Architectural and Optimization Blueprint for Cosmological Physics Engines

Architectural Transition: JavaScript and TypeScript Refactoring for Physics Runtimes

The modernization of high-performance cosmological simulators requires a rigorous evaluation of code architecture, resource management, and execution environments. Moving a web-based physics engine from a legacy JavaScript architecture to a modern, robust framework addresses the critical bottleneck of execution overhead in browser runtimes.[1] Legacy JavaScript, while flexible, suffers from dynamic typing overhead, unpredictable optimization paths in Just-In-Time (JIT) compilers, and latency spikes induced by automated garbage collection during intensive simulation frames.[1] Transforming the engine code into TypeScript introduces compile-time type safety, structured modular patterns, and clear definitions of physical state boundaries.[1] This architectural change ensures that data containers representing particles, gravitational matrices, and coordinate vectors compile down to highly optimized, monomorphic execution paths in the JIT compiler.[1]

To eliminate memory thrashing—where the continuous instantiation and destruction of coordinate vectors triggers browser garbage collection—the engine must employ pre-allocated array buffers and pool patterns.[1] Memory profiling workflows under Chrome DevTools or similar runtimes reveal that allocating memory within the 60Hz update loop is the leading cause of frame rate instability.[1] Decoupling the simulation mechanics is achieved by implementing an asynchronous, thread-isolated API structure inspired by modern engine designs, such as the Godot Physics Server API.[3] Under this model, the main execution thread manages the interface, coordinate rendering, and user inputs, while a dedicated physics background thread handles state integrations and collision detections.[3]

For advanced trajectory planning, orbit control modeling, and automated navigation, the physics runtime can be wrapped as an opaque state propagator, similar to the Vortex physics engine framework.[5] Within this abstraction, complex physical states are treated opaquely, meaning their dimensional and internal structures are hidden and instead mapped via distance metrics and low-dimensional projections.[5] This encapsulation allows the physics engine to interface directly with kinematic planning algorithms such as Kinematic Planning by Interior-Exterior Cell Exploration (KPIECE), Single-Query Bi-Directional Lazy Collision-Free Planner (SBL), and Expansive Space Trees (EST), where the planner drives vehicle actuators by mapping throttle, braking, and steering forces directly through the simulator's control APIs.[5] Regressions and stability are monitored using structured functional and performance tests to compare

physical outputs across engine iterations, utilizing a modern Forward+ rendering pipeline to ensure visual fidelity during real-time playback.[4]

Algorithmic Acceleration of Gravitational N-Body Solvers

Simulating high-density stellar regions or galactic clusters requires bypassing the quadratic scaling limit of direct-sum gravity solvers, where the computation of pairwise gravitational forces scales as $O(N^2)$.[6] The physics engine implements spatial partitioning algorithms to reduce this complexity to $O(N \log N)$.[2] By recursively dividing two-dimensional spatial coordinates into a quadtree, or three-dimensional systems into an octree, clusters of distant stars can be approximated as a single, combined center of mass.[6] The mathematical criterion for determining if a spatial node is sufficiently distant from a target body is governed by the ratio of the node width s to the distance d between the target body and the node's center of mass.[6] If this ratio falls below a defined accuracy threshold parameter θ, the node is treated as a single mass [6]:

$$\frac{s}{d} < \theta$$

Selecting the threshold parameter θ requires balancing execution speed and coordinate accuracy.[6] While high-precision astrophysics solvers use $\theta = 0.5$ to minimize integration errors, real-time web applications written in JavaScript are optimized by setting $\theta \approx 1.0$.[7] Performance profiling shows that for simulations with fewer than 6,000 points, the overhead of building the spatial tree and recalculating the center of mass at each step exceeds the benefits of the approximation.[7] Beyond 6,000 points, the $O(N \log N)$ curve scales much better than the direct-sum approach, maintaining interactive frame rates with an average coordinate deviation of less than 5% of a single pixel.[7]

