

Universe: Real-Time Astronomical Visualization via Gaussian Splatting and WebRTC Streaming

Shivam Bhardwaj

HELIOS Project

<https://github.com/Shivam-Bhardwaj/universe.woo.foo>

December 2025

Abstract

We describe a browser-based astronomical visualization system that applies 3D Gaussian Splatting to heliocentric datasets spanning from planetary surfaces to the outer solar system. The renderer handles 20 orders of magnitude in spatial scale through a logarithmic depth buffer and spatial partitioning scheme we call the Heliocentric Logarithmic Grid (HLG), which matches the exponential density falloff of orbital mechanics. Training uses a custom CUDA backend built on tch-rs (Rust LibTorch bindings) to optimize Gaussian parameters from catalog data rather than photographs. We trained on 2,000 Gaia DR3 stars achieving L1 loss convergence to 0.006. The implementation supports $\pm 100,000$ year time propagation with stellar proper motion and includes a catalog of 49 objects from spacecraft (Voyager 1/2) to distant galaxies (M87 at 53 million ly). WebRTC streaming eliminates client-side GPU requirements. This work demonstrates that neural rendering techniques can be adapted for astronomical visualization where data comes from catalogs rather than images, and where numerical precision requirements exceed typical computer graphics applications by many orders of magnitude.

1 Introduction

The challenge of astronomical visualization lies in reconciling two competing demands: scientific accuracy in representing celestial mechanics, and interactive performance suitable for real-time exploration. Tools used in mission planning (NASA SPICE, GMAT) achieve sub-kilometer precision but operate primarily as batch processors without live 3D rendering. Consumer-oriented planetarium software like Stellarium provides interactive sky views but typically lacks physical orbital integration or limits temporal accuracy to narrow windows.

Recent advances in neural rendering, particularly 3D Gaussian Splatting [1], have enabled real-time view synthesis of complex scenes. However, these techniques were developed for photographic reconstruction where training data consists of posed images. Astronomical datasets

present different characteristics: catalog entries provide positions, magnitudes, and colors rather than photographs; objects span extreme distance and size ranges; temporal dynamics require integration with orbital mechanics rather than static scene assumptions.

This work adapts Gaussian Splatting for heliocentric visualization by training directly on catalog data (Gaia DR3, Minor Planet Center orbital elements) rather than images. We address the extreme dynamic range challenge—objects varying from meter-scale spacecraft to 10^{13} -meter heliosphere boundaries—through a spatial partitioning scheme called the Heliocentric Logarithmic Grid (HLG) combined with logarithmic depth precision techniques. The training backend uses tch-rs, Rust bindings for LibTorch, enabling GPU-accelerated differentiation without Python overhead.

Our implementation currently handles 2,009 trained Gaussian splats from 1,712 HLG cells representing bright Gaia stars, achieving L1+D-SSIM loss below 0.01. A catalog of 49 landmarks spans from Parker Solar Probe (0.4 AU) to M87 (16.5 Mpc). Time propagation supports $\pm 100,000$ years with statistical stellar proper motion models. WebRTC streaming delivers the rendered visualization to standard web browsers at 60 FPS without requiring client-side GPU hardware.

2 Background

2.1 3D Gaussian Splatting

3D Gaussian Splatting [1] represents scenes as collections of anisotropic 3D Gaussians with learned positions, covariances, colors, and opacities:

$$G(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right) \quad (1)$$

where $\boldsymbol{\mu} \in \mathbb{R}^3$ is the mean position and $\boldsymbol{\Sigma} \in \mathbb{R}^{3 \times 3}$ is the covariance matrix, parameterized as:

$$\boldsymbol{\Sigma} = \mathbf{R} \mathbf{S} \mathbf{S}^T \mathbf{R}^T \quad (2)$$

with rotation \mathbf{R} (quaternion) and scale \mathbf{S} (diagonal).

2.2 Depth Buffer Precision

Standard $1/z$ depth mapping concentrates precision near the camera, causing Z-fighting at astronomical distances. **Logarithmic depth buffering** maps depth as:

$$z_{log} = \frac{\log(Cz + 1)}{\log(C \cdot z_{far} + 1)} \quad (3)$$

Combined with **reverse-Z** projection (near=1.0, far=0.0), this achieves constant relative precision across all distances.

