}})$$

$$\dot{\overrightarrow{x_1}} = \dot{\overrightarrow{x_0}} + \frac{h}{6} \cdot (\overrightarrow{k_{21}} + 2\overrightarrow{k_{22}} + 2\overrightarrow{k_{23}} + \overrightarrow{k_{24}})$$

where

$$\overrightarrow{k_{11}} = \dot{\overrightarrow{x_0}}$$

$$\overrightarrow{k_{21}} = f(\overrightarrow{x_0})$$

$$\overrightarrow{k_{12}} = \dot{\overrightarrow{x_0}} + \overrightarrow{k_{21}} \cdot \frac{h}{2}$$

$$\overrightarrow{k_{22}} = f\left(\overrightarrow{x_0} + \overrightarrow{k_{11}} \cdot \frac{h}{2}\right)$$

$$\overrightarrow{k_{13}} = \dot{\overrightarrow{x_0}} + \overrightarrow{k_{22}} \cdot \frac{h}{2}$$

$$\overrightarrow{k_{23}} = f\left(\overrightarrow{x_0} + \overrightarrow{k_{12}} \cdot \frac{h}{2}\right)$$

$$\overrightarrow{k_{14}} = \dot{\overrightarrow{x_0}} + \overrightarrow{k_{23}} \cdot h$$

$$\overrightarrow{k_{24}} = f(\overrightarrow{x_0} + \overrightarrow{k_{13}} \cdot h)$$

4. Engineering detail

There are four modules designed for this demo program: user defined class, solver, GUI, render.

1) The GUI module is developed using Qt, see Fig. 1. The interface is simple, just some boxes and 2 buttons in the bottom.

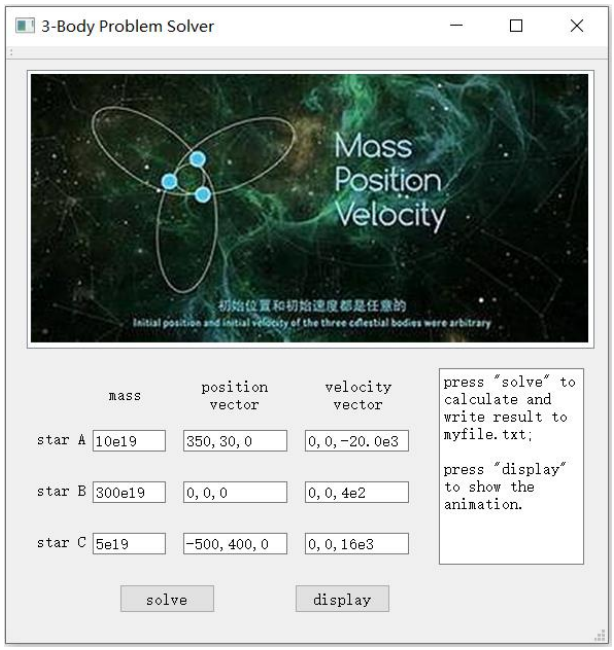

Fig. 1 Graphic user interface using Qt

2) A class named *DynamicStatus* is designed to describe the dynamic status of a point mass, see Fig. 2 (a), including the mass, position, velocity, acceleration, which can be calculated through methods defined in a class named *CalcClass*, see Fig. 2 (b).

```
DynamicStatus
Data members:
mass;           //scalar
position, velocity, acceleration; //3D vector
```

(a)

```
CalcClass
Function members:
Ftoa(force, DynamicStatus);    //Newton's 2nd law
RK4(DynamicStatus, deltaT); //4th order Runge-Kutta method
```

(b)

Fig. 2 User defined class

3) The most challenging part is the solver which can solve the initial condition problem of the differential equation. To realize this, the 4th order Runge-Kutta method (RK4) is applied, referring to the theory illustrated in 3.2. The control flow is just a simple loop of RK4 method.
4) The render is developed using OpenGL, see Fig. 3. It receives the 3 bodies' position data calculated by solver, and renders the animation in real-time.

Fig. 3 The animation rendered by OpenGL

5. Trouble shooting

5.1. The results do not converge

At first the data calculated by the solver using normal differential scheme can't converge. When two bodies get close, the distance gets very small, then the gravity becomes very large, and the velocity of the two bodies increase too fast, finally the three bodies fly away and never come back.

After investigation, the RK4 mothed was applied to handle this trouble. The data calculated by the RK4 solver was quite smooth, even when two bodies get close, the velocity didn't increase or decrease suddenly.

5.2. Even RK4 is applied, the three bodies still finally fly away

This is normal because the equation doesn't have a steady solution in most conditions. So in order to get an ornamental visual effect, the special initial conditions can be selected.