When compiling the engine for parallel architectures, developers must optimize the tree traversal code.[2] Traditional recursive traversal causes thread divergence and stack overflows on parallel hardware like GPUs due to limited register space.[2] Parallel implementations avoid these bottlenecks by:

1. Converting spatial coordinates into 30-bit Morton codes to sort particles along a space-filling Z-curve, ensuring that adjacent threads process spatially contiguous particles to maximize cache hits.[9]
2. Replacing recursive traversal functions with an explicit, fixed-size stack array within the thread's local register memory.[9]
3. Pre-allocating the octree structure inside a flat, contiguous typed array rather than using

dynamically allocated objects, which bypasses cache thrashing and register allocation bottlenecks.[2]

Alternative Relativistic and Cosmological Physics Subsystems

A robust cosmological simulator can be enhanced by allowing users to toggle between different physical models, providing an interactive laboratory to compare standard general relativity against alternative and historic theories.

Tired Light Cosmological Redshift

To offer alternatives to standard cosmic expansion, the engine can implement Fritz Zwicky's Tired Light hypothesis.[10] In this model, redshift is not caused by the Doppler stretching of space, but by photons continuously losing energy as they interact with the intergalactic medium (IGM).[11] The energy decay over a distance r is modeled as:

$$E(r) = E_0 e^{-Kr}$$

where E_0 is the initial photon energy and K is the energy attenuation coefficient.[10] Using the relation $E = \frac{hc}{\lambda}$, the wavelength shifts over distance according to:

$$\lambda(r) = \lambda_e e^{Kr}$$

where λ_e is the emitted wavelength, yielding the cosmological redshift relation [10]:

$$z = \frac{\lambda(r)}{\lambda_e} - 1 = e^{Kr} - 1$$

Expanding this relationship for low-redshift systems ($Kr \ll 1$) yields $Kr \approx z$, reproducing Hubble's Law when the attenuation coefficient K is set to the Hubble constant divided by the speed of light ($K = \frac{H_0}{c}$).[10]

An alternative tired light formulation treats the intergalactic medium as an ultra-low-density refracting fluid with an effective index of refraction $n_{\text{IGM}} = 1 + \epsilon$.[12] As a photon encounters N absorption and re-emission events over a path length x, it experiences a cumulative propagation delay proportional to the characteristic delay t_0 of the medium [12]:

$$t_2 = t_1 + Nt_0 = \frac{x}{c} + Nt_0$$

This delay gives the photon an effective velocity $v = \frac{x}{t_2}$ and an index of refraction $n = 1 + \epsilon = \frac{c}{v}$.[12] This model links the observed redshift directly to the physical properties of the intergalactic medium.[12]

The photon's energy loss can also be defined by an energy attenuation coefficient $\beta = -\frac{\ln(f)}{\text{ly}}$, where f is the fraction of energy transferred to the intergalactic medium per light-year, and an attenuation time constant $\tau = \beta c$.[11] This yields the time-dependent energy equation [11]:

$$E_t = E_0 e^{-\tau t}$$

This decay can be characterized by an attenuation half-time:

$$\theta_{1/2} = \frac{0.692}{\tau} = \frac{0.692}{\beta c}$$

Unlike radioactive decay, where particles spontaneously disintegrate, tired-light photons lose energy continuously while remaining intact, presenting an alternative explanation for cosmic redshift in a static universe.[11]

Quantized Inertia Modeling

To simulate stellar orbits and galactic rotations without adding dark matter, the engine can incorporate Michael McCulloch's Quantized Inertia (QI) theory.[13] QI suggests that inertial mass is not an inherent property of matter but rather arises from Unruh radiation produced by accelerating bodies.[13] The temperature T of this thermal Unruh radiation is given by [15]:

$$T = \frac{\hbar a}{2\pi c k}$$

where a is the magnitude of acceleration, $\hbar$ is the reduced Planck constant, c is the speed of light, and k is the Boltzmann constant.[15] This radiation is bounded by a relativistic Rindler horizon in the direction opposite to the acceleration vector and by the cosmic Hubble horizon in all directions.[13]