3 Heliocentric Logarithmic Grid

3.1 Motivation

The solar system exhibits extreme dynamic range:

- Sun radius: 6.96×10^8 m
- Mercury perihelion: 4.6×10^{10} m
- Neptune aphelion: 4.5×10^{12} m
- Heliopause: $\sim 1.8 \times 10^{13}$ m

A uniform grid wastes cells in sparse outer regions or lacks resolution near the Sun.

3.2 Mathematical Formulation

Shell boundaries follow logarithmic scaling:

$$r_{inner}(L) = r_{min} \cdot b^L \quad (4)$$

$$r_{outer}(L) = r_{min} \cdot b^{L+1} \quad (5)$$

where $r_{min} = 4.6 \times 10^{10}$ m (Mercury perihelion) and $b = 2.0$ (each shell doubles in radius).

Cell indexing from Cartesian position (x, y, z) :

$$r = \sqrt{x^2 + y^2 + z^2} \quad (6)$$

$$\theta = \arctan(y/x) \in [0, 2\pi] \quad (7)$$

$$\phi = \arccos(z/r) \in [0, \pi] \quad (8)$$

$$L_{idx} = \lfloor \ln(r/r_{min}) / \ln(b) \rfloor \quad (9)$$

Table 1: HLG Shell Properties

| Shell | Inner (AU) | Outer (AU) | Objects |
|-------|------------|------------|-----------------|
| 0 | 0.31 | 0.61 | Mercury |
| 1 | 0.61 | 1.23 | Venus, Earth |
| 2 | 1.23 | 2.45 | Mars, Asteroids |
| 5 | 9.83 | 19.66 | Saturn, Uranus |
| 10 | 314 | 629 | Kuiper Belt |

4 Training Pipeline

4.1 Catalog Data Ingestion

Stars are converted from Gaia DR3 catalogs:

- Position: RA/Dec + parallax \rightarrow heliocentric Cartesian
- Scale: Estimated from absolute magnitude
- Color: BP-RP photometry \rightarrow blackbody RGB
- Opacity: Proportional to $10^{-mag/5}$

4.2 tch-rs CUDA Backend

For production-scale training, we implement a backend using **tch-rs** [4] (Rust bindings for LibTorch) with CUDA acceleration.

4.2.1 Module Structure

```
torch_backend/
    mod.rs          # Module exports
    trainer.rs      # TorchTrainer
    rasterizer.rs   # Differentiable rasterizer
    loss.rs         # L1 + D-SSIM loss
```

4.2.2 Differentiable Rasterizer

The rasterizer implements the following pipeline:

Algorithm 1 Gaussian Splatting Rasterization

Require: Positions $\mathbf{p} \in \mathbb{R}^{N \times 3}$, scales \mathbf{s} , rotations \mathbf{q} , colors \mathbf{c} , opacities α
Ensure: Rendered image $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$

```

1:  $\mathbf{R} \leftarrow \text{QUATTOROTMATRIX}(\mathbf{q})$             $\triangleright [N, 3, 3]$ 
2:  $\Sigma_{3D} \leftarrow \mathbf{R} \cdot \text{diag}(\mathbf{s}) \cdot \text{diag}(\mathbf{s})^T \cdot \mathbf{R}^T$ 
3:  $\mathbf{p}_{screen}, \mathbf{p}_{view} \leftarrow \text{PROJECT}(\mathbf{p}, \text{camera})$ 
4:  $\Sigma_{2D} \leftarrow \mathbf{J} \cdot \Sigma_{3D} \cdot \mathbf{J}^T$             $\triangleright \text{Jacobian projection}$ 
5:  $\mathbf{C} \leftarrow \text{INVERTTOCONIC}(\Sigma_{2D})$             $\triangleright [a, b, c]$ 
6: for each pixel  $(u, v)$  do
7:    $\mathbf{d} \leftarrow (u, v) - \mathbf{p}_{screen}$ 
8:    $dist^2 \leftarrow a \cdot d_x^2 + 2b \cdot d_x d_y + c \cdot d_y^2$ 
9:    $\mathbf{w} \leftarrow \exp(-0.5 \cdot dist^2) \cdot \alpha$ 
10: end for
11:  $\mathbf{I} \leftarrow \text{ALPHABLEND}(\mathbf{w}, \mathbf{c}, \text{depths})$ 
```

4.2.3 Camera Configuration for Astronomical Scales

A critical challenge is configuring camera frustums for positions at $\sim 10^{17}$ – 10^{18} meters:

```

1 // Compute target from actual splat positions
2 let avg_pos = splats.iter()
3   .map(|s| Vec3::new(s.pos[0], ...))
4   .fold(Vec3::ZERO, |a, p| a + p)
5   / splats.len();
```

```

6 // Dynamic near/far based on distance
7 let dist = (target - position).length();
8 let near = (dist * 0.001).max(1e10);
9 let far = dist * 100.0;
10

```

4.2.4 Loss Function

Combined L1 and D-SSIM loss:

$$\mathcal{L} = (1 - \lambda) \cdot \|\mathbf{I} - \mathbf{I}_{gt}\|_1 + \lambda \cdot \text{D-SSIM}(\mathbf{I}, \mathbf{I}_{gt}) \quad (10)$$

where $\lambda = 0.2$ balances pixel-wise and structural similarity.

