The Barnes-Hut Algorithm: An N-Body Simulator

1st Gabriel Balarezo .B
School of Physical Sciences and Nanotechnology
Yachay Tech University
Urcuqui, Ecuador
juan.balarezo@yachaytech.edu.ec

***Abstract*—The N-body problem, which simulates the evolution of a system of particles interacting through a gravitational field, holds significant computational challenges due to its inherent complexity. In this project, we implement and compare two approaches to N-body simulations: a brute force method and the Barnes-Hut algorithm. The brute force method computes pairwise gravitational forces between all particles (bodies), which results in an $\mathcal{O}(N^2)$ time complexity–ensuring accuracy but poorly scaling with large systems. Conversely, the Barnes-Hut algorithm leverages a hierarchical tree structure– quad-tree for 2D and octree for 3D–to approximate distant forces, achieving a time complexity of $\mathcal{O}(N \log N)$ and improving performance for large-scale simulations. Both methods were implemented in `C++`, using Object Oriented Programming (OOP), `SDL` library for graphics rendering and `CMake` for project management. Preliminary results feature the trade-offs between computational cost and precision, with brute force excelling in small systems and Barnes-Hut outperforming it as N increases. This work details the algorithms, their implementations, and performance comparisons, offering insights into their practical applications and limitations. Future improvements, such as parallelisation, could further refine the performance of these algorithms.**

***Index Terms*—N-body simulation, Barnes-Hut algorithm, brute force method, computational physics, gravitational interactions.**

I. Introduction

In physics, the N-body problem refers to the task of predicting the individual motions of a group of N interacting particles (bodies)—such as planets, stars, or subatomic particles—under the influence of mutual forces, typically gravity. Given a set of initial conditions (positions, velocities) and the laws governing the interactions (e.g., Newton's law of gravitation), we are to determine the future state of the system. Applications of the N-body problem span various fields, including astrophysics [1] [2] (e.g., simulating galaxy formation), molecular dynamics [3] (e.g., simulating protein folding), and cosmology [4] (e.g., simulating large-scale structure formation).

For the purposes of this report, we shall focus on the gravitational N-body problem. Let a number, N, of particles interact classically through Newton's law of gravitation. Then, the equations of motion for the i-th particle are given by [5]:

$$\frac{d^2\mathbf{r}_i}{dt^2} = -\sum_{j=1,j\neq i}^{N} \frac{Gm_j(\mathbf{r}_i - \mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|^3} \tag{1}$$

where $\mathbf{r}_i$ is the position vector of the i-th particle, G is the universal constant of gravitation, and m_i is the mass of the i-th particle. For $N = 1$ and $N = 2$ the equations can be solved analytically. On the other hand, for $N \geq 3$, there is no general analytical solution. Thereby, numerical methods are required to approximate a solution.

In this context, some numerical methods for solving the N-body problem include [6]:

- **Direct N-body calculations**: These minimise the number of simplifying assumptions, but are rather computationally expensive, with a time complexity of $\mathcal{O}(N^2)$. Therefore, they are suitable for small systems.
- **Tree code methods**: Such as a Barnes-Hut simulation [7], are hierarchical methods which shorten the calculation of forces by grouping distant masses. This allows for a reduction in complexity to $\mathcal{O}(N \log N)$, making them suitable for larger systems.
- **Fast multipole methods**: These are more advanced techniques that further reduce the computational cost by exploiting series expansions of distant interactions. It is claimed that this approximation achieves a time complexity of $\mathcal{O}(N)$.

Our final goal is to implement a N-body simulator using both the brute force method and the Barnes-Hut algorithm using our knowledge of `C++` from our Advanced Programming course.

II. Brute Force Method

Let's consider two interacting bodies with masses m_1 and m_2, and positions $\mathbf{r}_1$ and $\mathbf{r}_2$ respectively , see Figure 1. The gravitational force acting on m_1 due to m_2 is in the direction of $\vec{r}_2 - \vec{r}_1 = \vec{r}_{12}$, and is given by:

$$\mathbf{F}_{12} = -\frac{Gm_1m_2}{|\mathbf{r}_{12}|^3}\mathbf{r}_{12} \tag{2}$$

Since the gravitational force is a symmetric interaction, the force acting on m_2 due to m_1 is given by:

$$\mathbf{F}_{21} = -\mathbf{F}_{12} \tag{3}$$

Therefore, the total force $\mathbf{F}_i$ on body i, due to its interactions with the other $N-1$ bodies is obtained by summing all the interactions:

$$m_i\frac{d^2\mathbf{r}_i}{dt^2} = \sum_{j=1,j\neq i}^{N} \mathbf{F}_{ij} = -\sum_{j=1,j\neq i}^{N} \frac{Gm_im_j}{|\mathbf{r}_{ij}|^3}\mathbf{r}_{ij} \tag{4}$$

and its equation of motion is given by:

$$\frac{d^2\mathbf{r}_i}{dt^2} = -\sum_{j=1,j\neq i}^{N} \frac{Gm_j(\mathbf{r}_i - \mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|^3} \tag{5}$$

