A WebGPU Processing Pipeline for Particle–Magnet Feature-Space Visualization

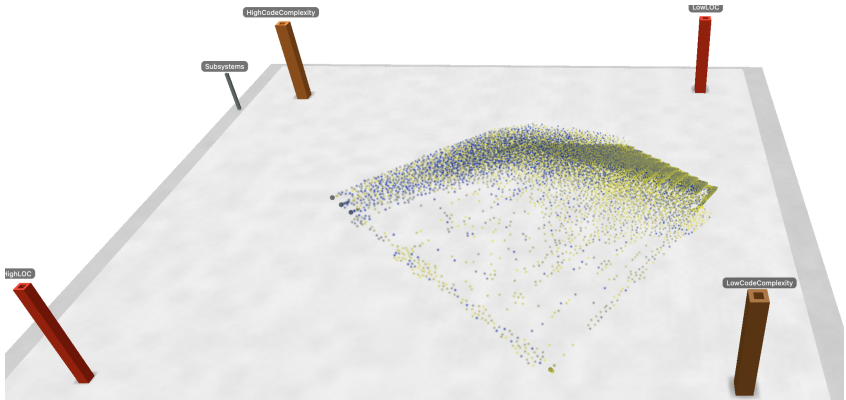
Jürgen Döllner **Willy Scheibel** **Jose Carvajal**

Hasso Plattner Institute / University of Potsdam, Germany

Web3D '26 13–15 October 2026 Doha, Qatar

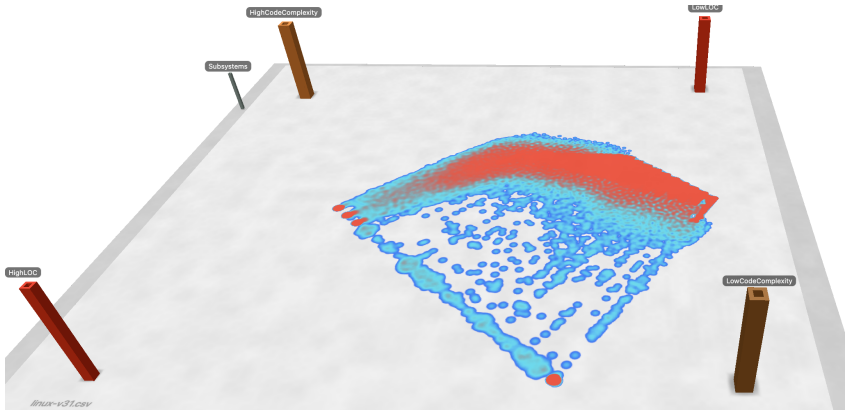
doi:10.1145/3822516.3837796

Motivation: Scalable Interactive Feature-Space Visualization



Example: Multivariate dataset derived from Linux kernel source code repository, approx. 96k objects represented by particles; analytical magnets used to view code complexity distribution.

Motivation: Scalable Interactive Feature-Space Visualization



Example: Particle density map. General characteristics of particle visualizations include non-uniform spatial distribution, overplotting, visual clutter, scale dependence, dynamic redistribution.

Overview: Particle–Magnet Visualization

Visualization Approach

- Each *multivariate object* becomes a feature vector with d dimensions.
- Each *feature vector* is represented by a particle.
- Each *particle* is represented by a graphical object, e.g., billboard disc or splat.
- A *magnet* evaluates an interpretable criterion based on one or more features for each feature vector.
- Particle-magnet *affinities* guide placement, i.e., the particle is positioned at a weighted barycentric target.
- Interactively adjusting magnet configurations and positions supports human-in-the-loop exploration.

Particle–magnet visualization preserves object-level identity and enables individual particle picking and inspection. Magnets explicitly express analysis criteria.

Motivation: Scalable Interactive Feature-Space Visualization

Observations

- Particle–magnet visualization requires recomputing particle states when magnets change.
- Affinity evaluation, filtering, positioning, visual mapping and drawing are coupled.
- CPU processing and repeated CPU–GPU transfers threaten interactive response.

Our boundary conditions:

- The number of *exploration-relevant* feature dimensions is relatively low $(10 - 30)$.
- Target particle counts reach up to 10^7.

Scalable end-to-end GPU processing with dependency-aware updates?

Research Gap and Contribution

Challenge
Large particle sets require expensive per-particle recomputation, GPU uploads and dense rendering.

Question
How can interactive magnet changes update only what actually depends on them?

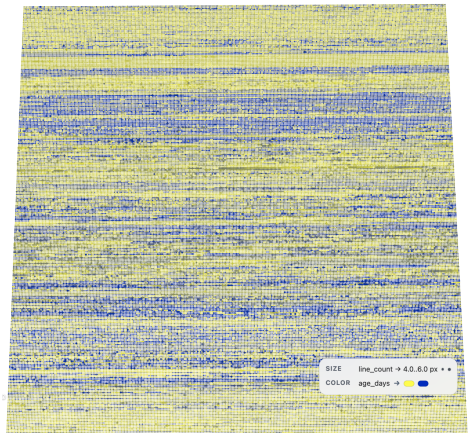
Contribution
A **GPU-resident, dependency-aware WebGPU pipeline** with reusable intermediates.

