TP 11 : N-bodies problem

Cours de modélisation numérique

19 May 2023

Introduction

The Barnes-Hut algorithm, presented in the course, improves the performance of a simulation of N bodies interacting with each other, for example the stars of a galaxy. To do this, we approximate the groups of distant bodies as a single body. In practice, this approximation is based on the use of quaternion trees.

For this tutorial, the code `barneshut.py` is provided on Moodle. This code fully implements the 2-dimensional n-body problem approximated by the Barnes-Hut algorithm, as well as an instance of a problem and a way to visualize it. Take the time to understand this code and verify that it works.

Work to do

1. 3D generalization

Make the necessary modifications to the code given to you in order to generalize it to the tridimensional case.

Note that the original code already contains functions to visualize the result in 2D and 3D, named respectively `plot_bodies_2D` and `plot_bodies_3D`. The same applies to the initialization of impulses, with the functions `init_momentum_2D` and `init_momentum_3D`. You must use the latter to initialize the star pulses in the 3D case.

As an indication, if the implementation is correct, the Z coordinate of body number 0 should be, at iteration 500, $Z = 0.55682037$, with $N = 200$ body and all other parameters left as default.

2. Performance and time complexity study in the 3D case

Study the performance of the algorithm as a function of the number of stars (obviously, it is advisable to comment on the lines related to visualization for this part, as well as to average the times taken by several different measurements).

Implement a naive version of the algorithm, which handles the interaction of each star with all the others. To do this, in the main loop, instead of the function `verlet`, use the function `verlet_bruteforce`, which you will have implemented yourself and which corresponds to the naive vision described above. Don't forget, for performance measurements, to also comment out the lines relating to the reconstruction of the quadtree.

Present, on the same graph, the performances of the two algorithms (execution time versus the number of stars). Analyze your results. What can you say about the time complexity of each algorithm ?

The function `clock()` of the module `time` makes it possible to obtain the time elapsed since the importation of the module (Unix) or since the first call to the function (Windows). Thus, the following code stores in the variable `total time` the execution time of the function $\texttt{my}_{f}unction$:

```
import time
begin = time.clock()
ma_fonction()
end = time.clock()
temps_total = end - begin
```