IE University

School of Science & Technology

Hierarchical N-Body Simulation of Galactic Dynamics in WebGPU

Enabling Scalable and Interactive Physics Simulations on Modern Web Platforms

Zaid Alsaheb

Bachelor of Computer Science & Artificial Intelligence

Supervised by: Professor Raul Perez Pelaez

1) Literature Review

Accurate simulation of galactic dynamics requires resolving gravitational interactions across large numbers of particles while maintaining numerical stability over long integration times. Achieving this balance has historically required substantial computational resources and specialized software, placing high-fidelity N-body simulations largely outside the reach of lightweight or widely accessible platforms. As a result, the relevant literature spans several traditionally separate domains, including astrophysical N-body methods, numerical integration techniques, GPU-accelerated computing, and browser-based visualization technologies.

This literature review is organized to clarify how these areas intersect and to highlight the technical gap addressed by this work. It first examines the physical and algorithmic foundations of gravitational N-body simulation, with particular attention to the computational scaling challenges that motivate hierarchical approximation methods such as Barnes–Hut. It then surveys prior efforts to accelerate N-body simulations using GPU architectures, with an emphasis on new emergent technologies that allow to achieve native performance in web browsers, drastically enhancing the range of devices that can perform these simulations.

Computational Constraints in Astrophysical Simulation

Over the past decades, progress in computational astrophysics has closely followed the performance improvements predicted by Moore's Law, with central processing unit (CPU) speeds increasing at a near-exponential rate. Once an algorithm was implemented, substantial performance gains could often be achieved simply by running existing code on newer hardware, with minimal additional development effort. However, as single-core CPU performance has plateaued (Sutter, 2005), this implicit scaling model has begun to break down. As a result, continued advances in computational astrophysics increasingly depend on exploiting parallelism and adapting algorithms to emerging computing architectures (Fluke et al., 2011).

One of the most significant architectural shifts has been the widespread adoption of graphics processing units (GPUs), which offer massive parallelism and high memory bandwidth. While GPUs provide the potential for orders-of-magnitude performance improvements (Owens et al., 2007), realizing these gains typically requires substantial algorithmic reformulation. Algorithms that scale well on serial or modestly parallel CPUs may perform poorly on highly parallel architectures if they involve irregular control flow or memory access pattern (Burtscher & Pingali, 2011).

Gravitational N-Body Simulations and Quadratic Scaling

Among the most computationally demanding problems in astrophysics are gravitational N-body simulations, which form the foundation of many studies in galactic dynamics and structure formation (Binney & Tremaine, 2008). In such simulations, a galaxy is modeled as a system of particles (representing stars or dark matter) whose evolution is governed by their mutual gravitational interactions (Zwart et al., 2007).

According to Newton's law of universal gravitation, the force exerted on a particle of mass m_i by another particle of mass m_j is given by

$$\vec{F}_{ij} = G\frac{m_i m_j(\vec{r}_j - \vec{r}_i)}{|\vec{r}_j - \vec{r}_i|^3} \tag{1}$$

Where $\vec{F}_{ij}$ denotes the gravitational force exerted on particle i by particle j, and G is the gravitational constant. The quantities m_i and m_j represent the masses of particles i and j, respectively. The vectors $\vec{r}_i$ and $\vec{r}_j$ give the positions of particles i and j in three-dimensional space.

The vector difference $\vec{r}_j - \vec{r}_i$ points from particle i toward particle j, while $|\vec{r}_j - \vec{r}_i|$ denotes the Euclidean distance between the two particles. The cubic power of the distance in the denominator ensures that the magnitude of the force follows an inverse-square law while preserving the correct force direction.

The total gravitational force acting on particle i is obtained by summing the pairwise force contributions from all other particles in the system, excluding self-interaction,

$$\vec{F}_i = \sum_{j \neq i}^{N} G\frac{m_i m_j(\vec{r}_j - \vec{r}_i)}{|\vec{r}_j - \vec{r}_i|^3} \tag{2}$$

Evaluating this expression requires computing $N - 1$ pairwise interactions per particle. Consequently, a direct implementation of gravitational force evaluation scales as $O(N^2)$ per simulation timestep. This quadratic scaling rapidly becomes computationally prohibitive as the number of particles increases, particularly when long integration times are required to observe large-scale dynamical phenomena such as spiral structure or halo evolution (Zwart et al., 2007).

Hierarchical Approximation and the Barnes–Hut Algorithm

To address the computational limitations imposed by direct force evaluation, a variety of approximation techniques have been developed to reduce the cost of N-body simulations while preserving essential physical behavior. One of the most influential of these methods is the Barnes–Hut algorithm, which exploits the hierarchical spatial structure of particle distributions to approximate gravitational interactions (Barnes & Hut, 1986).

Originally introduced by Barnes and Hut in 1986, the algorithm organizes particles into a tree structure, typically a quadtree in two dimensions or an octree in three dimensions. During force evaluation, distant groups of particles are approximated as a single effective mass located at their center of mass, allowing subsets of the force summation to be replaced by a single interaction. An opening-angle parameter controls the accuracy of this approximation, providing a tunable tradeoff between computational efficiency and force accuracy.