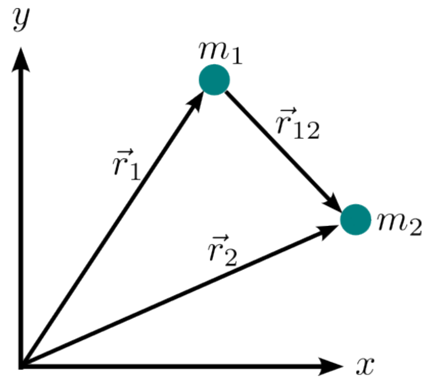

Fig. 1. Brute force method for two interacting bodies.

There is a singularity if $|\mathbf{r}_i - \mathbf{r}_j| = 0$, which occurs when two bodies are at the same position. To avoid this, the denominator is can be replaced by $(|\mathbf{r}_i - \mathbf{r}_j|^2 + \epsilon^2)^{3/2}$, where ϵ is a small constant, the softening parameter.

As mentioned earlier, the brute force method computes pairwise gravitational forces between all bodies, it is $N-1$ calculations per body, and $N(N-1)$ calculations in total. We can reduce this to half by leveraging the symmetry of the gravitational force, which ultimately results in a total of $N(N-1)/2$ calculations.

A. *Initial Conditions and Time Integration*

Equation (5) is a second-order ordinary differential equation (ODE). Since we are working in 2D, the overall system consists of $2N$ second-order ODEs, and hence, require $4N$ initial conditions. These initial conditions include the two Cartesian components of position $\mathbf{r}_i$ and the two components of the velocity $\dot{\mathbf{r}}_i$ of each one of the N particles.

On the other hand, to simulate the system over time, we need to integrate the equations of motion so we can predict the state of the system at time $t + \Delta t$, given its state at time t. A variety of numerical integrators can be used, such as Euler, Verlet, or Runge-Kutta. Since we are simulating a physical system with many interacting particles, it is crucial to use a stable integrator. For this reason, we choose Verlet integration, as it is numerically stable, time-reversible, and conservative, making it well-suited for simulations of this nature [8]. The Verlet integration is given by the following expressions:

$$\mathbf{r}_i(t+\Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t-\Delta t) + \mathbf{a}_i(t)\Delta t^2 + \mathcal{O}(\Delta t^4) \quad (6)$$

$$\dot{\mathbf{r}}_i(t+\Delta t) = \frac{\mathbf{r}_i(t+\Delta t) - \mathbf{r}_i(t)}{\Delta t} + \mathcal{O}(\Delta) \quad (7)$$

where $\mathbf{a}_i(t)$ is the acceleration of body i at time t, and Δt is the time step. The first equation updates the position of the body, while the second equation updates its velocity. The Verlet integration scheme is a symplectic integrator, which preserves the Hamiltonian structure of the equations of motion, making it suitable for long-term simulations of conservative systems.

Nevertheless, we must note that at the start of the Verlet integration, say at step $n = 1$, time $t = 1 = \Delta t$, in order to compute $\mathbf{r}_i(t=2)$ we need to know $\mathbf{r}_i(t=1)$ and $\mathbf{r}_i(t=1)$. This could represent a problem, because the only initials conditions we have are $\mathbf{r}_i(t=0)$ and $\dot{\mathbf{r}}_i(t=0)$. However, we know the value of $\ddot{\mathbf{r}}_i(t=0)$, and a suitable approximation for the position at the first time step can be obtained using the Taylor polynomial of degree two:

$$\mathbf{r}_i(t=1) = \mathbf{r}_i(t=0) + \dot{\mathbf{r}}_i(t=0)\Delta t + \frac{1}{2}\ddot{\mathbf{r}}_i(t=0)\Delta t^2 \quad (8)$$

This approximation is valid for small time steps, and it allows us to compute the initial position at the first time step.