Evaluation
Synthetic size variants of a real Linux kernel dataset, up to 1.5 million particles; compared with JavaScript and worker baselines.

This paper contributes a WebGPU-based processing and rendering pipeline for particle–magnet visualization, rather than a new visualization metaphor.

Case Study Dataset: Linux Kernel Source Code Repository

- October 2026 snapshot of `torvalds/linux` (GitHub).
- approx. **96,000** source files; approx. **43 million** parsed lines.
- Each file: **19** normalized numerical feature dimensions.
- Structural: LOC, complexity, nesting, architectural level, import/export density.
- Evolutionary: age, recency, growth, commits, 90/180-day churn.
- Social: contributor count and diversity; path and name retained as metadata.



Plain particle dump onto a regular grid in CSV order; no magnets are active.

Normalized Feature Space

Let $\mathcal{O} = \{o_1, \ldots, o_n\}$ be objects described by d interpretable, numerical attributes.
Each object is mapped to a feature vector $\tilde{\mathbf{x}}$ of the d-dimensional hypercube $\mathcal{F}$:

$$\tilde{\mathbf{x}}(o) = (\tilde{x}_1(o), \ldots, \tilde{x}_d(o)) \in \mathcal{F}, \qquad \mathcal{F} = [0,1]^d.$$

- Preprocessing maps $\mathcal{O}$ to F by analyzing per-dimension value distributions, applying percentile-based clipping, and normalizing values to $[0, 1]$.
- Categorical attributes can be mapped to visual variables (e.g., color, size) or to the z-coordinate in 3D layouts.
- GPU **feature store** F: fixed-stride 32-bit floats, uploaded once per session.
- Original attributes and metadata stay CPU-accessible for inspection.

Magnets

Each magnet M_j evaluates a criterion $w_j(o)$ for a given particle o:

$$w_j(o) \in [0, 1]$$

For n particles and m magnets, the GPU stores the $n \times m$ matrix

$$\boxed{W_{ij} = w_j(o_i)}$$

- Single-feature magnet: directly maps a feature dimension
- min magnet: takes the minimum of selected feature dimensions (i.e., logical AND)
- max magnet: takes the maximum of selected feature dimensions (i.e., logical OR)
- length magnet: computes the norm of a feature (sub)vector
- Other magnet types include profile, similarity, variance, inverse magnets

Particle Placement

For placement, we consider a magnet M_j, its location $\mathbf{s}_j \in \mathbb{R}^2$ and its signed strength α_j:

$$\hat{w}_j(o_i) = \alpha_j W_{ij}.$$

$$\mathbf{t}_j(o_i) = \begin{cases} \mathbf{s}_j, & \hat{w}_j(o_i) \geq 0, \\ \mathbf{s}_j + |\hat{w}_j(o_i)|\mathbf{d}_j, & \hat{w}_j(o_i) < 0, \end{cases} \qquad \mathbf{d}_j = \frac{-\mathbf{s}_j}{\|\mathbf{s}_j\|}.$$

- $\alpha_j > 0$: Attracts particles toward the magnet; the more strongly an object satisfies the criterion, the stronger the attraction.
- $\alpha_j < 0$: Repels particles by shifting their targets toward or beyond the reference-plane center.
- $\alpha_j = 0$: Disables the magnet's influence on placement.
- $\alpha_j \to \infty$: Pulls particles with nonzero affinity toward the magnet ("black hole").

Particle Placement

For a particle o_i, we compute the barycentric particle position q^* in the reference plane $\mathcal{P}$ based on magnet forces:

$$\mathbf{q}^*(o_i) = \Pi_{\mathcal{P}} \left(\frac{\sum_{j=1}^{m} |\hat{w}_j(o_i)| \, \mathbf{t}_j(o_i)}{\sum_{j=1}^{m} |\hat{w}_j(o_i)| + \beta} \right), \qquad \mathcal{P} = [-1, 1]^2$$

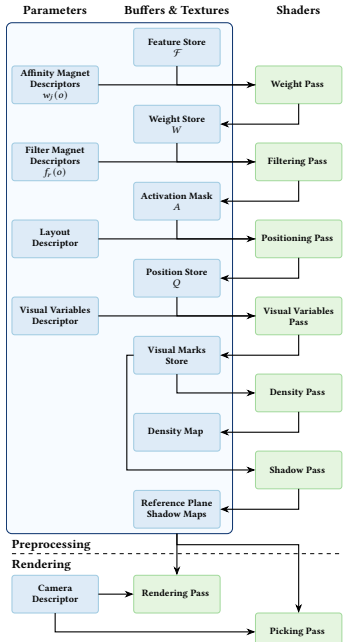
- $\beta > 0$ biases positions toward the origin; $\Pi_{\mathcal{P}}$ clamps to the reference plane.
- No nonzero influence $\Rightarrow$ target at the origin.
- Position buffer: $Q_i = (q_x^*(o_i), q_y^*(o_i), z(o_i))$.
- z supports planar, layered, height-based and 3D scatterplot layouts.