By replacing many distant pairwise interactions with aggregate approximations, Barnes–Hut reduces the computational complexity of force evaluation from $O(N^2)$ to approximately $O(N \log N)$. This reduction makes the algorithm particularly well-suited for simulating large, collisionless systems such as galaxies, where global dynamical behavior is often of greater interest than exact short-range interactions.

Numerical Integration and Stability Considerations

In addition to force evaluation, the accuracy and long-term stability of N-body simulations depend critically on the numerical integration of the equations of motion. In gravitational N-body problems, the primary quantity of interest is the time evolution of the particle positions, which is governed by Newton's equations of motion. For each particle i, the total gravitational force resulting from interactions with all other particles determines its acceleration according to

$$\sum_j \vec{F}_{ij} = m_i \vec{a}_i \tag{3}$$

Where $\vec{a}_i$ is the acceleration of the particle i. The acceleration is related to the particle velocity and position through time derivatives,

$$\vec{a}_i = \frac{d\vec{v}_i}{dt}, \qquad \vec{v}_i = \frac{d\vec{r}_i}{dt} \tag{4}$$

Because these equations generally cannot be solved analytically for systems containing many interacting particles, their evolution must be approximated numerically. Numerical integration methods advance the particle positions and velocities forward in time using discrete timesteps of size Δt, approximating the continuous dynamics of the system.

One of the simplest numerical integration schemes is the forward Euler method (Kreyszig, 2011). In this approach, particle positions are updated using the current velocity according to

$$\vec{r}_i^{\,n+1} = \vec{r}_i^{\,n} + \vec{v}_i^{\,n} \Delta t \tag{5}$$

Where the superscript n denotes the discrete timestep, with $\vec{r}_i^{\,0}$ representing the initial particle position. Velocities are updated similarly using the acceleration computed from the forces at the current timestep. While the Euler method is computationally inexpensive and straightforward to implement, it suffers from poor numerical stability and does not conserve energy, leading to significant accumulated errors and unphysical behavior over long integration times.

As a result, astrophysical N-body simulations commonly employ symplectic integration schemes such as the Verlet or leapfrog methods, which are specifically designed to preserve the Hamiltonian structure of the equations of motion. These methods exhibit significantly improved long-term energy conservation and are therefore better suited for simulations of galactic dynamics that require stable evolution over many dynamical timescales (Springel, 2005).

In Barnes–Hut simulations, errors introduced by numerical integration interact with approximation errors arising from hierarchical force evaluation (Salmon & Warren, 1994). Prior studies have shown that, for many galactic-scale simulations, integration error can dominate force-approximation error when inappropriate timestepping schemes or integration methods are used (Springel, 2005). Consequently, careful selection of numerical integration techniques remains essential even when approximate force models are employed.

GPU Acceleration of Hierarchical N-Body Methods

The computational demands of gravitational N-body simulations have made them a natural target for graphics processing units (GPUs), which provide massive data parallelism and high memory bandwidth. Modern GPUs are designed with thousands of small, efficient cores capable of executing the same operation on many data elements simultaneously (Fluke et al., 2011). This architecture is particularly well-suited to the independent, particle-wise force computations inherent in N-body simulations (Nyland et al., 2009).

To leverage GPUs effectively, computational tasks are expressed in terms of small programs that run on the GPU cores called shaders. Originally, shaders were designed to compute visual effects for rendering pipelines, including vertex transformations, fragment coloring, and texture operations. Early GPU-based scientific computing therefore relied on repurposing graphics shaders for numerical tasks, encoding computation as rendering operations and storing data in textures (Owens et al., 2008).

Over time, GPU programming languages such as CUDA and OpenCL have generalized this model to allow general-purpose computation, enabling the parallel execution of tasks that are not directly related to graphics, including N-body simulations (Fluke et al., 2011; Nyland et al., 2009).

Despite these advances, hierarchical algorithms such as Barnes–Hut present persistent challenges for GPU architectures. Tree construction and traversal introduce irregular memory access patterns and branch divergence, both of which reduce parallel efficiency on SIMD-style hardware (Burtscher & Pingali, 2011; Karras, 2012). To mitigate these issues, prior work has proposed optimizations including linearized tree representations, stackless traversal methods, and the use of space-filling curves to improve memory coherence (Burtscher & Pingali, 2011;

Gaburov et al., 2010). While these techniques have enabled efficient native GPU implementations, they rely on low-level memory management and flexible data structures that have traditionally been inaccessible in browser-based computing environments.

Since the introduction of the Barnes–Hut algorithm, numerous modifications and alternative hierarchical methods have been proposed to improve accuracy and efficiency, including higher-order multipole expansions and the Fast Multipole Method (FMM) (LF & Rokhlin, 2001; Nyland et al., 2009; Wang, 2021). These approaches differ primarily in how distant particle interactions are approximated and aggregated. Despite these developments, Barnes–Hut remains widely used due to its favorable balance between computational cost, accuracy, and implementation complexity.

