OpenCosmo Visualization: In-Browser Exploration of Cosmological Particle Data

Idunnuoluwa A. Adeniji
University of Illinois Chicago and Argonne National Laboratory
Chicago / Lemont, IL, USA
iaden@uic.edu

Joseph A. Insley
Argonne Leadership Computing Facility Argonne National Laboratory
Lemont, IL, USA
insley@anl.gov

Michael E. Papka
University of Illinois Chicago and Argonne National Laboratory
Chicago / Lemont, IL, USA
papka@uic.edu

***Abstract*—Querying a cosmological simulation and seeing it are usually separated by a dedicated tool, a preprocessing step, and a broken workflow. The OpenCosmo Visualization Application (OVA) closes that gap inside the OpenCosmo Portal: a completed query opens directly into an interactive 3D view of the queried particles, in the same browser tab, with no export, installation, or preprocessing. Built on WebGL2 and Babylon.js, OVA adds per-type controls, dynamic filtering by scalar range, a 3D clipping plane, screen-space selection statistics, and one-click immersive WebXR on top of point rendering. Even on integrated laptop graphics, OVA ingests and renders 5M points of a HACC halo end-to-end in about 6.7 seconds, sustains 114.8 FPS, and records zero blocking main-thread tasks.**

***Index Terms*—Apache Parquet, Cosmological simulation, GPU rendering, HACC, Point cloud visualization, Scientific visualization, WebAssembly, Web-based visualization, WebXR**

I. Introduction

OpenCosmo Portal [1], [2] gives researchers query access to flagship HACC [3] cosmological simulations, returning millions of particles as Apache Parquet [4]. Those results have arrived as numbers: inspecting a halo meant downloading it, installing a viewer or writing a Python pipeline, and rebuilding that workflow every query. OVA answers this at the application level: an interactive 3D viewer embedded in the portal so querying and seeing occupy one browser session (Fig. 1), rendered on the client GPU via Babylon.js [5] over WebGL2. Its contributions are (1) closing the query-to-visualization loop inside a portal; (2) zero-preprocessing client-side ingestion of the portal's native Parquet via WebAssembly and Apache Arrow, with a `Transferable` zero-copy handoff to the main thread; (3) a GPU point-rendering pipeline with per-type controls, dynamic filtering, GPU 3D clipping, spatial-selection statistics, and immersive WebXR in one interface; and (4) a demonstration that shifting visualization to the client removes reliance on shared HPC resources. A preliminary version appeared at IEEE eScience 2025 [6].

II. Related Work

Firefly [7], the closest prior work, is a browser-based 3D particle viewer on Three.js. Its limitation for portal workflows is preprocessing: source data must first be converted to JSON or a proprietary binary, breaking the continuous in-portal loop that OVA preserves. OVA ingests the portal's native Parquet directly via WebAssembly and derives scalars such as velocity magnitude in a Web Worker at load time. ParaViewWeb [8] exposes the full ParaView/VTK pipeline through the browser, but separates a server-side ParaView process from a lightweight client, requiring a visualization server, a reverse proxy, and a session launcher. OVA targets an earlier point in the workflow: the first look at a queried subset, without server-side visualization infrastructure. Desktop tools such as yt [9], ParaView, and VisIt remain the standard for in-depth analysis of full HACC volumes; OVA is a zero-installation entry point after a portal query, not a replacement.

III. System Architecture and Implementation

Ingestion. OVA reads the portal's native Parquet directly in the browser, with *zero preprocessing* and no server-side rendering. All work runs *client-side* inside a *Web Worker* so the interface stays responsive: `parquet-wasm` decodes the buffer into an Apache Arrow table, and derived scalars such as velocity magnitude are computed alongside without touching source files. Data is held as flat columnar `Float32Array` buffers and handed to the main thread through the `Transferable` interface, which moves ownership of the underlying buffers in constant time rather than deep-copying.

Rendering. WebGL2 with *Babylon.js* draws the dataset as a single mesh through a custom GLSL shader pair: a Gaussian point sprite with additive blending and Reinhard tonemapping, so dense regions accumulate brightness while voids stay dark. Scalar recoloring, opacity, and clipping execute in-shader without re-uploading geometry.

IV. Workflow Integration and Interactive Demo

User experience. A researcher runs a *Visualize Halo* job in the portal, and on completion *Open in Visualizer* hands the queried Parquet files to OVA in the same tab. A brief client capability check reports GPU acceleration and WebXR availability in plain language, then the halo appears as a rendered 3D point cloud within seconds (Fig. 1).

Interactivity. Users navigate through orbit, pan, zoom, and preset views; filter by scalar range and point cap; adjust scalar, colormap, size, and opacity independently per particle type; cut cross-sections with a 3D clipping plane; extract per-type statistics from a screen-space box selection; and enter

