ASSIGNMENT 3

Due on April 27, 2017 (23:59:59)

1 Introduction

In this assignment, you will implement N-body simulation(see Figure 1) using a brute-force and a divide and conquer algorithm in Part 1 and Part 2 respectively.

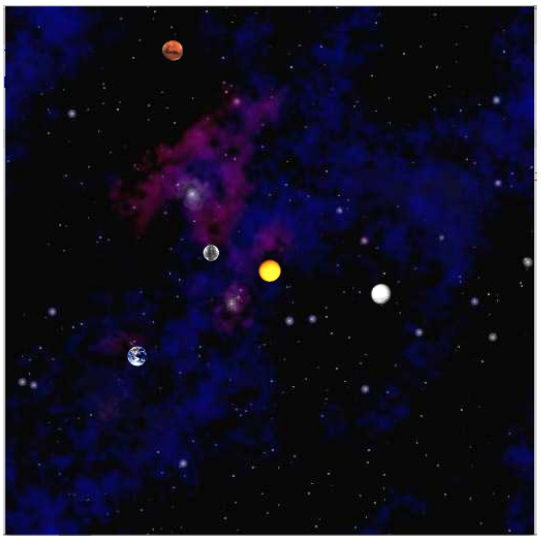


Figure 1: A screenshoot from N-Body Simulation of *planets.txt* universe.

2 Background

In 1687, Isaac Newton introduced his famous laws of motion that describe the relationship between a mass and the forces exerted to it and motion arose after the interaction of the forces with the mass. In this assignment, you will simulate the motions of N bodies in 2D plane by calculating gravitational forces exerted to each other using laws of universal gravitation. There many applications use this simulation methods in cosmology, semiconductors, fluid dynamics and complex physical systems.

Briefly you will write a Java program NBody.java which:

- Reads three command-line arguments *brute* or *quad* as algorithm types and two double arguments T and Δt .
- Reads in the universe from standard input using StdIn, using several parallel arrays to store the data.

This assignment is adapted from the programming assignment of COS126 course of Computer Science Department of Princeton University[1]

- Simulates the universe, starting at time t = 0.0, and continuing as long as t ¡ T, using the leapfrog scheme described below.
- Animates the results to standard drawing using StdDraw.
- Prints the state of the universe at the end of the simulation (in the same format as the input file) to standard output using StdOut.

3 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 scaling 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 file or color value used to display the particle. As an example, planets.txt contains data for our own solar system (up to Mars):

planets.txt

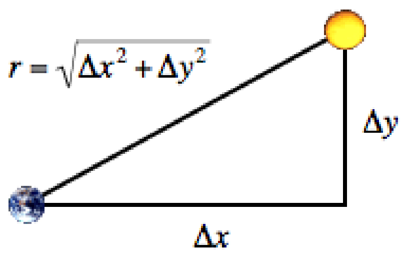
```
5
2.50e+11
1.4960e+11  0.0000e+00  0.0000e+00  2.9800e+04  5.9740e+24  earth.gif
2.2790e+11  0.0000e+00  0.0000e+00  2.4100e+04  6.4190e+23  mars.gif
5.7900e+10  0.0000e+00  0.0000e+00  4.7900e+04  3.3020e+23  mercury.gif
0.0000e+00  0.0000e+00  0.0000e+00  0.0000e+00  1.9890e+30  sun.gif
1.0820e+11  0.0000e+00  0.0000e+00  3.5000e+04  4.8690e+24  venus.gif
```

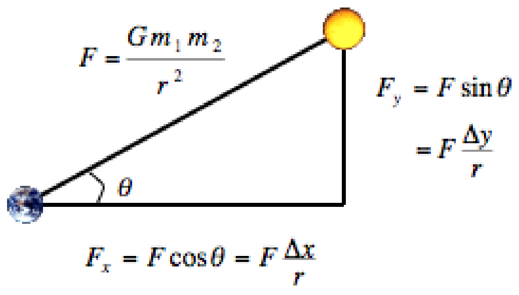
4 The Physics

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.