General Purpose GPU Computation in the Web

Modern web applications increasingly rely on GPU acceleration to deliver interactive, intelligent, and computation-heavy experiences directly in the browser. Access to GPU resources enables use cases such as real-time large-model reasoning, embedding generation, multimedia processing, and advanced data visualization without requiring users to install specialized software or manage local hardware (Congote et al., 2011; Sung et al., 2025; Usta, 2024). For users working across devices or within managed environments e.g., students, researchers, enterprise teams), browser-based GPU access lowers friction, improves accessibility, and supports scalable, on-demand compute.

WebGL has long served as the primary mechanism for accessing GPU acceleration within web browsers. Standardized by the Khronos Group as a JavaScript binding to OpenGL ES, WebGL was designed to provide portable and efficient access to GPU hardware for real-time graphics rendering across a wide range of devices (Khronos Group, 2024). Its success has enabled a broad ecosystem of browser-based visualization tools, interactive simulations, and educational applications.

Although WebGL exposes programmable shaders, its computational model is fundamentally centered around the graphics rendering pipeline. All computation must be framed in terms of vertex and fragment processing stages, with data represented as textures and intermediate results captured via framebuffers (Congote et al., 2011). As a consequence, general-purpose computation in WebGL requires reformulating numerical algorithms as sequences of rendering passes, often involving non-intuitive data layouts and multiple shader invocations to emulate iteration and state updates.

This graphics-centric abstraction has enabled a class of data-parallel algorithms to be executed efficiently in the browser, particularly those that map naturally onto image-based representations. However, it imposes significant constraints on algorithms that require irregular memory access, dynamic data structures, or complex control flow (Congote et al., 2011) . In the context of gravitational N-body simulations, these constraints complicate the implementation of adaptive time stepping, hierarchical spatial decomposition, and recursive tree traversal, all of which are central to physically accurate Barnes–Hut simulations (Burtscher & Pingali, 2011).

As a result, prior WebGL-based N-body implementations have typically prioritized real-time performance and visual plausibility over numerical rigor. Common approaches include limiting particle counts, employing softened or approximate force models, reducing integration accuracy, or relying on fixed spatial grids rather than fully adaptive hierarchical trees (Sengupta et al., 2025). While such methods are effective for interactive visualization and outreach, they are not well suited to large-scale galactic simulations where long-term energy conservation and accurate force evaluation are essential.

WebGPU is a modern web standard designed to provide low-level, high-performance access to GPU hardware from within the browser. Unlike WebGL, which is built around a traditional graphics pipeline, WebGPU exposes explicit support for general-purpose GPU computation through compute shaders, storage buffers, and programmable pipelines (GPU for the Web Working Group, 2026). Its design closely mirrors contemporary native graphics and compute APIs such as Vulkan, Metal, and Direct3D 12, enabling fine-grained control over memory layout, synchronization, and parallel execution (Usta, 2024).

Central to WebGPU’s computational model is the compute shader: a general-purpose GPU program that executes independently of the rendering pipeline. Compute shaders allow developers to express parallel workloads directly in terms of data processing rather than image generation, making them well suited to scientific computing tasks such as particle simulation, spatial partitioning, and force evaluation (Sung et al., 2025; Usta, 2024). This model enables algorithms to be structured in a manner similar to native GPU implementations, without the need for graphics-specific workarounds (Sengupta et al., 2025).

The explicit memory model and flexible buffer abstractions provided by WebGPU make it possible to represent complex data structures, including hierarchical trees and linearized spatial indices, directly on the GPU (Sung et al., 2025; Usta, 2024). These features are particularly relevant for Barnes–Hut simulations, which rely on dynamic tree construction and traversal to achieve sub-quadratic scaling. While such patterns were previously impractical in web environments, WebGPU enables their implementation with performance characteristics comparable to native GPU code, subject to browser and hardware constraints (Sengupta et al., 2025).

2) Methodology

2.1 Research Design and Objectives

This work adopts a computational-methods design in which a gravitational N-body solver is implemented and evaluated with a focus on reproducibility, long-term numerical stability, and computational scalibility beyond the $O(N^2)$ cost of direct summation. The solver is implemented in C++20 using the WebGPU C API and is built from a single codebase targetting both:

- Native desktop execution, using WebGPU backends such as wgpu-native and Dawn.
- Browser execution, compiled via Emscripten.

This dual-target approach enables interactive visualizaiton as well as headless batch runs, allowing performance and numerical behavior to be evaluated under comparable conditions across platforms.