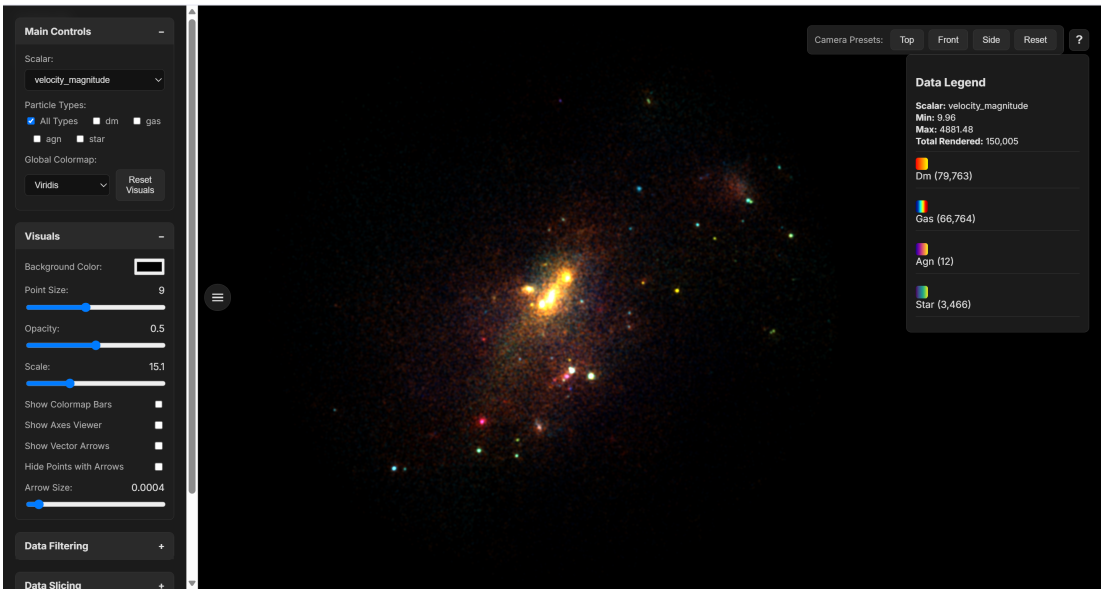


Fig. 1: Overview of the OVA interface rendering a HACC halo returned by a portal query. The main view shows GPU-based point rendering of a cosmological structure; the left panel exposes controls for colormap, filtering, slicing, and camera presets; and the right panel gives a live data legend with per-type scalar range and rendered-point counts. These components are tightly coupled within a single browser tab, letting users explore, filter, and analyze the queried dataset without leaving the portal session.

immersive WebXR in one click with the full visualization state preserved.

V. Performance and Results

Development and benchmarking used a consumer laptop with 16 GB of memory and 16 hardware threads under Google Chrome, with rendering on the integrated AMD Radeon 780M through ANGLE over Direct3D 11. The test dataset was a 5M-particle HACC halo in Parquet format. Ingestion completed end-to-end in about 6.7 s, averaged over three runs; the `Transferable` path avoided the structured-clone copy that otherwise dominates loading, and a `PerformanceObserver`-based long-task monitor recorded zero blocking main-thread tasks during ingestion. At the full 5M-particle count, WebGL2 rendering sustained 114.8 FPS on AC power, on this integrated GPU.

VI. Conclusion and Future Work

OVA shows that a simulation portal can incorporate its own visualization layer: queries become interactive 3D exploration in one session, on hardware users already own, with no external pipelines. The current backend is WebGL2 with Babylon.js. Ongoing work focuses on tighter portal embedding, addition of level-of-detail rendering, harnessing WebGPU technologies for better performance, and richer interaction inside immersive sessions.

References

[1] P. R. Wells, M. Buelhmann, P. Larsen, W. M. Hicks, M. Dhillon, I. A. Adeniji, K. Heitmann, S. Habib, B. Côté, T. Uram, G. McFarland, A. Hearin, E. Shinabro, and M. E. Papka, "Opencosmo: Community portal and analysis framework for flagship cosmological simulations," 2026. [Online]. Available: https://arxiv.org/abs/2607.16059

[2] Argonne National Laboratory, "OpenCosmo Portal," https://cosmoexplorer.alcf.anl.gov/.

[3] S. Habib, A. Pope, H. Finkel, N. Frontiere, K. Heitmann, D. Daniel, P. Fasel, V. Morozov, G. Zagaris, T. Peterka, V. Vishwanath, Z. Lukić, S. Sehrish, and W. k. Liao, "HACC: Simulating sky surveys on state-of-the-art supercomputing architectures," *New Astronomy*, vol. 42, pp. 49–65, 2016, https://doi.org/10.1016/j.newast.2015.06.003.

[4] D. Vohra, "Apache parquet," in *Practical Hadoop Ecosystem: A Definitive Guide to Hadoop-Related Frameworks and Tools*. Berkeley, CA, USA: Apress, 2016, pp. 325–335, https://doi.org/10.1007/978-1-4842-2199-0_8.

[5] Babylon.js, "Babylon.js Documentation," https://doc.babylonjs.com/, [Online].

[6] I. A. Adeniji, J. Insley, M. Han, J. Y. Kim Knowles, M. E. Papka, V. A. Mateevitsi, and S. Rizzi, "Interactive exploration of hacc cosmology data using webxr," in *2025 IEEE International Conference on eScience (eScience)*, 2025, https://doi.org/10.1109/eScience65000.2025.00044.

[7] A. B. Gurvich and A. M. Geller, "Firefly: A browser-based interactive 3D data visualization tool for millions of data points," *The Astrophysical Journal Supplement Series*, vol. 265, no. 2, p. 38, 2023, https://doi.org/10.3847/1538-4365/acb59f.

[8] S. Jourdain, U. Ayachit, and B. Geveci, "ParaViewWeb: A web framework for 3D visualization and data processing," *International Journal of Computer Information Systems and Industrial Management Applications*, vol. 3, pp. 870–877, 2011.

[9] M. J. Turk, B. D. Smith, J. S. Oishi, S. Skory, S. W. Skillman, T. Abel, and M. L. Norman, "yt: A multi-code analysis toolkit for astrophysical simulation data," *The Astrophysical Journal Supplement Series*, vol. 192, no. 1, p. 9, 2011, https://doi.org/10.1088/0067-0049/192/1/9.