Computational Project

Jimmy Chen

10th March 2024

Project B: Hierarchical methods for N-body simulation.

1 Foreword

This is the notes I made when I first started to learn about this project. It contains some resources I've found useful as I try to grasp what these algorithms do. Some things and ideas have changed since then, so this is **not meant to be an accurate description of the current state of the project**.

2 Barnes-Hut algorithm

See this link and this for helpful illustrations of the algorithm. Another useful guide is the original paper.

Also this makes it clear that we traverse the tree for each particle.

2.1 Description

2.1.1 Construction of the tree

The algorithm represents the whole of simulation space as an *octree*, dividing the space into 8 smaller cubes. We keep on dividing each sub-cell in this way into oct-subtrees, until each tree only contains one mass. We then update the nodes so that we have stored on each the centre of mass position, and total mass. We must also log the length of one side of the cube the tree represents, l.

2.1.2 Calculation of forces

We set a threshold $\theta \approx 0.5$. To calculate the force on each particle a, we traverse the tree. For each node B we arrive at:

- If B is leaf node containing a, skip (no self-interaction);
- If B is another leaf node, calculate the interaction;
- If $l/|x_B - x_a| < \theta$, where x_B is the CoM displacement, treat all mass in cell represented by B as a point particle and calculate interaction.

- Otherwise, go to the next level of the tree.

We note that this could be improved by introducing multipoles into each of the parent boxes.

3 Fast multipole method

Because of the mathematical rigour involved, the papers from the original authors (this) are not super accessible. Other resources are therefore necessary.

Advanced, quick introduction: IBM Research

Great series by NCTS: Rio Yokota, Fast Multipole Method

Dehnen has written a number of helpful papers on this; See Dehnen 2014, Dehnen 2000 and Dehnen 2002.

Cheng et al. 1999 is also highly useful, as it contains a procedural description of how the algorithm works. It does not employ dual tree traversal however.

3.1 Description

3.1.1 Mathematical description

Our task is to calculate potential ψ. For Newtonian gravitation in 3D, with the right units,

$$\psi = \frac{1}{|x - y|} \tag{1}$$

Where x, y are vectors. Forces can be derived by taking derivatives.

We assume a relatively homogeneous distribution of masses. We then split our cuboid of simulation space into octrees of cells. We continue splitting all the sub-regions simultaneously until each region contains merely $O(1)$ number of masses.

Cells are *sufficiently far away* from each other if they are not adjacent to each other.

3.1.2 Tree-building

We face the problem of needing to know what particles are in each box.

A first memory optimization would be to use a list `souls`, which stores pointers to actual particle `structs`, as these are larger in size and should be kept static.

A naive attempt will be to use a sorting algorithm, since after one small time step the list order on a manually defined comparison criterion, aka `operator<`

should not change significantly. For this purpose (nearly sorted arrays), insertion sort appears to approach a near $O(N)$ performance: See here for a comparison.

However we notice we can do this as we construct the tree, since we need to find which cell the particle belongs to anyway. We can then do the following:

```
for each BODY in BODIES
  recursively add it into a leaf box
  note which octant of the box the BODY belongs to
  if (box has already has BODIES >= upper limit)
      split the box according to the octant information
      form the child boxes
  else
      simply append index of BODY into list SOULS of the box
```

This might not be necessary for efficient splitting though, since each time we split there is only a constant $O(1)$ number of particles on that leaf node.

Our algorithm will create an *adaptive tree*.

3.1.3 Dual tree traversal

A detailed explanation can be found on pages 6, 7 and 15 of *Dehnen 2002*. For the mutual interaction variant it is common to pick the CoM as cell interaction centres (Sec. 5.1, Dehnen 2014).

3.1.4 Multipole expansion

This is explained in detail in pages 2 (spherical harmonics) and 17 (Cartesian FMM)in *Dehnen 2014*.

3.1.5 Integration solver

As a first step we shall implement *leapfrog* which requires only acceleration, which is readily given by *Dehnen 2014*. We may generalise to more complicated methods later on. Some explanations for the method can be found here.

4 Implementation

We will use object-oriented C++. A good reference on this is Pitt-Francis and Whiteley, *Guide to Scientific Computing in C++*.