The methodological choices follwo directly from the literature foundations on hierarchical N-body simulation and GPU parallelism:

- **Force Model**: Newtonian gravity with Plummer-type softening to avoid the $\frac{1}{r^2}$ singularity and reduce spurious two-body relaxation in collisionless regimes.
- **Acceleration strategy**: a Barnes-Hut style hierarchical approximation that reduces force-evaluation from $O(N^2)$ to approximately $O(N \log N)$ (Barnes & Hut, 1986). In the primary path, both tree construction and force evaluation are executed on the GPU: a Linear Bounding Volume Hierarchy (LBVH) is constructed fully on-device using the parallel method of (Karras, 2012), removing the per-step CPU to GPU tree upload bottleneck.
- **Time Integration**: a second order symplectic leapfrog integrator (kick–drift–kick; also known as velocity Verlet in an equivalent form) selected for improved long-term energy behavior relative to forward Euler in gravitational systems (Springel, 2005).
- **Compute platform**: WebGPU compute shaders for general-purpose parallel kernels, enabling both native and browser deployment through the same low-level API surface.

The evaluation is structured around three operational research questions:

1. **Scalability**: How does runtime per timestep scale with N for a WebGPU Barnes-Hut implementation compared to a direct $O(N^2)$ baseline at small N?
2. **Numerical quality**: For fixed N, how do timestep size Δt and opening angle θ affect (a) long-term conservation behavior (energy and momentum drift where measurable) and (b) stability under long integrations?
3. **Platform feasibility**: what particle counts and timestep rates are practical under WebGPU constraints (32-bit GPU arithmetic, buffer/memory limits, scheduling overhead, and device variability)?

2.2 Initial conditions and benchmark scenarios

No external astronomical datasets are used. All experiments are generated from synthetic initial conditions and produce derived outputs (trajectories, diagnostic scalars, and timing logs). This design eliminates licensing and privacy concerns and enables controlled, repeatable comparisons across parameter sweeps.

Each experiment is fully specified by command-line parameters: scenario type, seed, N, Δt, θ, softening ε, and step count. Initial condition generation uses `std::mt19937` seeded by `--seed` (default seed = 42), ensuring deterministic reproduction of particle distributions and velocities.

2.2.1 Scenario A – Two-body orbit (sanity check)

- **Scope**: $N = 2$ (enforced regardless of the `--N` parameter)
- **Purpose**: verifies integrator correctness and sensitivity to Δt and ε.
- **Setup**: two equal-mass particles ($m = 1000$ each) separated by $d = 10$ units along the x-axis, with tangential velocities along the z-axis computed for a softened circular orbit:

$$v = \sqrt{\frac{Gmd^2}{2(d^2 + \varepsilon^2)^{\frac{3}{2}}}} \tag{6}$$

- **Key variables**: Δt, ε
- **Limitations**: not representative of large-N hierarchical behavior

2.2.2 Scenario B – Plummer sphere (spherical equilibrium test)

- **Scope**: $N \in [10^3, 10^5]$ (depending on hardware).
- **Purpose**: tests tree accuracy and stability in a compact 3D distribution with known analytic properties.
- **Setup**: a Plummer model (Aarseth et al., 1974) with scale length $a = 5$. Radii sampled via inverse CDF:

$$r = \frac{a}{\sqrt{u^{-\frac{2}{3}} - 1}} \tag{7}$$

with u clamped to $[0.001, 0.999]$. Angular coordinates are isotropic (uniform cos(θ), uniform ϕ). Speeds are sampled via rejection sampling using

$$g(q) = q^2(1 - q^2)^{\frac{7}{2}} \tag{8}$$

against the local escape velocity

$$v_e = \sqrt{\frac{2GM}{\sqrt{r^2 + a^2}}} \tag{9}$$

with isotropic velocity directions.

- **Key variables**: θ, ε, Δt
- **Limitations**: does not emphasize disk morphology (spirals/bars)

2.2.3 Scenario C – Rotating exponential disk (galaxy-like morphology test)

- **Scope**: $N \in [10^4, 10^5]$
- **Purpose**: evaluates long-term evolution and visually interpretable galactic dynamics
- **Setup**: radii drawn from an exponential distribution (rate 0.08) and clamped to 50, uniform azimuth. Vertical height is drawn from $N(0, 0.3)$ scaled by $\frac{1}{1+0.5r}$. Masses are uniform in $[0.5, 2.0]$. Circular velocities are assigned with an approximate enclosed-mass estimate, using (for $r > 0.1$)

$$v = 0.5\sqrt{\frac{M_{\text{enclosed}}}{r}} \tag{10}$$

with tangential direction.

- **Key variables**: disk scale length, thickness, velocity dispersion, θ, Δt
- **Limitations**: simplified dynamical setup (not a full multi-component Milky Way model); enclosed-mass estimate is approximate.

2.2.4 Sampling and robustness across seeds

Because these scenarios are stochastic, robustness is assessed by repeating runs with different seeds (`--seed 1`, `--seed 2`, ...) and comparing diagnostics and timing. Runs are considered valid if they complete without NaNs/overflow and produce consistent parameter logs. Deliberately unstable settings (e.g., excessively large Δt) are retained as documented failures for robustness reporting rather than silently excluded.

