Software Requirements Specification for Project “N-Body Simulation”

---

1. Authors

• Anisimov Vladimir Sergeevich
• Bukhanov Bogdan Pavlovich
• Kiryushin Ivan Sergeevich
• Peshkov Alexey Maksimovich

---

2. Introduction

This project aims to develop a physical simulator for studying the gravitational interaction between multiple bodies.

The simulator is organized as a two-stage workflow: from an initial configuration it computes the system’s state after an arbitrary time horizon and then produces a time-resolved playback (video) of the resulting motion.

The primary focus is algorithmic efficiency and time-integration design for N-body dynamics on a single workstation. Fast force evaluation (baseline $O(N^2)$, Barnes–Hut/octree) is paired with adaptive, hierarchical time stepping and sub-cycling for close encounters, partitioning the evolution into well-controlled time slices to preserve accuracy and stability over long horizons.

Key functionality:

- Accurate modeling of Newtonian gravitational forces with softening.
- Batch simulation from initial conditions to a specified end time; optional checkpoints.
- Post-processing pipeline that renders frames and composes a video timeline for playback.

---

3. Glossary

Term	Definition
N-body problem	Motion of N particles under mutual Newtonian gravity.
Barnes–Hut	Tree-based approximation algorithm reducing $O(N^2)$ force computation to $O(N \log N)$.
Softening (ε)	Parameter preventing singularities at small separations.
Headless run	Non-graphical batch simulation used to produce results for later rendering.
Headless mode	Simulation mode without graphics, used for benchmarks or data export.

4. Actors

Actor	Role	Goals / Responsibilities
Student / Researcher	End-user	Prepare initial conditions, run batch simulations to target time, generate video and data for analysis.
Instructor / Demonstrator	Presenter	Use predefined scenarios to illustrate gravitational phenomena in class or talks.
Developer	Developer	Develop, optimize, and maintain the simulation software, ensure the system runs efficiently, fix bugs, and implement new features as needed.

5. Functional requirements

5.1. Strategic Use-cases

- **UC-S-1:** Initial condition preparation and validation
- **UC-S-2:** Batch simulation to target horizon
- **UC-S-3:** Rendering and video export; reproducibility
- **UC-S-4:** Achieve the maximum feasible body count on the available hardware under time, memory, and accuracy constraints.

5.2. Use-cases for Student / Researcher

Use-case UC-1-1: Create a new simulation

Actors: Student / Researcher
Goal: Define initial conditions and simulation parameters for a batch run.
Preconditions: Application running; default project/scene loaded.
Main success scenario:

1. User opens "New Simulation."
2. Specify bodies (count, masses, positions, velocities) and other data of system in the input HDF5 file.
3. Set total duration T_end.
4. Confirm to create the internal model and persist configuration.

Use-case UC-1-2: Load initial conditions

Actors: Student / Researcher
Goals: Load initial conditions from file for a batch simulation.
Preconditions: Application running.
Main success scenario:

1. User opens an input file.

2. The program validates schema and units; loads data.

3. User confirms; the system builds the internal model.

Alternative scenario "A":
Trigger: User opens a file with invalid data (start from step 2).

1. Parser reports mismatches; the UI shows a clear error and suggests fixes; user retries..

Use-case UC-1-3: Render timeline and export video

Actors: Student / Researcher
Goals: Produce a video playback of the computed motion.
Main success scenario:

1. User selects a dataset or checkpoint range for rendering.

2. Configures visualization (camera path, scale, color mapping, trails, overlays, FPS, resolution).

3. System renders frames off-screen and composes a video file.

4. Exported assets include video and visualization metadata for reproducibility.

Rendering metrics (captured in test scenarios):

• Total render time; average FPS; time per frame; encode time.
• Dropped/retimed frames (if any); peak VRAM usage (if available).
• Recorded context: resolution, target FPS, overlays enabled, GPU/device and driver/runtime versions.
• Output data: indicators are recorded in a separate file.

5.3. Use-cases for Instructor

Use-case UC-2-1: Demonstrate physical concepts

Actors: Instructor
Goals: Demonstrate physical concepts
Main success scenario:

1. Select a predefined scenario (two-body, cluster, disk/merger).

2. Run batch simulation to T_end.

3. Optionally render multiple variants to compare ε, or mass ratios.

5.4. Use-cases for Developer

Use-case UC-3-1: Run batch simulation to T_end

Actors: Developer
Goals: Compute system evolution from initial state to the specified time horizon.
Main success scenario:

1. User starts the batch run in headless mode.
2. The engine evaluates forces ($O(N^2)$ or Barnes–Hut) and integrates motion with the selected time-stepping policy (adaptive/hierarchical where enabled). Optionally exports images or video frames.
3. The system saves periodic checkpoints and a final trajectory dataset (states vs time).
4. On completion, a run report with metadata (seed, integrator, tolerances, commit/version) is produced.

Performance metrics (captured in test scenarios):

• Total wall-clock time; per-phase timings: tree build, force evaluation, integration, checkpoint I/O.
• Throughput: steps/s and bodies steps/s.
• Memory: peak RSS; when applicable, peak GPU memory.
• Output data: indicators are recorded in a separate file.

---

6. System-wide functional requirement

- **FR-SYS-01:** Import/export initial conditions and results.

- **FR-SYS-02:** Store seeds, units, integrator metadata, tolerances, and commit/version info for reproducible runs.

- **FR-SYS-03:** Provide a headless simulation mode and a separate rendering tool; GUI must not block the simulation.

- **FR-SYS-04:** Support predefined scenarios and user presets.

- **FR-SYS-05:** Support video export with configurable resolution, frame rate, and overlays (time, scale bar).

7. Non-functional requirement

7.1 Environment

- OS: Windows 10+, Ubuntu 22.04+.
- CPU: x86-64, ≥ 4 cores.
- GPU (optional): NVIDIA GPU with CUDA support (Compute Capability ≥ 5.0).
- Drivers: NVIDIA driver with CUDA Toolkit (например, CUDA 12.x).
- Toolchain: C++17+, CMake; rendering via OpenGL/SDL/SFML.
- Build modes: CPU_ONLY, CUDA (CMake options).

7.2 Performance

CPU baseline: With Barnes–Hut, N=10k finishes a 10k-step batch in practical time; off-screen render ≥ 30 FPS @1080p.

CUDA path: Offload force eval/tree to GPU; target ≥ 3× speedup vs CPU for N ≥ 50k;
minimize and overlap H2D/D2H transfers (streams, async copies);
per-device auto-tuning (block/grid size, shared memory usage).

7.3 Reliability & Accuracy

- Deterministic per backend (CPU or specific CUDA device/configuration).

- GPU vs CPU parity: L2 rel. error ≤ 1e−6 over 10^4 steps on standard tests.

- Energy drift ≤ 1×10^{-3} (Leapfrog, appropriate ε).

- Stable 1-hour batch run; CUDA errors trapped with clear messages; watchdog fallback to CPU.

7.4 Portability & Fallback

- Device selection by platform/device; sensible default to fastest discrete GPU.

- If CUDA unavailable, automatically run CPU path (logged).

- Precision policy: FP32 by default; FP64 when available or requested.

- Kernels avoid vendor-specific extensions by default; guarded optimizations when present.

7.5 Extensibility & Observability

- Swappable CPU/CUDA backends; clean interfaces for evaluators/integrators.
- Configurable and persisted knobs (θ, ε, Δt policy, WG sizes).
- Unit tests with CPU oracle; timing/memory metrics exported (e.g., JSON)