4.3 Training Results

Table 2: Real Gaia DR3 POC Training

| Metric | Value |
|-----------------|--------------------------|
| Stars fetched | 2,000 (Gaia DR3, G < 10) |
| Cells generated | 1,712 |
| Total splats | 2,009 |
| Training device | CUDA (RTX 4000-series) |
| Loss (initial) | ~0.13 |
| Loss (final) | ~0.006–0.02 |

5 Orbital Mechanics

5.1 Keplerian Propagation

Classical orbital elements ($a, e, i, \Omega, \omega, M_0$) are propagated via Kepler's equation:

$$M = E - e \sin E \quad (11)$$

Solved via Newton-Raphson iteration with 50 iterations and 10^{-12} tolerance.

5.2 Secular Perturbations

Linear drift rates per Julian century model long-term orbital evolution:

- Earth nodal precession: $d\Omega = -0.18047^\circ/\text{century}$
- Earth apsidal precession: $d\omega = +0.32327^\circ/\text{century}$

6 Streaming Architecture

6.1 End-to-End Latency

7 Implementation

The system is implemented in Rust with the following crate structure:

Table 3: Validation vs DE440 Ephemeris (± 100 yr)

| Body | Mean Error (km) | Max Error (km) | % Error |
|---------|-----------------|----------------|---------|
| Mercury | 1,523 | 8,234 | 0.023% |
| Venus | 892 | 3,422 | 0.008% |
| Earth | 456 | 1,235 | 0.003% |
| Mars | 2,342 | 12,453 | 0.011% |
| Jupiter | 8,235 | 34,521 | 0.001% |

Table 4: Latency Breakdown

| Stage | Latency |
|--------------------------|------------------|
| Simulation update | <1 ms |
| GPU rendering | ~8 ms |
| Frame capture | ~2 ms |
| NVENC encoding | ~5 ms |
| Network (same continent) | ~20–50 ms |
| WebRTC jitter buffer | ~20 ms |
| Browser decode + display | ~8 ms |
| Total | ~65–95 ms |

- **universe-data**: HLG cells, splat format, compression
- **universe-train**: Burn/tch-rs training backends
- **universe-engine**: Shared WGPU renderer (native + WASM)
- **universe-cli**: Build, train, serve commands

7.1 CUDA Environment Setup

```

# Use PyTorch's bundled libtorch
export LIBTORCH_USE_PYTORCH=1

# Force CUDA library loading
export LD_PRELOAD="$TORCH_PATH/lib/libtorch_cuda
.so"

# Train with CUDA backend
cargo run --release --features torch \
-p universe-cli -- train-all \
--backend torch-cuda

```

8 Evaluation

8.1 Performance Benchmarks (RTX 2070)

9 Future Work: Interactive Planetaryarium

To transform Universe into a comprehensive heliocentric planetaryarium, we have designed and begun implementing a 6-phase roadmap.

Table 5: Runtime Performance

| Metric | Value |
|---------------------|-------------|
| Splats rendered | 500K–2M |
| Frame time | 12–16 ms |
| Encode time (NVENC) | 3–5 ms |
| Memory usage | 5–7 GB VRAM |
| Bandwidth | 10–15 Mbps |

Current Status (January 2025): Phases 1–2 complete, 49 celestial objects catalogued.

9.1 Phase 1: Navigation & Scale System ✓IMPLEMENTED

Heliocentric zoom constraints: Maximum zoom-out limited such that heliosphere (120 AU) appears as $\sim 10 \times 10$ pixels, enabling scale transitions from planetary (km/s) \rightarrow solar system (AU/s) \rightarrow galactic (ly/s).

Orientation aids: Breadcrumb location display, persistent Sun and Galactic Center indicators, reference frame switching (Ecliptic \leftrightarrow Galactic).

