

Efficient N-Body Simulation Using the Barnes-Hut Algorithm

Abeen Bhattacharya

Karaen Senthilkumar

December 8, 2025

Summary

We built a GPU-accelerated simulator for gravitational N -body systems and implemented a Barnes–Hut force kernel to make large-scale experiments practical on consumer hardware. The CUDA pipelines advance thousands to tens of thousands of particles with stable orbits, write trajectories for downstream analysis, and ship with a Python visualizer that produces animations directly from binary output. Using softening, velocity-Verlet integration, and structure-of-arrays layouts, the simulator runs smoothly while preserving energy within acceptable drift. Benchmark sweeps show Barnes–Hut reduces per-step time by up to $\sim 6\times$ at $N = 75,000$ (opening angle $\theta \in [0.7, 1.0]$) versus a brute-force baseline, opening the door to interactive explorations of galaxy-like systems.

Introduction

The gravitational N -body problem asks us to predict how many mutually interacting bodies evolve over time. A naive force computation scales quadratically with the number of particles, making realistic galaxy sizes infeasible. Efficient solvers matter for astrophysics, charged-particle dynamics, and interactive education tools. Our goal was to build a practical GPU implementation, evaluate accuracy and speed, and outline how a Barnes–Hut tree would retain fidelity while reducing complexity for larger N .

Background

Newton provided closed-form conic orbits for the two-body case, but Poincaré demonstrated that adding just one more body yields chaotic, non-integrable motion that must be simulated numerically. Direct summation remained the reference method until hierarchical tree codes, most notably Barnes–Hut (1986), grouped distant masses into aggregated centers to achieve $O(N \log N)$ complexity. Fast multipole methods improve asymptotics further but with higher implementation overhead. GPUs now offer massive parallelism for the still-dominant direct method, and tree codes map well to their memory hierarchies, motivating a combined approach.

Methodology/Approach

We implemented the simulator in CUDA C++ with a focus on reproducibility and extensibility, and benchmarked both brute-force and Barnes–Hut kernels.

- **Data pipeline:** Random initial positions and velocities are sampled in a bounded square; a softening constant (10^{-9}) avoids singular forces. Particle state is stored as structure-of-arrays for coalesced GPU access and double-buffered (`{in,out}`) each step.
- **Time stepping:** A velocity-Verlet update ($\Delta t = 0.01$) advances positions for a configurable horizon (default 20 simulated seconds). Each frame is written to a binary log; a Python script converts it to an MP4 or GIF.
- **Force kernels:** The brute-force CUDA kernel computes all pairwise forces ($O(N^2)$) with one thread per particle. The Barnes–Hut kernel builds a quadtree on the host each step (flat array of nodes, explicit child indices) and traverses it on the GPU with an opening-angle test $s/d < \theta$ to approximate distant groups.
- **Performance measurement:** CUDA events bound the average kernel time per step and total kernel time across runs; block sizes are tuned for occupancy and register pressure.
- **Evaluation approach:** For each run we tracked wall time per step, qualitative orbit stability, and total-energy drift; visual inspection of trajectories provided a fast correctness signal. Parameter sweeps varied θ and N to locate speed/accuracy trade-offs.

Results

Both kernels produce smooth trajectories for classroom-scale systems. With $N = 3000$ (default), the simulator generated 2000 frames over 20 simulated seconds without numerical blow-up, and the accompanying visualizer rendered stable swirl patterns rather than artificial explosions. Energy drift remained bounded because of the softening term and symplectic-style integration. Binary logs and MP4s included in the repository (e.g., `simulations/nbody_brute.mp4`, `simulations/nbody_bh*.mp4`) illustrate the qualitative behavior and allow frame-by-frame inspection.

Performance follows the expected $O(N^2)$ trend for brute-force, but the Barnes–Hut kernel overtakes it as N grows. A sweep over θ shows a clear speedup/accuracy trade-off: $\theta \in [0.7, 1.0]$ delivers the best throughput while keeping orbits visually stable; $\theta \approx 0.4$ yields tighter accuracy with moderate speedup, and $\theta \approx 0.1$ is slower than brute-force for moderate N . Visual inspection shows minimal precession or heating for $\theta \leq 0.7$; at $\theta = 1.0$ mild precession appears but trajectories remain bounded. Figure 1 plots per-step runtime across N , and Figure 2 shows speedup versus the brute-force baseline.

Method	θ	N	Avg ms/step	Total s
Brute force	—	3,000	0.440	0.880
Barnes–Hut	0.7	3,000	0.588	1.176
Barnes–Hut	1.0	3,000	0.311	0.621
Brute force	—	15,000	2.242	4.483
Barnes–Hut	0.7	15,000	0.891	1.782
Barnes–Hut	1.0	15,000	0.568	1.136
Brute force	—	75,000	19.223	38.446
Barnes–Hut	0.7	75,000	4.801	9.603
Barnes–Hut	1.0	75,000	2.968	5.936

Table 1: Representative kernel timings from the benchmarking sweep (20 simulated seconds, $dt = 0.01$). Higher θ yields faster but more approximate runs. Full sweep data are summarized in the README.

