PS3a: N-Body Simulation (static)

Due: Thursday, June 2, 11:59pm

In 1687, Sir Isaac Newton formulated the principles governing the motion of two particles under the influence of their mutual gravitational attraction in his famous *Principia Mathematica*. However, Newton was unable to solve the problem for three particles. Indeed, in general, solutions to systems of three or more particles must be approximated via numerical simulations. Your challenge is to write a program to simulate the motion of n particles in the plane, mutually affected by gravitational forces, and animate the results. Such methods are widely used in cosmology, semiconductors, and fluid dynamics to study complex physical systems. Scientists also apply the same techniques to other pairwise interactions including Coulombic, Biot-Savart, and van der Waals.

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

1 Details

For Part A of this assignment, we will create a program that loads and displays a static universe. In Part B, we will add the physics simulation and animate the display.

- Make sure to download the universe specification files and images files from `nbody.zip`
- You should build a command-line app which reads the universe file (e.g. `planets.txt`) from `stdin`. Name your executable `NBody`, so you would run it with e.g.

```
./NBody < planets.txt
```

The `< planets.txt` construct is known as an *input redirect*.

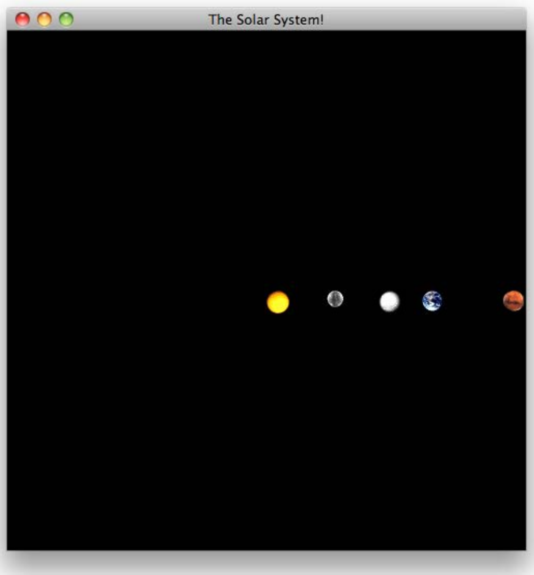
1.1 Reading in the universe

The input format is a text file that contains the information for a particular universe (in SI units). The first value is an integer n which represents the number of particles. The second value is a real number r which represents the radius of the universe, used to determine the scale of the drawing window. Finally, there are n rows and each row contains 6 values. The first two values are the x and y coordinates of the initial position; the next pair of values are the x and y components of the initial velocity; the fifth value is the mass; the last value is a `string` that is the name of an image used to display the particle. As an example, `planets.txt` contains data for our own solar system (up to Mars):

```
...xpos...  ...ypos...  ...xvel...  ...yvel...  ...mass...  filename
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
```

You should read in exactly as many rows of body information as are indicated by n, the first value in the file.

The `planets.txt` universe file contains the Sun and the first four planets, with the Sun at the center of the universe ($x = 0$, $y = 0$) and the four planets in order toward the right (as below). When this is working, you should be rewarded with:



For this project, you'll implement a `Universe` class which will contain all celestial bodies, and a `CelestialBody` class representing the celestial bodies.

- The class `Universe` should create `CelestialBodys`.
- The `CelestialBody` should have the following features:
  - It must be `sf::Drawable`, with a `private virtual void` method named `draw` (as required of `sf::Drawable` subclasses).
  - Each instance of the class should contain all properties needed for the simulation (e.g. x and y position, x and y velocity, mass, and image data).
  - It may contain a `sf::Sprite` object (as well as the `sf::Texture` object needed to hold the `Sprite`'s image).
- You **must** override the input stream operator >> for both `Universe` and `CelestialBody` and use it to load parameter data into an object.
- You **must** override the output stream operator << for both `Universe` and `CelestialBody` and use it to write the state back out in the same format as the input.

2 Extra Credit

You can earn extra credit by drawing a background image. This image should appear *behind* all of the planets. If you do any of the extra credit work, make sure to describe exactly what you did in `Readme-ps3a.md`.

3 Development Process

There are a lot of parts to this assignment. We'd suggest the following incremental development process:

1. Create a bare-bones version of your `Universe` class.
   (a) Override the << operator to read just the number of planets and universe radius.
   (b) Override the >> operator to print the information back.
   (c) Verify that both work as intended.
2. Create a bare-bones version of your `CelestialBody` class.
   (a) Override the << operator to read the planet data.
   (b) Override the >> operator to print the information back.
   (c) Read one planet from the file and verify that it works as intended.

3. Setup the main draw loop in your main file.
    - Hint: Your class will need to know both the universe and viewport window dimensions.
    - Hint: The universe's center is (0, 0), but for SFML, (0, 0) is in the upper-left corner.
4. Implement the `draw` method in the `CelestialBody` class. Draw this in your loop.
5. Move control of the `CelestialBody` into the `Universe`. Implement the `draw` method in your `Universe` class.
6. Read the rest of the planets into your `Universe`.

4 What to turn in

Your makefile should build a program named `NBody`.

Submit a tarball 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`, `lint` and `clean`. Make sure that all prerequisites are correct.
- Your `Readme-ps3a.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
    4. Any other notes
- Any other source files that you created.
- Any images or other resources not provided (such as for the extra credit)
- A screenshot of program output

Make sure that all of your files are in a directory named `ps3a` before archiving it and that there are no `.o` or other compiled files in it.

5 Grading rubric

Feature	Points	Comment
Core Implementation	10	Full & Correct Implementation
	2	Contains all code files and builds
	1	Contains `Universe` and `CelestialBody` classes
	1	Both classes are `Drawable` with *private* `draw()` methods
	1	Loads universe from `stdin`
	2	Draws the planets to the screen correctly
	1	Overrides `<<` and `>>` operators for both classes
	1	Supports arbitrary number of `CelestialBody` objects
	1	Scales to different universe sizes
Makefile	2	
	1	Has targets `NBody`, `lint`, `all`, and `clean`
	1	`all` builds `NBody`
Screenshot	1	
Readme	2	Complete
	1	Describes the algorithms or data structures used.
	1	Describes how the features were implemented.
Extra Credit	1	
	+1	Shows background image
Penalties		
	-2	Linting problems
	-1	Non-private fields
	-1	Submission includes `.o` files
	-1	Submission not in a directory named `ps3a`
	-10%	Each day late
Total	15	