B. *Pseudo Code*

Now that we have all the necessary equations, we can write the pseudo code for the brute force method. The algorithm consists of the following steps:

Algorithm 1 Brute Force Method

```
1: Initialise positions and velocities of $N$ bodies
2: Set time step $\Delta t$
3: while simulation not finished do
4:     for $i = 1$ to $N$ do
5:         Compute acceleration $\mathbf{a}_i(t)$ using Equation (5)
6:         Update position $\mathbf{r}_i(t+\Delta t)$ using Equation (6)
7:         Update velocity $\dot{\mathbf{r}}_i(t+\Delta t)$ using Equation (7)
8:     end for
9: end while
```

III. The Barnes-Hut Algorithm

As previously mentioned, the direct method scales poorly with large systems, and hence, we need to find a more efficient way to compute the forces acting on each body. The Barnes-Hut algorithm is an approximation algorithm for performing an N-body simulation with a time complexity of $\mathcal{O}(N \log N)$. The main idea is to approximate long-range forces by replacing a group of distant bodies with their centre of mass. In exchange for a small loss of precision, this scheme significantly speeds up calculations.

A. *Quad Tree Construction*

Figure 2 shows the quad tree construction. The steps can be summarised as follows:

- The simulation space is divided into four quadrants, each represented by a node in the tree. Each node contains a list of bodies that are located within its boundaries.
- If a node contains more than one body, it is subdivided into four child nodes, and the bodies are distributed among them.
- This process continues recursively until each node contains at most one body or reaches a predefined maximum depth.

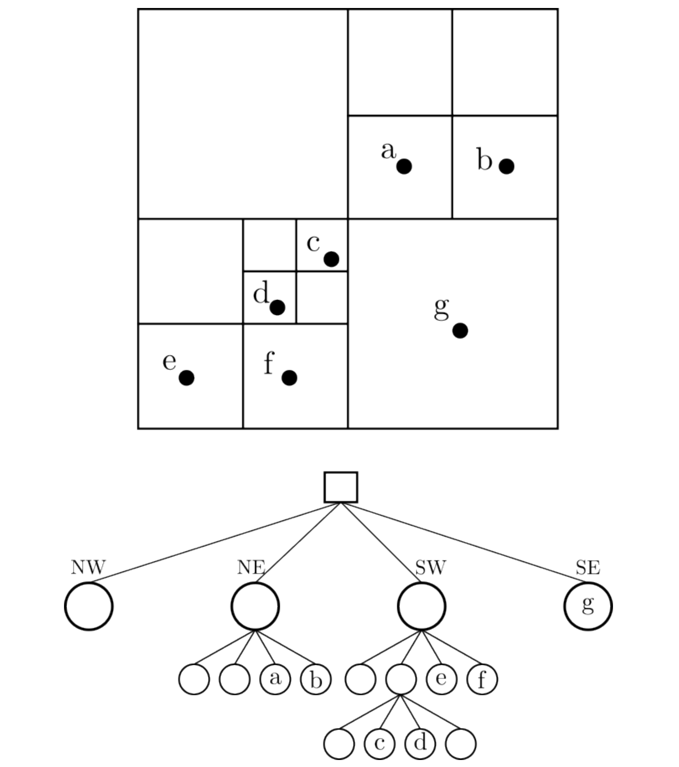

Fig. 2. Quadtree construction. Each node represents a quadrant and stores the centre of mass of its child nodes and the position of the centre of mass.

Additionally, we store the centre of mass of each node computed as follows:

$$\mathbf{r}_{cm} = \frac{\sum_{i=1}^{N} m_i \mathbf{r}_i}{\sum_{i=1}^{N} m_i} \tag{9}$$

where $\mathbf{r}_{cm}$ is the position of the centre of mass, m_i is the mass of body i, and $\mathbf{r}_i$ is its position. Moreover, we also need to compute the total mass of the node, which is given by:

$$M_{cm} = \sum_{i=1}^{N} m_i \tag{10}$$

where M_{cm} is the total mass of the node.

Finally, we need to define a condition such that nearby interactions are computed with the direct method, but approximates long-range interactions. This condition is called the θ criterion, and is defined as follows:

$$\frac{d}{D} < \theta \tag{11}$$

where d is the distance between the body and the centre of mass of the node, D is the size of the node, and θ is a parameter that determines the accuracy of the approximation. If the condition is satisfied, we can use the centre of mass to compute the force acting on the body. Otherwise, we need to compute the forces acting on each body in the node using the direct method.

IV. Preliminary results

Even though the logic of the direct method is quite straightforward, the graphics rendering need to be optimised so the overall performance is improved. In this section we included some images of the preliminary results of the brute force method. The simulation was run with 1000 bodies, and the time step was set to 0.0001 seconds. The simulation was run for 1000 steps, and the results are shown in Figure 3.