At extremely low accelerations, the Unruh wavelengths become so long that they are damped by the Hubble horizon, reducing the inertial mass of the body.[13] The modified inertial mass m_i is expressed as [13]:

$$m_i = m\left(1 - \frac{2c^2}{|a|\Theta}\right)$$

where Θ is the co-moving cosmic diameter (8.8×10^{26} meters) and $|a|$ is the acceleration relative to surrounding matter.[13]

In orbiting systems, this equation can be rewritten by defining the acceleration gravitationally, $a = \frac{GM}{r^2}$, yielding [13]:

$$m_i = m\left(1 - \frac{2c^2r^2}{GM\Theta}\right)$$

where M is the central mass, G is the gravitational constant, and r is the orbital radius.[13] As the orbital radius increases, the mutual acceleration decreases.[13] As the acceleration approaches the minimum cosmic limit $a_{\min} = \frac{2c^2}{\Theta} \approx 2 \times 10^{-10}\ \mathrm{m/s^2}$, the inertial mass m_i approaches zero.[13] This mass collapse causes the gravitational acceleration to increase again, establishing a stable equilibrium that explains flat galactic rotation curves and stellar velocities in wide binary systems like Proxima Centauri without requiring dark matter.[13]

Woodward-Mach Effect Propulsion Simulations

For engines featuring spacecraft simulations, developers can implement James Woodward's Mach Effect Thruster (MET) equations to simulate propellantless propulsion.[16] Derived from linearized general relativity under weak-field approximations ($g_{\mu\nu} \approx \eta_{\mu\nu} + h_{\mu\nu}$), gravity can be modeled using gravito-electromagnetism (GEM) Maxwell-like field equations and potentials [17]:

$$E = -\nabla\phi - \frac{1}{c}\frac{\partial \mathbf{A}}{\partial t}$$

$$B = \nabla \times \mathbf{A}$$

$$\nabla \cdot \mathbf{A} + \frac{4}{c}\frac{\partial \phi}{\partial t} = 0$$

where $\mathbf{A}$ is the gravitational vector potential, ϕ is the scalar potential, and the final equation is the Lorentz gauge condition.[17] Under proper acceleration, an object that absorbs internal energy experiences a transient mass fluctuation δm_0 given by [18]:

$$\delta m_0 - \frac{1}{4\pi G}\left[\frac{1}{\rho_0 c^2}\frac{\partial P}{\partial t} - \left(\frac{1}{\rho_0 c^2}\right)^2 \frac{P^2}{V}\right]$$

where ρ_0 is the proper density of the dielectric material, V is its volume, and P is the instantaneous power delivered to the system.[18]
By driving a capacitor (the fluctuating mass) and a piezoelectric actuator (such as lead-zirconate-titanate) in phase, the engine produces a rectified, time-averaged force [16]:

$$F_{\text{net}} - \langle \delta m_0(t) a(t) \rangle \neq 0$$

This allows the simulator to model reactionless propulsion for orbital maneuvers.[16] To ensure realism, the engine must also model common experimental failure modes, including the Dean effect (spurious signals caused by vibrations) and thermal expansion that detunes the system's resonant frequency.[19]

Astronomical Interfaces, Coordinate Transformations, and Lightcone Rendering

Converting raw physics coordinates into realistic star maps requires accurate calculations of distances and angular sizes.[20] The engine maps linear physical coordinates to angular observations by converting arcseconds and parsecs.[20] The distance d_{pc} in parsecs is calculated from the parallax angle θ_{arcsec} in arcseconds [20]:

$$d_{\text{pc}} = \frac{1}{\theta_{\text{arcsec}}}$$

One parsec equals approximately 3.08×10^{16} meters, 3.26 light-years, or 2.06×10^5 Astronomical Units (AU), where $1\ \text{AU}$ is the mean distance from the Earth to the Sun (1.496×10^{11} meters).[21] For high-resolution planet and terrain simulations, spatial coordinates can be mapped to downscaled global datasets (such as CHELSEA or ETOPO) at resolutions of 30 or 60 arcseconds.[22]
For deep-space rendering, the engine constructs a lightcone centered on the observer.[24] This lightcone slices spatial partitions of the simulation to match the travel time of light, mapping simulated stars as Single Stellar Populations (SSPs).[24] The engine computes k-corrections and intergalactic medium absorption to calculate the observed flux through standard filter bands.[24] Visual accuracy can be enhanced by applying point spread function (PSF) blurring and background thermal noise to match high-precision space telescopes.[24] For example, the engine can mimic the James Webb Space Telescope, which features a pointing precision of 1

arcsecond and isolates vibrations to 2 milliarcseconds.[25]
To calculate exposure settings, the engine measures stellar fluxes in surface brightness units (flux per square arcsecond).[26] This is particularly useful for simulating planetary systems, such as Neptune or Uranus sequences, where the surface brightness determines the necessary exposure times and gain settings [26]:

$$\text{Flux}_{\text{planet}} = \text{Surface Brightness} \times \text{Apparent Area in arcsec}^2$$

Vast distances can be navigated using a logarithmic depth buffer, which dynamically scales coordinate precision from meters to gigaparsecs.[27]

Structural Data Tables

The following tables synthesize the physical models, coordinate mappings, and control systems needed to implement these features in a cosmological physics engine.

Table 1: Redshift-Distance-Time Lookup Matrix

To optimize rendering speeds, the engine can pre-compute cosmological distances and light travel times using a lookup table based on redshift (z) values.[28]

Redshift Parameter (z)	Light Travel Time (Years)	Current Co-Moving Distance (Light-Years)
0.0000715	$1.000 \times$ (1 Million) [28]	$1.000 \times$ (1 Million) [28]
0.10	$1.286 \times$ (1.286 Billion) [28]	$1.349 \times$ (1.349 Billion) [28]
0.25	$2.916 \times$ (2.916 Billion) [28]	$3.260 \times$ (3.260 Billion) [28]
0.50	$5.019 \times$ (5.019 Billion) [28]	$5.936 \times$ (5.936 Billion) [28]
1.00	$7.731 \times$ (7.731 Billion) [28]	$1.015 \times$ (10.147 Billion) [28]
2.00	$1.032 \times$ (10.324	$1.542 \times$ (15.424

	Billion) [28]	Billion) [28]
4.00	$1.209 \times$ (12.094 Billion) [28]	$2.075 \times$ (20.745 Billion) [28]
10.00	$1.318 \times$ (13.184 Billion) [28]	$2.660 \times$ (26.596 Billion) [28]

Table 2: Comparative Framework of Physical Models

This table compares the mathematical structures, scaling behaviors, and physical assumptions of the various engines supported by the simulator.

Physics Model	Mathematical Core	Key Variables	Scale Limitations	Physical Assumptions
Barnes-Hut Gravity	$\frac{s}{d} <$ [6]	s: Node width, d: Distance, θ: Accuracy [6]	Ineffective for $N <$ [7]	Classic Newtonian gravity and baryonic matter [6]
Standard Redshift	$z =$ [12]	λ_o: Observed, λ_e: Emitted wavelength [12]	Bound by Hubble horizon [28]	Expanding universe and FLRW metric [28]
Tired Light	$z = e^{Kr} -$ [10]	K: Attenuation, r: Path length [10]	High error at large z	Static universe and energy-attenuating medium [11]
Quantized Inertia	$m_i =$ [13]	a: Acceleration, Θ: Cosmic diameter [13]	Dominates at $a <$ [13]	Inertia caused by horizon-bounded Unruh waves [13]
Woodward	$\delta m_0 \propto \frac{\partial P}{\partial t} -$	P: Power, ρ_0:	Requires high-frequency	Transient mass fluctuations in

Effect	[18]	Density, V: Volume [18]	power [19]	accelerated dielectrics [16]

Table 3: Physics Engine API Architecture and Interface Modality

This table outlines the decoupled API architecture, navigation controls, and state management interfaces.