2.3 Physical model and state representation

2.3.1 Softened gravitational acceleration

Each particle represents a mass element evolving under self-gravity in an isolated (open) domain. Using dimensionless units with $G = 1$, the softened acceleration of particle i is

$$a_i = G \sum_{j \neq i} m_j \frac{r_j - r_i}{\left(\|r_j - r_i\|^2 + \varepsilon^2\right)^{\frac{3}{2}}} \tag{11}$$

Softening parameter ε defaults to 0.5 and is configurable.

2.3.2 GPU-friendly mass packing

To reduce memory bandwidth, each particle mass is stored in the w component of its position vector (`vec4: x,y,z,m`), avoiding a dedicated mass buffer.

2.3.3 Precision strategy

GPU kernels operate in 32-bit floating point to maximize throughput and match typical WebGPU availability. Diagnostic quantities (energy and momentum) are computed on the CPU in double precision to reduce accumulation error. This split reflects the practical precision/ performance trade-off in WebGPU environments.

2.4 Time Integration

2.4.1 Primary integrator: symplectic leapfrog (KDK)

The primary integration scheme is a fixed-timestep, second-order symplectic leapfrog (kick-drift-kick). With timestep Δt (default (0.0001) the update is:

1. Half-kick

$$v_i^{n+\frac{1}{2}} = v_i^n + \frac{\Delta t}{2} a_i^n \tag{12}$$

2. Drift

$$r_i^{n+1} = r_i^n + \Delta t v_i^{n+\frac{1}{2}} \tag{13}$$

3. Recompute acceleration a_i^{n+1} using updated positions
4. Half kick

$$v_i^{n+1} = v_i^{n+\frac{1}{2}} + \frac{\Delta t}{2} a_i^{n+1} \tag{14}$$

This choice is motivated by the well-known long-term stability advantages of symplectic schemes in gravitational dynamics (Springel, 2005), particularly when combined with approximate force evaluation.

2.4.2 2.4.2 Euler integrator (baseline/fallback)

A forward Euler method is retained as `--integrator euler` to provide a stability baseline. Its update sequence is: tree build $\rightarrow$ force evaluation $\rightarrow$

$$v \leftarrow v + a\Delta t, \;\; r \leftarrow r + v\Delta t \tag{15}$$

2.5 Hierarchical force evaluation

2.5.1 Monopole approximation

Hierarchical evaluation approximates distant particle groups by a single monopole at the node center of mass. For a node with total mass M and center of mass R,

$$a_{i,\text{node}} = GM \frac{R - r_i}{(\|R - r_i\|^2 + \varepsilon^2)^{\frac{3}{2}}} \tag{16}$$

, This monopole approximation is used consistently across both tree topologies implemented here: a binary BVH (GPU primary) and an 8-way octree (CPU fallback) Only the tree representation and opening criterion differ.

2.5.2 Opening Criteria

An internal node is accepted if it is sufficiently small relative to its distance from the target particle.

- GPU BVH (tight AABB with maximum extent $\mathrm{maxExtent}$):

  $$\frac{\mathrm{maxExtent}^2}{d^2} < \theta^2 \tag{17}$$

  where

  $$\mathrm{maxExtent} = \max(\Delta x, \Delta y, \Delta z) \tag{18}$$

  derived from node AABB bounds. This criterion reflects non-cubic node shapes in a BVH more accurately than a uniform half-width.

- CPU octree (half-width h, distance squared $d^2 = \|r_i - R\|^2$):

  $$\frac{h^2}{d^2} < \theta^2 \tag{19}$$

  This squared form avoids a square root.

Default $\theta = 0.75$ is used as a practical balance between accuracy and performance.

2.6 Software architecture and execution modes

2.6.1 GPU-primary stepping with on-demand diagnostics

The primary execution mode is GPU-primary: all physics operations (integration, tree construction, and force evaluation) are performed on the GPU each step. Diagnostic quantities are computed on the CPU only at configurable intervals through staging-buffer readback of positions and velocities. Diagnostic frequency is set to every 60 frames in interactive mode, and every step or every 50 steps in headless mode depending on N.

To support cross-checking and fallback behavior, CPU mirror arrays (`cpuPositions_`, `cpuVelocities_`, `cpuAccelerations_`) are retained and used by non-primary modes (Euler and CPU-tree leapfrog), where scalar CPU loops and a CPU octree are executed in parallel with the GPU path.

This design yields two methodological advantages:

1. No per-step CPU physics overhead in the primary mode: the tree is built directly from GPU-resident particle state.

2. Cross-validation potential across runs: selecting CPU octree vs GPU BVH paths provides two independent hierarchical implementations for comparative diagnostics, aiding debugging and sensitivity analysis.

2.6.2 Tools and build system

- Language: C++20 (orchestration, physics, UI); WGSL (compute and rendering).
- API: WebGPU C API only (no wrapper libraries).
- Build system: CMake + FetchContent with pinned versions (deterministic builds).
- Dependencies (fetched automatically): WebGPU-distribution (v0.2.0), GLFW (3.4), glfw3webgpu (v1.2.0), spdlog (v1.16.0), Dear ImGui (v1.90.9), GLM (1.0.2).
- Backends: `WGPU` (wgpu-native), `DAWN` (Dawn), `EMSCRIPTEN` (browser build with `-sASYNCIFY`, `-sALLOW_MEMORY_GROWTH=1`, `-sUSE_GLFW=3`).
- Deterministic PRNG: `std::mt19937` seeded by `--seed` (default 42).

