# PS3b: N-Body Simulation

Average time to complete assignment: $\sim$ 5 hours.

In Part B, we are adding physics simulation and animation to the program created in Part A.

- Your `CelestialBody` class should be extended so that the physics simulation can modify the velocities of each object.
- You must implement a method named `step` in the class `Universe` which takes a time parameter (`double seconds`) and moves the `CelestialBody` object given its internal velocity for that much time.
- You may internally represent forces, or you may calculate these from outside the `CelestialBody`.
- Your main routine should keep track of elapsed time and terminate the simulation when time has elapsed beyond the time limit provided at the command line.
- You **must** use smart pointers to manage the lifetimes of your `CelestialBody` objects.
- You **may** optionally (extra credit) display the elapsed time on the main screen.

## 1 Details

Your must build a command-line app which accepts the two command-line arguments $T$ and $\Delta t$ and reads the universe file from `stdin`. Name your executable `NBody`, so you would run it with e.g.

```
./NBody 157788000.0 25000.0 < planets.txt
```

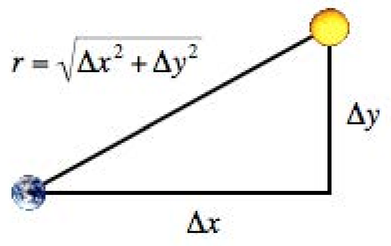
- Note that $T$ and $\Delta t$ are `double` command-line arguments.
- The universe is read from standard input (as in Part A).
- The universe format is the same as for Part A.
- Simulates the universe, starting at time $t = 0.0$ and continuing as long as $t < T$, using the *leapfrog* scheme described below.
- **After the animation stops**, your program should output the final state of the universe in the same format as the input. **Nothing else should be printed**.

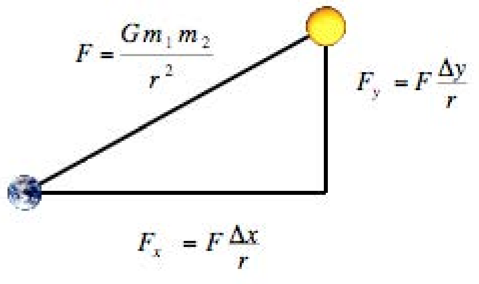
## 2 Simulating the Universe: Physics

We review the equations governing the motion of the particles, according to Newton's laws of motion and gravitation. Don't worry if your physics is a bit rusty; all of the necessary formulas are included below. We'll assume for now that the position $(p_x, p_y)$ and velocity $(v_x, v_y)$ of each particle is known. In order to model the dynamics of the system, we must know the net force exerted on each particle.

### 2.1 Pairwise Force

*Newton's law of universal gravitation* asserts that the strength of the gravitational force between two particles is given by the product of their masses divided by the square of the distance between them, scaled by the gravitational constant $G$ ($6.67 \times 10^{-11} \mathrm{N\,m^2/kg^2}$). The pull of one particle towards another acts on the line between them. Since we are using Cartesian coordinates to represent the position of a particle, it is convenient to break up the force into its $x$ and $y$-components $(F_x, F_y)$ as illustrated below.




$F = \frac{G m_1 m_2}{r^2}$
$F_y = F\frac{\Delta y}{r}$
$F_x = F\frac{\Delta x}{r}$

## 2.2 Net Force

The *principle of superposition* says that the net force on a particle in the $x$ or $y$ direction is the sum of the pairwise forces acting on the particle in that direction.

## 2.3 Acceleration

*Newton's second law of motion* postulates that the accelerations in the $x$ and $y$ directions are given by $a_x = F_x/m$ and $a_y = F_y/m$.

## 3 Simulating the Universe: Numerics

We use the *leapfrog finite difference approximation scheme* to numerically integrate the above equations. This is the basis for most astrophysical simulation of gravitational systems. In the leapfrog scheme, we discretize time and update the time variable $t$ in increments of the *time quantum* $\Delta t$ (measured in seconds). We maintain the position $(p_x, p_y)$ and velocity $(v_x, v_y)$ of each particle at each time step. The steps below illustrate how to evolve the positions and velocity of the particles.