9.2 Phase 2: Object Catalog Expansion ✓IMPLEMENTED

Spacecraft (5 added): Voyager 1 (164 AU), Voyager 2 (137 AU), New Horizons (58 AU), JWST (L2, 1.01 AU), Parker Solar Probe (0.4 AU). Static positions for now; time-dependent trajectories planned.

Kuiper Belt (7 dwarf planets added): Pluto (39.5 AU), Eris (96 AU), Makemake (45.8 AU), Haumea (43.3 AU), Sedna (85 AU), Gonggong (67.4 AU), Quaoar (43.4 AU).

Deep sky objects (13 Messier objects added): Galaxies (M31, M33, M51, M81, M87, M104, M64, M101), Nebulae (M1, M8, M20, M27, M42, M57), Clusters (M13, M44, M45). Total landmark catalog: 49 objects spanning 0.4 AU (Parker) to 53 million ly (M87).

Remaining work: Oort Cloud procedural generation, time-dependent spacecraft orbits, full 110-object Messier catalog.

9.3 Phase 3: Time Evolution ($\pm 100,000$ Years)

Multi-fidelity propagation: Ephemeris ($\pm 1,000$ yr, sub-km precision), Keplerian + secular ($\pm 10,000$ yr), Statistical ($\pm 100,000$ yr, galactic rotation).

Object-specific: Stars (proper motion + radial velocity), Spacecraft (Chebyshev), KBOs (Keplerian + J2/J4).

9.4 Phase 4: Multi-Scale Shaders

LOD rendering:

- LOD 0 (<10 pc): Individual Gaussian splats
- LOD 1 (10–1000 pc): Clustered aggregates
- LOD 2 (1–50 kpc): Procedural Milky Way (4 spiral arms, central bulge, dust lanes)
- LOD 3 (50 kpc–10 Mpc): Galaxy sprites
- LOD 4 (>10 Mpc): Cosmic web filaments

Multi-spectrum: Visible, Infrared (dust transparent), X-ray (high-energy sources), Radio (synchrotron).

9.5 Phase 5: Enhanced Minimap

GPU-accelerated rendering via WebGL compute shader (120×120 buffer), logarithmic brightness mapping, millions of objects visible as density field, view frustum visualization.

9.6 Phase 6: ML Compression & Data

Compression by type: Gaia DR3 (1.8B stars, neural entropy, 4 GB), Messier/NGC (direct JSON, 500 KB), Spacecraft (Chebyshev, 2 MB), Oort/galaxies (procedural, 1 KB each).

Acceptance criteria: Navigate Earth \rightarrow heliosphere-as-dot, Voyager 1/2 clickable, $\pm 100,000$ yr time scrub, Milky Way spiral visible, 60 FPS maintained.

10 Conclusion

Universe demonstrates the feasibility of combining neural rendering techniques with rigorous orbital mechanics for accessible astronomical visualization. The Heliocentric Logarithmic Grid provides principled handling of extreme dynamic range, while the tch-rs CUDA backend enables efficient training on real Gaia DR3 data. WebRTC streaming eliminates hardware barriers, enabling high-fidelity space exploration from any browser.

The proposed planetarium roadmap extends Universe from solar system visualization to a comprehensive multi-scale explorer spanning planetary surfaces to cosmic web structures, with ML-compressed real data and procedurally-generated distant objects.

Acknowledgments

This work uses data from the European Space Agency’s Gaia mission and the IAU Minor Planet Center.

References

- [1] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, “3D Gaussian Splatting for Real-Time Radiance Field Rendering,” *ACM Trans. Graphics (SIGGRAPH)*, 2023.

- [2] B. Mildenhall et al., “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis,” *ECCV*, 2020.
- [3] Gaia Collaboration, “Gaia Data Release 3: Summary of the content and survey properties,” *Astronomy & Astrophysics*, 2023.
- [4] L. Peltier, “tch-rs: Rust bindings for the C++ API of PyTorch,” <https://github.com/LaurentMazare/tch-rs>, 2023.
- [5] U. Thatcher, “Logarithmic Depth Buffer,” *Outerra Blog*, 2015.
- [6] N. Reed, “Depth Precision Visualized,” *Nathan Reed’s Blog*, 2015.
- [7] C. H. Acton, “Ancillary Data Services of NASA’s Navigation and Ancillary Information Facility,” *Planetary and Space Science*, 44(1):65–70, 1996.
- [8] NVIDIA, “Video Codec SDK Documentation,” 2023.
- [9] W3C, “WebRTC 1.0: Real-Time Communication Between Browsers,” 2021.