2.7 WebGPU Compute Methodology

2.7.1 GPU Data Layout

Simulation state is stored in WebGPU storage buffers:

- `positions: array<vec4f>` - (x, y, z, m)
- `velocities: array<vec4f>` - $(v_x, v_y, v_z, 0)$
- `accelerations: array<vec4f>` - $(a_x, a_y, a_z, 0)$
- `bvhNodes: array<BVHNode>` - GPU-built LBVH nodes (force traversal)
- `octreeBuffer: array<Node>` - flattened CPU octree nodes (fallback only)
- `paramsBuffer: Params` - uniform parameters shared between C++ and WGSL (layout-matched)

The BVH uses a binary representation with $2N - 1$ nodes: internal nodes indexed $0...N - 2$, leaves indexed $N - 1...2N - 2$. Each BVH node stores center of mass and a tight AABB. The CPU octree node layout uses explicit child fields `c0..c7` to avoid dynamic indexing limitations in WGSL implementations.

In-place updates are used: there is no double buffering of positions/velocities/accelerations. Correct ordering between compute passes relies on WebGPU's implicit storage-buffer synchronization between passes within a single command buffer submission. Bind groups are recreated each frame.

2.7.2 Compute Shaders

The implementation comprises 12 WGSL compute shaders:

- Integration and force: direct summation (baseline), octree traversal (fallback), BVH traversal (primary), plus kick/drift and Euler integration shaders.

- LBVH construction (7 passes): two-pass bounding box reduction, Morton code generation, bitonic sort (multiple sub-passes), Karras (2012) topology construction, leaf initialization, and bottom-up aggregation via atomic counters.

Workgroup sizes are fixed per kernel (e.g., 64 for force evaluation, 256 for integration and tree building) and are reported as part of the implementation configuration.

2.7.3 Per-timestep execution sequence (GPU-primary leapfrog)

Each timestep is recorded into a single command encoder and submitted as one command buffer:

1. Half-kick: $v \leftarrow v + \frac{a\Delta t}{2}$
2. Drift: $r \leftarrow r + (v\Delta r)$
3. LBVH build (7 passes): global AABB $\rightarrow$ Morton codes $\rightarrow$ bitonic sort $\rightarrow$ Karras build $\rightarrow$ leaf init $\rightarrow$ bottom-up aggregation
4. BVH force evaluation: iterative traversal with fixed-depth explicit stack (depth 64; sufficient for all tested N)
5. Half-kick: $v \leftarrow v + \frac{a\Delta t}{2}$
6. Diagnostics readback (periodic): stage-map-readback $\rightarrow$ CPU double-precision diagnostics

2.7.4 GPU traversal (iterative, no recursion)

Tree traversal is implemented iteratively in the BVH force shader. One GPU thread is assigned per particle. The thread maintains an explicit stack of node indices, beginning from the root. A node is either accepted (leaf or opening-criterion satisfied) and accumulated via the monopole approximation, or expanded by pushing its children. Fast inverse square root (`inverseSqrt`) is used for inverse-distance evaluation. A self-interaction guard avoids adding contributions for degenerate near-zero distances.

This approach preserves the Barnes–Hut approximation structure while accommodating GPU execution constraints and limiting branch divergence where possible (Burtscher & Pingali, 2011; Karras, 2012; Nyland et al., 2009) .

2.7.5 GPU LBVH construction (Karras 2012)

The LBVH is built fully on-device in seven conceptual steps:

1. Two-pass parallel reduction to compute global AABB.
2. Morton code generation by normalizing positions to a $[0, 1023]^3$ integer grid and interleaving bits (30-bit code).
3. Bitonic sort of Morton codes and particle indices (key–value), using multiple (k, j) sub-passes with dynamic uniform offsets; padded-to-power-of-two arrays are used, with sentinel codes for padding elements.

4. Parallel binary tree topology construction using the Karras (2012) delta function (leading zeros of XOR of adjacent codes, with tie-breaking for duplicates).
5. Leaf initialization mapping sorted indices to particle positions/masses and point AABBs.
6. Bottom-up aggregation of internal-node AABBs and centers of mass using atomic counters to ensure both children are ready before parent evaluation.

The resulting BVH is immediately traversable without CPU-side construction or upload, eliminating a per-step CPU bottleneck in the primary mode.

2.7.6 CPU octree construction (fallback paths)

The CPU octree is used only by Euler and CPU-tree leapfrog modes. It is built from CPU mirror arrays by computing a bounding box, inserting particles via octant selection, propagating centers of mass bottom-up, and optionally flattening to a GPU-friendly node array when GPU evaluation is used. GPU buffers auto-resize during uploads when needed.