System Component	Interface & Control Mapping	Data Structure	Performance Mechanism
Physics Server API	Thread-isolated low-level API calls [3]	contiguous pre-allocated array buffers [2]	Offloads physics calculations to background web workers [3]
Planning Wrapper	applyControl(double *control) mapping [5]	Opaque state coordinates with distance metrics [5]	Interfaces directly with motion planners (KPIECE, SBL, EST) [5]
Desktop Navigation	Left-mouse drag (Orbit), Middle-mouse drag (Pan), Scroll-wheel (Zoom), Home (Reset) [29]	Logarithmic camera matrix coordinates [27]	Prevents rendering artifacts at extreme zoom ranges [27]
Trackpad Navigation	Left-mouse drag (Orbit), Shift + Left-mouse drag (Pan), Right-mouse drag (Zoom), Alt + '/' (Reset) [29]	Logarithmic camera matrix coordinates [27]	Maintains precise navigation across different inputs [29]

Conclusions and Actionable Engineering Recommendations

To improve the performance, accuracy, and depth of the cosmological physics engine, developers should focus on three primary upgrades:

First, performance bottlenecks in JavaScript can be minimized by replacing dynamic objects with flat, contiguous memory structures.[2] Pre-allocating parallel arrays to store particle states (positions, velocities, and masses) prevents garbage collection overhead and optimizes memory access patterns.[2] If the engine is compiled for GPUs, constructing spatial indices using Morton codes will group nearby particles, reducing thread divergence and speeding up parallel tree traversals.[9]

Second, the simulation's depth can be enhanced by allowing users to toggle between standard and alternative cosmological models. Integrating alternative physics theories—such as Zwicky's tired light redshift, McCulloch's Quantized Inertia, and Woodward's Mach Effect—provides a unique and educational environment for testing these models against observational data.[12]

Third, to support navigating across vast distance scales, the engine should implement a logarithmic depth buffer in the rendering pipeline.[21] Standardizing the camera controls to support both desktop and trackpad inputs ensures a smooth user experience when zooming from planetary surfaces to intergalactic space.[29]

Works cited

1. PhysicsEngine[v1.7.0] Demo : r/learnjavascript - Reddit, accessed May 28, 2026, https://www.reddit.com/r/learnjavascript/comments/1t7l22a/physicsenginev170_demo/
2. Optimizing N-Body Simulation with Barnes-Hut Algorithm and CUDA | Medium, accessed May 28, 2026, https://medium.com/@hsinhungw/optimizing-n-body-simulation-with-barnes-hut-algorithm-and-cuda-c76e78228c28
3. Little demo with the Physics Server API : r/IndieDev - Reddit, accessed May 28, 2026, https://www.reddit.com/r/IndieDev/comments/1qqvbyr/little_demo_with_the_physics_server_api/
4. 3D Physics Tests Demo - Godot Asset Library, accessed May 28, 2026, https://godotengine.org/asset-library/asset/2747
5. Planning using the Vortex physics engine - Kavraki Lab, accessed May 28, 2026, https://ompl.kavrakilab.org/2012/04/27/planning-using-the-vortex-physics-engine.html
6. Barnes–Hut simulation - Wikipedia, accessed May 28, 2026, https://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation
7. The Barnes-Hut Approximation, accessed May 28, 2026, https://jheer.github.io/barnes-hut/
8. The Barnes-Hut Algorithm - arbor.js, accessed May 28, 2026, https://arborjs.org/docs/barnes-hut
9. Barnes-Hut CUDA Simulation Performance - NVIDIA Developer Forums, accessed May 28, 2026, https://forums.developer.nvidia.com/t/barnes-hut-cuda-