1. For each particle:
   (a) Calculate the net force $(F_x, F_y)$ at the current time $t$ acting on that particle using Newton's law of gravitation and the principle of superposition. Note that force is a vector (i.e. it has a direction). In particular, be aware that $\Delta x$ and $\Delta y$ are signed (positive or negative). In the diagram above, when you compute the force the sun exerts on the earth, the sun is pulling the earth up ($\Delta y$ positive) and to the right ($\Delta x$ positive).

   (b) Calculate its acceleration $(a_x, a_y)$ at a time $t$ using the net force and Newton's second law of motion: $a_x = F_x/m$, $a_y = F_y/m$.

   (c) Calculate its new velocity $(v_x, v_y)$ at the next time step by using the acceleration computed in Step 2a and the *old velocity*. Assuming the acceleration remains constant in this interval, the new velocity is $(v_x + \Delta t \cdot a_x, v_y + \Delta t \cdot a_y)$.
2. Calculate the new position $(p_x, p_y)$ of each particle at time $t + \Delta t$ by using the *new velocity* and the *old position*. Assuming the velocity remains constant in this interval, the new position is $(p_x + \Delta t \cdot v_x, p_y + \Delta t \cdot v_y)$.
3. For each particle, draw it using the new position.

The simulation is more accurate when $\Delta t$ is very small, but this comes at the price of more computation.

## 4 Unit Tests

You must add additional unit tests to validate the behavior of the `step` method. The reference implementation has the same additional functions in part (b) as in part (a).

Note that while the `position` and `velocity` getters return `Vector2f`s, you should do your work internally with double precision rather than single precision.

One of the flawed implementations is broken in a very subtle way. Detecting it will require your own input data. Correctly identifying it is worth extra credit.

## 5 Extra Credit

You can earn extra credit by drawing the elapsed time in the simulation. Be sure to include units. Finally, you can create your own universe input file. If you do any of the extra credit work, make sure to describe exactly what you did in `Readme-ps3.md`. Be sure to include any files in your submission.

## 6 What to turn in

Your makefile should build a program named `NBody` and a static library `NBody.a`.

Submit a zip archive to Blackboard containing:

- Your main file `main.cpp`.
- Your Universe (`Universe.cpp`, `Universe.hpp`) and CelestialBody (`CelestialBody.cpp`, `CelestialBody.hpp`) classes.
- The makefile for your project. The makefile should have targets `all`, `NBody`, `NBody.a`, `lint` and `clean`. Make sure that all dependencies are correct.
- Your `Readme-ps3.md` that includes
  1. Your name
  2. Statement of functionality of your program (e.g. fully works, partial functionality, extra credit)
  3. Key features or algorithms used (updated for part b)
  4. Any other notes
- Any other source files that you created.
- Any images or other resources not provided (such as for the extra credit). The grader will provide images in the *base* directory, not a sub-directory.
- A screenshot of program output

Make sure that all of your files are in a directory named `ps3b` before archiving it and that there are no `.o` or other compiled files in it.

## 7 Grading rubric

| Feature | Points | Comment |
|---|---|---|
| Unit Tests | 5+1 | |
| | 2 | Computes acceleration correctly |
| | 2 | Velocity updates correctly |
| | 1 | Order of operations |
| | 1 | XC one |
| Autograder | 19 | Full & Correct Implementation |
| | 4 | Contains all code files and builds |
| | 4 | Has working >> and << operators from Part A |
| | 8 | `step()` works as expected |
| | 2 | Obeys Newton's third law |
| | 1 | Precision |
| Drawing | 10 | |
| | 3 | Draws the universe correctly |
| | 4 | Updates UI after each step |
| | 3 | Planets move in the correct direction (counter-clockwise for `planets.txt`) |
| Simulation | 9 | |
| | 4 | Simulation stops after the elapsed time |
| | 3 | Prints the final state of the universe when the universe closes |
| | 2 | Uses smart pointers |
| Screenshot | 2 | |
| Readme | 5 | Complete |
| | 2.5 | Describes the algorithms or data structures used. |
| | 2.5 | Describes how the features were implemented. |
| Extra Credit | 9 | |
| | +2 | Shows elapsed time (with units) |
| | +3 | Creates a new universe file and describes in readme. |
| Penalties | | |
| | -5 | Memory leaks or issues |
| | -5 | Linting problems |
| | -3 | Non-private fields |
| | -10% | Each day late |
| Total | 50 | |