2.8 Rendering and interactive operation (visualization mode)

For visualization, particles are rendered as instanced billboard quads with additive blending. Positions and colors are read directly from storage buffers via `@builtin(instance_index)` (no vertex buffer). Depth testing is enabled with depth writes disabled; fragments are masked to a circular footprint with a soft alpha falloff. An ImGui overlay provides interactive control of parameters and displays diagnostics and timing breakdowns. Rendering is a presentation layer and does not alter the simulation state.

2.9 Evaluation protocol: baselines, metrics, and parameter sweeps

2.9.1 Baselines

Two baselines are used:

1. Direct summation: $O(N^2)$ for small N, used as a reference computation path. Additionally, potential energy is computed by direct pair summation only when $N \leq 5000$ due to its $O(N^2)$ cost.
2. Forward Euler: (`--integrator euler`) as a numerical stability baseline relative to leapfrog.

2.9.2 Primary metrics

- **Runtime per timestep** (ms/step), broken down into:
  - tree build time (GPU LBVH or CPU octree + upload),
  - force evaluation time (GPU BVH traversal or CPU Barnes–Hut),
  - integration time (kick/drift dispatches and any CPU mirror loops on fallback paths). Timing is measured using `std::chrono::high_resolution_clock`.

- **Scaling with particle count**: empirical scaling trends across (N) for hierarchical vs direct modes.
- **Long-term stability** (where measurable): energy drift

$$\Delta E(t) = \frac{|E(t) - E(0)|}{E(0)} \tag{20}$$

  reported only for

$$N \leq 5000 \tag{21}$$

  where potential energy is computed.

2.9.3 Secondary Metrics

- **Linear momentum magnitude**

$$\|P(t)\| = \| \sum_i m_i v_i \| \tag{22}$$

  double precision), expected to remain near zero for symmetric initial conditions.
- **Qualitative morphology (disk runs)**: persistence and evolution of large-scale structure, reported descriptively rather than as a ground-truth numerical metric.

2.9.4 Parameter sweeps (ablation-style sensitivity analysis)

To characterize accuracy–performance trade-offs and stability regimes, controlled sweeps are performed via CLI:

- $\theta \in \{0.3, 0.5, 0.7, 1.0\}$
- Δt across a stable range (scenario dependent)
- ε across representative values
- Euler vs leapfrog
- CPU octree vs GPU LBVH construction paths

Results are compared using exported CSV logs.

2.9.5 Error analysis procedure

When instability or anomalous drift occurs, runs are inspected for correlations with: dense regions vs diffuse regions, large θ, large Δt, small ε, and platform/backend differences. Failures (NaNs, overflow, extreme velocities) are detected explicitly and logged.

2.10 Validation and Robustness

2.10.1 Theoretical Grounding

- Barnes–Hut hierarchical approximation and tunable opening angle θ follow Barnes & Hut (1986).

- Leapfrog integration is a standard choice for long-horizon gravitational dynamics due to symplectic stability properties (Springel, 2005)
- The GPU hierarchy construction follows Karras (2012), a widely used parallel LBVH method.
- GPU traversal methodology follows established considerations for irregular tree algorithms on SIMD-style hardware (Burtscher & Pingali, 2011; Karras, 2012)

2.10.2 Empirical validation

- Scenario A (two-body) provides a controlled sanity check for orbit stability and integrator correctness.
- Deterministic seeds and complete parameter logging allow exact repetition.
- Fixed step counts (`--steps`) provide consistent comparison across runs; instability events are recorded rather than filtered.

2.10.3 Practical constraints

Interactive runs couple stepping to the render loop; headless mode prioritizes throughput. Particle count is adjustable (2 to 100,000). Potential energy tracking is intentionally limited to $N \leq 5000$ to avoid prohibitive overhead; for larger N, stability is assessed through kinetic energy and momentum diagnostics plus qualitative behavior.

2.11 Data ethics, security, and integrity

No personal or sensitive data are collected. All computation runs locally (native) or within the user's browser sandbox (Emscripten). Diagnostic logs and trajectories are exported only when explicitly requested (`--export`). Each CSV export is linked to the complete set of runtime parameters (scenario, N, Δt, θ, ε, seed, integrator, steps). The primary practical risk is high GPU load; mitigation is provided through adjustable N and configurable step limits.

2.12 Reproduciblity and traceability

Reproducibility is ensured through:

- deterministic initial condition generation (`std::mt19937`, seed recorded; default 42),
- complete CLI parameter logging at startup (via spdlog),
- pinned third-party dependency versions through CMake FetchContent,
- logging of GPU adapter name and selected WebGPU backend,
- CSV export containing per-step diagnostics and timing: `step`, `time`, `kinetic_energy`, `potential_energy (if N<=5000)`, `total_energy`, `energy_drift`, `px`, `py`, `pz`, `tree_build_ms`, `force_ms`, `integrate_ms`,
- dual build targets (native + browser) produced from the same source, enabling cross-environment comparison.