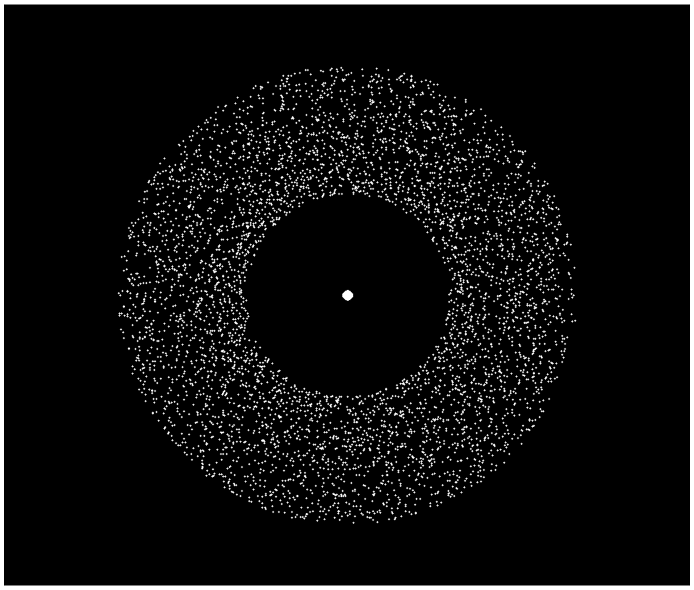

Fig. 3. Initial state of a system of 1000 particles.

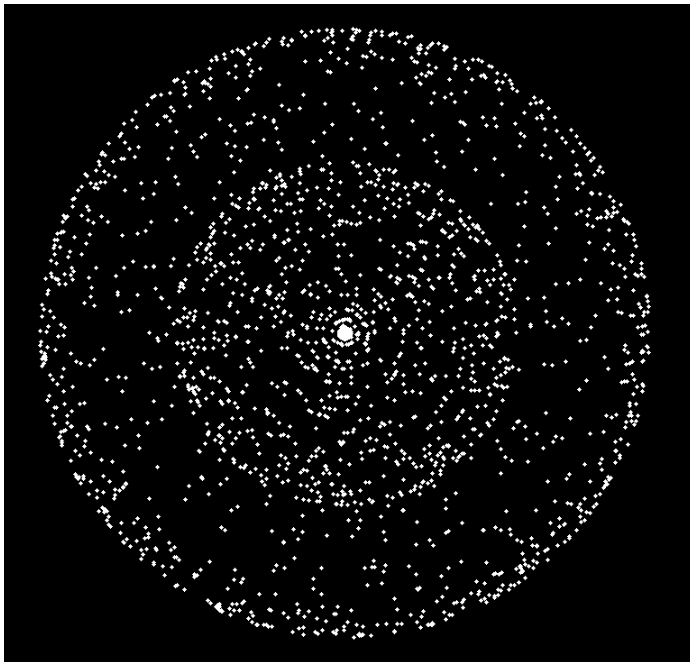

Fig. 4. Final state of a system of 1000 particles.

References

[1] L. Wang, M. Iwasawa, K. Nitadori, and J. Makino, “petar: a high-performance n-body code for modelling massive collisional stellar systems,” *Monthly Notices of the Royal Astronomical Society*, vol. 497, no. 1, pp. 536–555, 07 2020. [Online]. Available: https://doi.org/10.1093/mnras/staa1915

[2] “Chapter 31. Fast N-Body Simulation with CUDA,” Apr. 2025, [Online; accessed 6. Apr. 2025]. [Online]. Available: https://developer.nvidia.com/gpugems/gpugems3/part-v-physics-simulation/chapter-31-fast-n-body-simulation-cuda

[3] V. Maksimenko, A. Lipnitskii, A. Kartamyshev, D. Poletaev, and Y. R. Kolobov, “The n-body interatomic potential for molecular dynamics simulations of diffusion in tungsten,” *Computational Materials Science*, vol. 202, p. 110962, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0927025621006571

[4] A. Klypin, “Methods for cosmological n-body simulations,” *URL http://www. skiesanduniverses. org/resources/KlypinNbody. pdf*, 2017.

[5] D. Heggie, “Gravitational n-body problem (classical),” in *Encyclopedia of Mathematical Physics*, J.-P. Françoise, G. L. Naber, and T. S. Tsun, Eds. Oxford: Academic Press, 2006, pp. 575–582. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B0125126662000031

[6] D. C. Heggie, “The classical gravitational n-body problem,” *arXiv preprint astro-ph/0503600*, 2005.

[7] J. Barnes and P. Hut, “A hierarchical o (n log n) force-calculation algorithm,” *nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[8] W. T. V. B. P. F. William H. Press, Saul A. Teukolsky, *Numerical recipes: the art of scientific computing*, 3rd ed. Cambridge University Press, 2007. [Online]. Available: http://gen.lib.rus.ec/book/index.php?md5=0edc79280d53ef968ffa05d63dfd5e55