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 ($6.67 \times 10^{-11} Nm^2/kg^2$). 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.

Net force. The principle of superposition says that the net force acting on a particle in the




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

x- or y-direction is the sum of the pairwise forces acting on the particle in that direction.

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

5 Part 1: Simulating the universe with Brute-Force algorithm(40 Points)

You will use a brute force algorithm in Part 1 to implement the above equations: this is the basis for gravitational systems. In simulation, we discretize time, and update the time variable t in increments of the time quantum Δt (measured in seconds). Your program will update 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 update the positions and velocities of the particles.

- **Step 1.** For each particle: 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 direction). In particular, be aware that Δx and Δy 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 (Δy positive) and to the right (Δx positive).
- **Step 2.** For each particle:
  1. Calculate its acceleration (a_x, a_y) at time t using the net force computed in Step 1 and Newton's second law of motion: $a_x = F_x/m$, $a_y = F_y/m$.
  2. Calculate its new velocity (v_x, v_y) at the next time step by using the acceleration computed in Step 2.1 and the velocity from the old time step: Assuming the acceleration remains constant in this interval, the new velocity is $(v_x + \Delta t a_x$, $v_y + \Delta t a_y)$.
  3. Calculate its new position (p_x, p_y) at time $t + \Delta t$ by using the velocity computed in Step 2.2 and its old position at time t: Assuming the velocity remains constant in this interval, the new position is $(p_x + \Delta t v_x, p_y + \Delta t v_y)$
- **Step 3.** For each particle: Draw it using the position computed in Step 2.

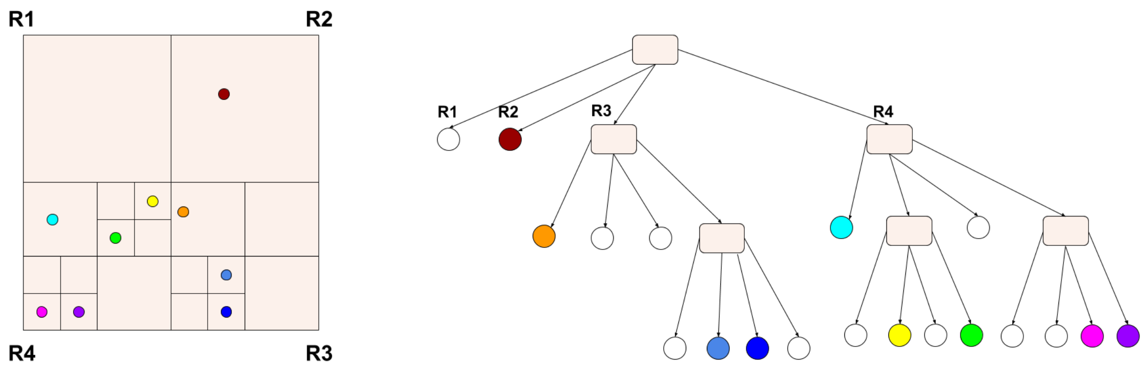


Figure 2: A Quad tree illustration.

6 Part 2: Simulating the universe with Quad Tree algorithm(60 Points)

In Part 1, a brute force solution is used to compute mutual forces for each body, so force calculations per step is proportional to N^2. When N is large, this approach is not practical. In Part 2, you will implement Barnes-Hut algorithm on quad tree data structure.

Barnes-Hut Algorithm. The main idea of Barnes-Hut algorithm is to consider nearby bodies as a single body by grouping them. If a group is sufficiently far away from a body, graviational effects can be calculated as if that group is a single body with center of mass of the group. The center of mass of a group of bodies can be calculated averaging position of a body in that group, weighted by mass. Formally, position of center of mass (x, y) can be calculated as follows for two bodies with masses m_1 and m_2 with total mass m:

$$m = m_1 + m_2$$

$$x = (x_1m_1 + x_2m_2)/m$$

$$y = (y_1m_1 + y_2m_2)/m$$

This grouping scheme is carried out using quad tree representation of 2D space. The whole space is divided recursively according to distribution of bodies to space. Quad tree is a data structure that is similar to binary search tree except that each node has four children instead of two children. Each node represents a region of the two dimensional space. The topmost node represents the whole space, and its four children represent the four quadrants of the space. As can be seen in the diagram given in Figure 2, whole space recursively divided until each region contains 1 body at most. Therefore, internal nodes may have less than 4 non-empty children.Each internal node represent a group with center of mass and total mass of the bodies are included in it and each external node represent a single body or empty space.