simulation-performance/344140
10. Tired light hypothesis question deriving the distance redshift relation, accessed May 28, 2026, https://physics.stackexchange.com/questions/451135/tired-light-hypothesis-question-deriving-the-distance-redshift-relation
11. The Tired-Light Hypothesis: Derivations of Basic Relations, accessed May 28, 2026, https://www.gsjournal.net/Science-Journals/Communications-Cosmology/Download/9433
12. An Alternative Explanation for Cosmological Redshift - arXiv, accessed May 28, 2026, https://arxiv.org/pdf/0706.2885
13. Testing quantized inertia on Proxima Centauri | Monthly Notices of the Royal Astronomical Society - Oxford Academic, accessed May 28, 2026, https://academic.oup.com/mnrasl/article/532/1/L67/7682393
14. Quantised inertia from relativity and the uncertainty principle - ResearchGate, accessed May 28, 2026, https://www.researchgate.net/publication/309102743_Quantised_inertia_from_relativity_and_the_uncertainty_principle
15. Superluminal Travel from Quantised Inertia - TSI Journals, accessed May 28, 2026, https://www.tsijournals.com/articles/superluminal-travel-from-quantised-inertia.pdf
16. The Woodward Effect: Math Modeling and Continued Experimental Verifications at 2 to 4 MHz - SciSpace, accessed May 28, 2026, https://scispace.com/pdf/the-woodward-effect-math-modeling-and-continued-experimental-4t9whuhruq.pdf
17. (PDF) Theory of a Mach Effect Thruster I - ResearchGate, accessed May 28, 2026, https://www.researchgate.net/publication/269207998_Theory_of_a_Mach_Effect_Thruster_I
18. Woodward Effect - Encyclopedia.pub, accessed May 28, 2026, https://encyclopedia.pub/entry/34880
19. The Woodward effect and the dream of non-newtonian propulsion - INAF IASF-Milano, accessed May 28, 2026, https://www.iasf-milano.inaf.it/Astro-Siesta/astro_danielep_2.pdf
20. Converting Arcseconds to Parsecs - Lesson - Study.com, accessed May 28, 2026, https://study.com/academy/lesson/converting-arcseconds-to-parsecs.html
21. Astronomical Distances - AQA A Level Physics Revision Notes - Save My Exams, accessed May 28, 2026, https://www.savemyexams.com/a-level/physics/aqa/17/revision-notes/9-astrophysics/9-2-classification-of-stars/9-2-3-astronomical-distances/
22. PIFSC-Protected-Species-Division/picMaps: Mapping utilities for PIFSC - GitHub, accessed May 28, 2026, https://github.com/PIFSC-Protected-Species-Division/picMaps
23. custom dataset • pastclim - GitHub Pages, accessed May 28, 2026, https://evolecolgroup.github.io/pastclim/articles/a2_custom_datasets.html

24. flaminiafortuni/FORECAST - GitHub, accessed May 28, 2026, https://github.com/flaminiafortuni/FORECAST
25. James Webb Space Telescope - Wikipedia, accessed May 28, 2026, https://en.wikipedia.org/wiki/James_Webb_Space_Telescope
26. nasa-jpl/cgi-eetc: CGI engineering exposure time calculator (eetc) - GitHub, accessed May 28, 2026, https://github.com/nasa-jpl/cgi-eetc
27. Arcseconds to Parsecs converter | Cosmology calculator to transform between angular size and linear size, accessed May 28, 2026, http://arcsec2parsec.joseonorbe.com/
28. Redshift - Las Cumbres Observatory, accessed May 28, 2026, https://lco.global/spacebook/light/redshift/
29. cjhosken/ap_physics_engine: AP Physics Engine is a 3D Java-based program that allows you to simulate concepts that are learnt in AP Physics 1. - GitHub, accessed May 28, 2026, https://github.com/cjhosken/ap_physics_engine