

Assignment 3, Philipp Noel von Bachmann

March 2, 2022

1 Problem

In the N-body Problem, we want to predict the movement of n bodies that interact with each other over time. The interaction is given by the force they exert on each other, which in turn changes the acceleration of each body, thus the velocity and position. The problem is modeled by assuming pairwise interaction between the bodies and taking their sum as the total force exerted onto each body.

In this exercise, we use an algorithm known as the Barnes-Hut. This algorithm reduces the computational cost by approximating a group of stars that are far away as one single big star. Together with a representation known as a quad-tree, this enables us to reduce the computational complexity significantly.

2 Solution

2.1 Data structure

2.1.1 Body

Like in the previous assignment, we have the following 4 characteristics of each body:

- velocity
- position
- mass
- brightness

Mass and brightness are represented as double, since they are both continuous. As position and velocity are both vectors, they are represented as a struct of coordinates. Since we model the problem in 2D, each vector thus comprises of an x and y coordinate. The characteristics are collected in a struct called `body`. This format also allows for easy reading of the input `gal` file, since we can directly read each block into a `body` struct.

In later parts ??, we also keep track of the accelerations of each body in an array of coordinates. We opt not to integrate this into the `body` struct to be able to retain easy writing and reading of files.

2.1.2 Quadtree

A quadtree is a tree of nodes, where every node represents a subset of an area of interest. Each node can be either an internal or external node.

An external node is a node in whose area there is only one body. If in the area of a node contains more than one body, it is called an internal node. Instead of having an body that belongs to this node, the node has 4 child nodes which represent the 4 subareas of this nodes area with their correpsonding bodies. To construct a quadtree, we therefore split all nodes recursively until each leaf node is an external node. A node whose area contains no bodies is also considered a external node.

In general, we represent a node in a structure with the following attributes:

- lower: coordinate representing the lower edge of the area covered by the node
- upper: coordinate representing the upper edge of the area covered by the node
- children: A list containing a pointer to each of the children nodes, if there are any, else a NULL pointer.
- body: A pointer to the body contained by this node if an external node, else a NULL pointer.
- total mass: A float representing the total mass of the bodies in the area of this node.
- center of mass: A coordinate, representing the center of all bodies contained by this node weighted by their mass.

2.2 Structure

The baseline code consists of the following modules/functions:

- utility functions for coordinates: compute direction and compute norm
- compute force: computes the force that acts onto a body
- compute body update: computes the update step for each body
- step: handles the update step, calls the subroutines. Only this function actually changes the state of our bodys.

In the final code, we restructured it to the following, see mostly section ?? for details.

- utility functions for coordinates: compute direction and compute norm
- compute acceleration: computes the acceleration that acts onto a body
- update body: computes the update step for each body and applies it
- step: calls the subroutines and iterates over all bodys

3 Performance and Discussion

3.1 Hardware information

- CPU: Apple M1 with 4 performance cores and 4 efficiency cores
- GPU: Apple M1 8 core.
- Compiler: Slang, Version 13.0.0

3.2 Performance Measurement script

The performance was measured with a simple shell script to enable reproducability. In the script we vary the number of bodies. In the end we also check that the code is correct with the `compare_gals` file. Our time measurement was done within the code with the help of the time `timeoftheday` function. We only started the time measurement immediatly before calling the step loop and stopped the measurement directly after the loop.

3.3 Performance and Results

3.4 Calculation of θ_{\max}

We calculate θ_{\max} as described in the Assignment, Section 5.2. This results in $\theta_{\max} = 0.25$, where an `pos_maxdiff` = 0.000885759450 was achieved.

3.5 Baseline results

The following table shows the results of the algorithm with no optimization for a variable number of bodies:

steps	10	100	1000
time (s)	0.003	0.121	3.943

Compared to the last assignment, these times are better for the case of $N = 10$, but worse for higher numbers.

3.6 Analysis of runtime

This section analyzes the runtime of the two main parts of the algorithm: Building the tree and calculating the forces.

4 TODO

ideas: -

5 References

- For the timing, the `get_wall_seconds` from Lab06, exercise 7
- For data reading, the function `read_doubles_from_file` from `compare_gal_files`, but slightly modified