To obtain net force for each body on quad tree reprsentations is quite easy task. For a body, net force is calculated by traversing the quad tree nodes starting from the root. If the center-of-mass of an internal node is sufficiently far from the body, consider the bodies contained in that part of the tree as a single body, whose position is the group's center of mass and whose mass is the group's total mass. After that, we do not need to individually calculate gravitational forces of bodies in the group, so we have more faster algorithm. We can determine if a node is sufficiently far away by checking ratio r/d, where r is the width of the region represented by the internal node, and d is the distance between the body and the node's center-of-mass. Then by simply thresholding this ratio with value t, we can decide if the internal node is sufficiently far away or not. We can take this threshold value as $t = 0.5$. Note that, this value determine how our simulation will be faster or accurate. In summary, you will carry out the following steps:

1. Construct a quad tree.
2. Calculate net forces for each body
3. Calculte accelaration and velocities
4. Calculate new positions
5. Animate the motions of bodies

6.1 Construct a quad tree

You can simply adapt the Java code of binary search tree(BST) to quad tree by considering the following rules instead of BST's own rules to insert a body B into the tree rooted at node X recursively:

- If X does not contain a body, put the new body B here
- If X is an internal node, update the center-of-mass and total mass of X. Recursively insert the body B in the appropriate quadrant.
- If X is an external node, say containing a body named C, then there are two bodies B and C in the same region. Subdivide the region further by creating four children. Then, recursively insert both B and C into the appropriate quadrant(s). Since B and C may still end up in the same quadrant, there may be several subdivisions during a single insertion. Finally, update the center-of-mass and total mass of X.

6.2 Calculate net forces for each body

After inserting N bodies to a quad tree, you can start to animation loop calculating net forces accordingly accelaretions, velocities and positions. Similar to insertion scheme, you will follow the recursion steps for each body B:

- Start from the root node
- If the current node is an external node (and it is not body B), calculate the force exerted by the current node on B, and add this amount to B's net force.
- Otherwise, calculate the ratio r/d. If $r/d < t$, treat this internal node as a single body, and calculate the force it exerts on body B, and add this amount to B's net force.
- Otherwise, run the procedure recursively on each of the current node's children.

7 Getting Started

You are given some libraries and demos(StdIn, StdOut, StdDraw, etc.) to read input file and draw simulation on Java SE environment in Piazza Resources. You do not need to write these procedures from the begining.

Notes

- Save all your work until the assignment is graded.You can ask your questions about the experiment on Piazza.
- You will only submit your java source files in a folder in the given format below

  studentid.zip
  — – <src>
  — – — – NBody.java
  — – — – *.java

- You should test your code in department servers(*dev.cs.hacettepe.tr*)
- Your output format is exactly the same with input format.
- **Compiling and execution:**

```
javac NBody.java
java Nbody brute 157788000.0 25000.0 < input.txt > output.txt
java NBody quad 157788000.0 25000.0 < input.txt > output.txt
```

Policy

All work on assignments must be done individually unless stated otherwise. You are encouraged to discuss with your classmates about the given assignments, but these discussions should be carried out in an abstract way. That is, discussions related to a particular solution to a specific problem (either in actual code or in the pseudocode) will not be tolerated. In short, turning in someone elses work(from internet), in whole or in part, as your own will be considered as a violation of academic integrity. Please note that the former condition also holds for the material found on the web as everything on the web has been written by someone else.

References

1. http://www.cs.princeton.edu/courses/archive/spr15/cos126/assignments/nbody.html