## 8 Testing and debugging

Below are the outputs your program should print after running on the `planets.txt` for various lengths of time. **Make sure you are printing your output at the very end of the program before comparing to these.** The exact numbers you get may be slightly different than the ones below if you do your computations in a slightly different order than our solution. **This is fine**, and is caused by rounding errors that affect the results differently depending on the precise order of the statements. **The biggest errors will appear to be in the sun's position.** Don't worry; all the values are printed in scientific notation. The planets' positions all end in `e+10` or `e+11` meaning you should multiply the numbers by $10^{10}$ or $10^{11}$. The exponents on the sun's position are on the scale of $10^5$. An error in the leading digit of the sun's position is therefore miniscule compared to the planet positions. Another way to think about this is that if the window is $512 \times 512$ pixels, representing coordinates ranging from $-2.5 \times 10^{11}$ to $2.5 \times 10^{11}$. Even if the leading digit of the sun's position is off by one, the error is on the order of 10000, which is far less than that one pixel in the window!

Inserting break points is a good way to trace what your program is doing. Here are the results for a few sample inputs.

```
// zero steps
5
2.50e+11
1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e+24  earth.gif
```

```
2.2790e+11  0.0000e+00  0.0000e+00  2.4100e+04  6.4190e+23  mars.gif
5.7900e+10  0.0000e+00  0.0000e+00  4.7900e+04  3.3020e+23  mercury.gif
0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00  1.9890e+30  sun.gif
1.0820e+11  0.0000e+00  0.0000e+00  3.5000e+04  4.8690e+24  venus.gif
```

```
// one step
5
2.50e+11
1.4960e+11  7.4500e+08  -1.4820e+02 2.9800e+04  5.9740e+24  earth.gif
2.2790e+11  6.0250e+08  -6.3860e+01 2.4100e+04  6.4190e+23  mars.gif
5.7875e+10  1.1975e+09  -9.8933e+02 4.7900e+04  3.3020e+23  mercury.gif
3.3087e+01  0.0000e+00  1.3235e-03  0.0000e+00  1.9890e+30  sun.gif
1.0819e+11  8.7500e+08  -2.8329e+02 3.5000e+04  4.8690e+24  venus.gif
```

```
// two steps
5
2.50e+11
1.4959e+11  1.4900e+09  -2.9640e+02 2.9799e+04  5.9740e+24  earth.gif
2.2790e+11  1.2050e+09  -1.2772e+02 2.4100e+04  6.4190e+23  mars.gif
5.7826e+10  2.3945e+09  -1.9789e+03 4.7880e+04  3.3020e+23  mercury.gif
9.9262e+01  2.8198e-01  2.6470e-03  1.1279e-05  1.9890e+30  sun.gif
1.0818e+11  1.7499e+09  -5.6660e+02 3.4998e+04  4.8690e+24  venus.gif
```

```
// three steps
5
2.50e+11
1.4958e+11  2.2349e+09  -4.4460e+02 2.9798e+04  5.9740e+24  earth.gif
2.2789e+11  1.8075e+09  -1.9158e+02 2.4099e+04  6.4190e+23  mars.gif
5.7752e+10  3.5905e+09  -2.9682e+03 4.7839e+04  3.3020e+23  mercury.gif
1.9852e+02  1.1280e+00  3.9705e-03  3.3841e-05  1.9890e+30  sun.gif
1.0816e+11  2.6248e+09  -8.4989e+02 3.4993e+04  4.8690e+24  venus.gif
```

```
// one year (365.25 days or 31557600 seconds)
5
2.50e+11
1.4959e+11  -1.6531e+09 3.2949e+02  2.9798e+04  5.9740e+24  earth.gif
-2.2153e+11 -4.9263e+10 5.1805e+03  -2.3640e+04 6.4190e+23  mars.gif
3.4771e+10  4.5752e+10  -3.8269e+04 2.9415e+04  3.3020e+23  mercury.gif
5.9426e+05  6.2357e+06  -5.8569e-02 1.6285e-01  1.9890e+30  sun.gif
-7.3731e+10 -7.9391e+10 2.5433e+04  -2.3973e+04 4.8690e+24  venus.gif
```