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

HELIOS Project
https://github.com/helios-project/universe

December 2024

Abstract

We present **Universe**, a novel real-time astronomical visualization system combining 3D Gaussian Splatting, high-fidelity orbital mechanics, and WebRTC streaming for interactive, scientifically-accurate solar system exploration from any web browser. Unlike existing planetarium software relying on pre-rendered assets, Universe renders using differentiable neural representations trained on astronomical catalogs (Gaia DR3, MPCORB), propagates orbital positions across 10,000 years using Keplerian mechanics with secular perturbations, and streams rendered output at 60 FPS via hardware-accelerated encoding. The system achieves astronomical-scale rendering (1 meter to 10^{20} meters) without Z-fighting through logarithmic depth buffering, reverse-Z projection, and a novel Heliocentric Logarithmic Grid (HLG) spatial partitioning scheme. We introduce a CUDA-accelerated training backend using tch-rs (LibTorch bindings) achieving convergent loss on real Gaia DR3 data. To our knowledge, this represents the first integration of neural radiance field techniques with astrodynamics simulation for real-time cloud-streamed visualization.

1 Introduction

Astronomical visualization faces a fundamental tension between scientific accuracy and interactive performance. Professional tools like NASA's SPICE toolkit provide sub-kilometer precision but lack real-time visualization. Consumer planetarium software (Stellarium, Celestia) offers interactivity but sacrifices physical accuracy or temporal range.

Universe bridges this gap through:

1. **Neural scene representations** (3D Gaussian Splats) trained directly from astronomical catalogs
2. **Mission-grade orbital mechanics** integrated with real-time rendering
3. **GPU-rendered frame streaming** to browsers without client-side GPU requirements
4. **Numerical precision** across 20 orders of magnitude in spatial scale

1.1 Key Contributions

- **Heliocentric Logarithmic Grid (HLG):** Spatial partitioning using logarithmic radial shells matching astronomical density falloff
- **Astronomical-Scale Neural Rendering:** First application of 3D Gaussian Splatting to heliocentric visualization with logarithmic depth
- **tch-rs CUDA Training Backend:** Differentiable rasterizer using raw LibTorch for CUDA-accelerated optimization
- **Zero-Install Cloud Astronomy:** Browser-accessible exploration via WebRTC streaming

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:

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

where $\boldsymbol{\mu} \in \mathbb{R}^3$ is the mean position and $\boldsymbol{\Sigma} \in \mathbb{R}^{3\times3}$ is the covariance matrix, parameterized as:

$$\boldsymbol{\Sigma} = \mathbf{RSS}^T\mathbf{R}^T \tag{2}$$

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:

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

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:

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

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

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):

$$r = \sqrt{x^2 + y^2 + z^2} \tag{6}$$

$$\theta = \arctan(y/x) \in [0, 2\pi) \tag{7}$$

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

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

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 → heliocentric Cartesian
- Scale: Estimated from absolute magnitude
- Color: BP-RP photometry → 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 $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$

$\mathbf{R} \leftarrow \text{QuatToRotMatrix}(\mathbf{q})$ ▷ $[N, 3, 3]$
$\mathbf{\Sigma}_{3D} \leftarrow \mathbf{R} \cdot \text{diag}(\mathbf{s}) \cdot \text{diag}(\mathbf{s})^T \cdot \mathbf{R}^T$
$\mathbf{p}_{screen}, \mathbf{p}_{view} \leftarrow \text{Project}(\mathbf{p}, \text{camera})$
$\mathbf{\Sigma}_{2D} \leftarrow \mathbf{J} \cdot \mathbf{\Sigma}_{3D} \cdot \mathbf{J}^T$ ▷ Jacobian projection
$\mathbf{C} \leftarrow \text{InvertToConic}(\mathbf{\Sigma}_{2D})$ ▷ $[a, b, c]$
for each pixel (u, v) **do**
  $\mathbf{d} \leftarrow (u, v) - \mathbf{p}_{screen}$
  $dist^2 \leftarrow a \cdot d_x^2 + 2b \cdot d_x d_y + c \cdot d_y^2$
  $\mathbf{w} \leftarrow \exp(-0.5 \cdot dist^2) \cdot \alpha$
end for
$\mathbf{I} \leftarrow \text{AlphaBlend}(\mathbf{w}, \mathbf{c}, \text{depths})$

4.2.3 Camera Configuration for Astronomical Scales

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

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

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

4.2.4 Loss Function

Combined L1 and D-SSIM loss:

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

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:

$$M = E - e \sin E \tag{11}$$

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: $d\Omega = -0.18047°$/century
- Earth apsidal precession: $d\omega = +0.32327°$/century

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%

6 Streaming Architecture

6.1 End-to-End Latency

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**

7 Implementation

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

- `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)

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

9 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.

Future work includes multi-user sessions, extended datasets (full Gaia DR3, exoplanet systems), and VR support via WebXR.

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.