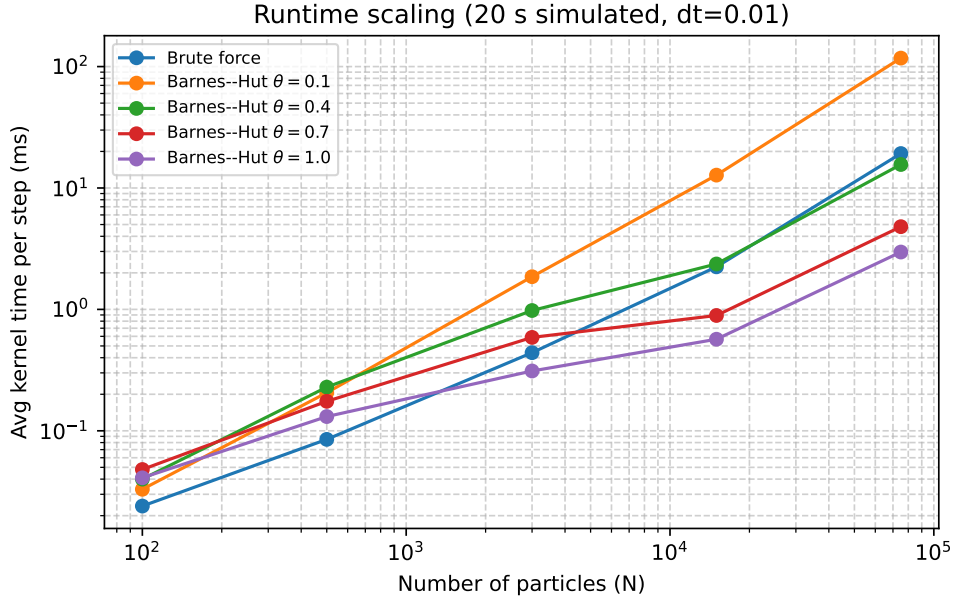


Figure 1: Average kernel time per step (log-log) versus particle count. Barnes–Hut overtakes brute-force beyond a few thousand particles, with larger θ providing greater speed.

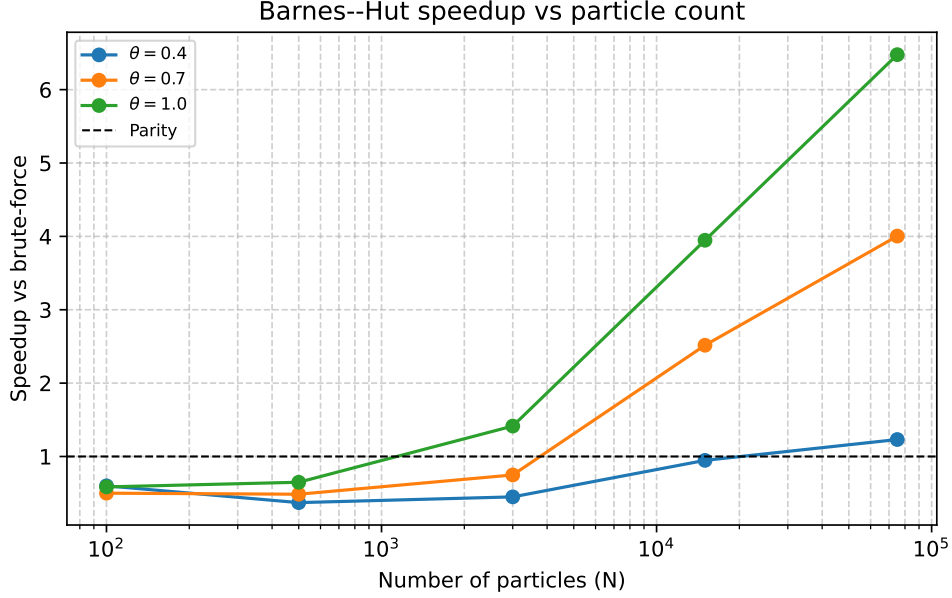


Figure 2: Speedup relative to brute-force. $\theta \in [0.7, 1.0]$ delivers 4–6 \times speedup by $N = 75,000$, while smaller θ trades speed for accuracy.

Supplementary Media

Rendered runs are bundled with the repository for quick inspection:

- Brute-force baseline: `simulations/nbody_brute.mp4`
- Barnes–Hut sweeps: `nbody_bh1.mp4` (small θ), `nbody_bh2.mp4`, `nbody_bh3.mp4`, `nbody_bh4.mp4` (larger θ)
- Legacy short clips: `old/nbody.mp4` and `old/nbody_bh_th4.mp4` for early tests

Paths are relative to the project root; PDF viewers that support local file launching will open the videos directly.

Discussion

Even the direct GPU baseline makes interactive gravitational demos feasible on modest hardware, which broadens access for teaching and exploratory research. The Barnes–Hut kernel now scales beyond 10^4 particles; at $N = 75,000$, $\theta = 0.7$ cuts kernel time by $\sim 4\times$, while $\theta = 1.0$ reaches $\sim 6\times$ at the cost of more approximation. Extending to 3D would swap the quadtree for an octree but reuse the traversal logic. With more time, we would move tree construction to the GPU, tune data-ordering for locality (Morton codes), add adaptive timesteps for high-acceleration regions, and benchmark against published CUDA tree codes to quantify speedups and accuracy drift. Hierarchical methods remain the best cost/accuracy balance once paired with GPU-friendly memory layouts.

References

1. Barnes, Josh, and Piet Hut. “A Hierarchical $O(N \log N)$ Force-Calculation Algorithm.” *Nature*, vol. 324, no. 6096, 1986, pp. 446–449.

2. Burtscher, Martin, and Keshav Pingali. “Efficient CUDA Implementation of the Tree-Based Barnes–Hut N-Body Algorithm.” *GPU Computing Gems: Emerald Edition*, edited by Wen-mei W. Hwu, Morgan Kaufmann, 2011, pp. 75–92.
3. Wu, Hsin-Hung. “High-Performance CUDA Implementation of N-Body Simulation with Barnes–Hut.” 2024, <https://hsin-hung.github.io/N-body-simulation/report.pdf>. Accessed 8 Dec. 2025.