References

Aarseth, S., Henon, M., & Wielen, R. (1974). A Comparison of Numerical Methods for the Study of Star Cluster Dynamics. *\aap*, *37*(1), 183–187.

Barnes, J., & Hut, P. (1986). A hierarchical O(N log N) force-calculation algorithm. *Nature*, *324*(6096), 446–449. https://doi.org/10.1038/324446a0

Binney, J., & Tremaine, S. (2008). *Galactic Dynamics: Second Edition* (REV - Revised, 2). Princeton University Press. http://www.jstor.org/stable/j.ctvc778ff

Burtscher, M., & Pingali, K. (2011). An Efficient CUDA Implementation of the Tree-Based Barnes Hut n-Body Algorithm. *GPU Computing Gems Emerald Edition*, . https://doi.org/10.1016/B978-0-12-384988-5.00006-1

Congote, J., Segura, A., Kabongo, L., Moreno, A., Posada, J., & Ruiz, O. (2011). Interactive visualization of volumetric data with WebGL in real-time. *Proceedings of the 16th International Conference on 3d Web Technology*, 137–146. https://doi.org/10.1145/2010425.2010449

Fluke, C. J., Barnes, D. G., Barsdell, B. R., & Hassan, A. H. (2011). Astrophysical Supercomputing with GPUs: Critical Decisions for Early Adopters. *Publications of the Astronomical Society of Australia*, *28*(1), 15–27. https://doi.org/10.1071/as10019

Gaburov, E., Bédorf, J., & Portegies Zwart, S. (2010). Gravitational tree-code on graphics processing units: Implementation in CUDA. *Procedia Computer Science*, *324*, 1119–1127. https://doi.org/10.1016/j.procs.2010.04.124

GPU for the Web Working Group. (2026,). *WebGPU (GPU for the Web) Specification*.

Karras, T. (2012). Maximizing parallelism in the construction of BVHs, octrees, and k-d trees. *Proceedings of the Fourth ACM SIGGRAPH / Eurographics Conference on High-Performance Graphics*, 33–37.

Khronos Group. (2024,). *WebGPU Specification*.

Kreyszig, E. (2011). *Advanced Engineering Mathematics* (10th ed.). Wiley.

LF, G., & Rokhlin, V. (2001). A Fast Algorithm for Particle Simulation. *Journal of Computational Physics*, *73*, 325–348. https://doi.org/10.1016/0021-9991(87)90140-9

Nyland, L., Harris, M., & Prins, J. (2009). Fast N-body simulation with CUDA. *GPU Gem, Vol. 3*, 677–695.

Owens, J. D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A. E., & Purcell, T. J. (2007). A Survey of General-Purpose Computation on Graphics Hardware. *Computer*

Graphics Forum, *26*(1), 80–113. https://doi.org/https://doi.org/10.1111/j.1467-8659.2007.01012.x

Owens, J., Houston, M., Luebke, D., Green, S., Stone, J., & Phillips, J. (2008). GPU computing. *Proceedings of the IEEE*, *96*, 879–899. https://doi.org/10.1109/JPROC.2008.917757

Salmon, J. K., & Warren, M. S. (1994). Skeletons from the Treecode Closet. *Journal of Computational Physics*, *111*(1), 136–155. https://doi.org/https://doi.org/10.1006/jcph.1994.1050

Sengupta, S., Wu, N., Varvello, M., Jana, K., Chen, S., & Han, B. (2025). From WebGL to WebGPU: A Reality Check of Browser-Based GPU Acceleration. *Proceedings of the 2025 ACM Internet Measurement Conference*, 1018–1024. https://doi.org/10.1145/3730567.3764504

Springel, V. (2005). The cosmological simulation code gadget-2. *Monthly Notices of the Royal Astronomical Society*, *364*(4), 1105–1134. https://doi.org/10.1111/j.1365-2966.2005.09655.x

Sung, N.-J., Ma, J., Kim, T., Choi, Y.-j., Choi, M.-H., & Hong, M. (2025,). *Real-Time Cloth Simulation Using WebGPU: Evaluating Limits of High-Resolution*. https://arxiv.org/abs/2507.11794

Sutter, H. (2005). The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, *30*(3), 202–210. http://www.gotw.ca/publications/concurrency-ddj.htm

Usta, Z. (2024). WEBGPU: A NEW GRAPHIC API FOR 3D WEBGIS APPLICATIONS. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 377–382. https://doi.org/10.5194/isprs-archives-XLVIII-4-W9-2024-377-2024

Wang, Q. (2021). A hybrid Fast Multipole Method for cosmological N-body simulations. *Research in Astronomy and Astrophysics*, *21*(1), 3. https://doi.org/10.1088/1674-4527/21/1/3

Zwart, S. F. P., Belleman, R. G., & Geldof, P. M. (2007). High-performance direct gravitational N-body simulations on graphics processing units. *New Astronomy*, *12*(8), 641–650. https://doi.org/https://doi.org/10.1016/j.newast.2007.05.004