Moving a magnet changes Q, but existing affinities W can be reused.

WebGPU Pipeline

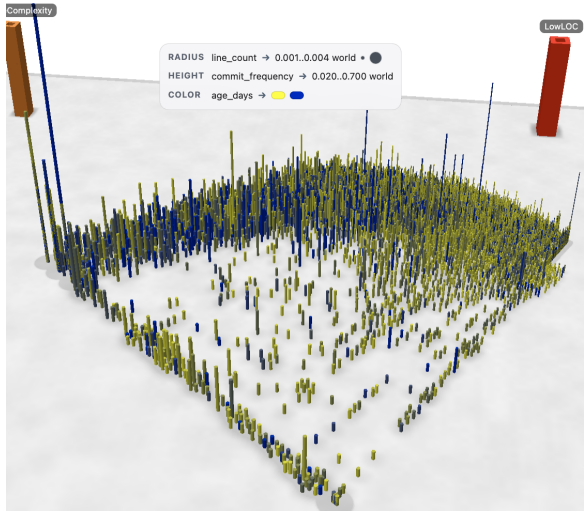
- F: normalized feature store
- W: particle–magnet affinities
- A: particle activation mask (filtering)
- Q: particle positions
- Visual-attribute buffers, density maps, and shadow maps

All per-particle data processing and rendering are performed entirely on the GPU.



Particle Rendering

- Particles are rendered by glyphs. Examples:
  - 2D discs
  - 2D Gaussian splats
  - 3D cylinders
- Visual mappings control glyph appearance, e.g., size, color and opacity.
- Optional screen-space density adjusts opacity or splat falloff in overlapping regions.
- The renderer consumes GPU-resident Q, A and visual buffers directly.



Evaluation: A Software Mining Use Case

Input workload
Linux kernel dataset expanded or truncated to 25–1600%: **23,633 to 1,512,480 particles**.

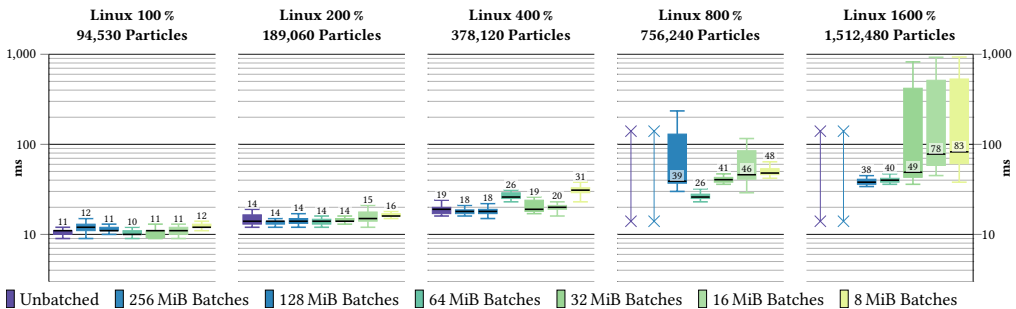
Four implementations
Main-thread JavaScript; JavaScript Web Worker; unbatched WebGPU; batched WebGPU (8–256 MiB).

Environment
Firefox 153; Intel Core i9-10900K; 64 GB RAM; NVIDIA Quadro RTX 5000 (16 GiB); driver 595.84.

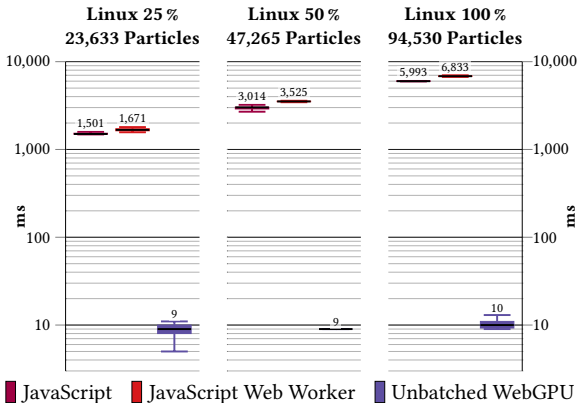
Measurements
50 measurements after 5 warm-up iterations, using GPU timestamp queries or CPU wall-clock timing as appropriate.

Measured core runtime starts at the weight pass; this is not an end-to-end frame-rate benchmark.

Results


Linux 100 %
94,530 Particles
Linux 200 %
189,060 Particles
Linux 400 %
378,120 Particles
Linux 800 %
756,240 Particles
Linux 1600 %
1,512,480 Particles
1,000
100
10
ms
11 12 11 10 11 11 12
14 14 14 14 14 15 16
19 18 18 26 19 20 31
39 26 41 46 48
38 40 49 78 83
Unbatched
256 MiB Batches
128 MiB Batches
64 MiB Batches
32 MiB Batches
16 MiB Batches
8 MiB Batches

Results



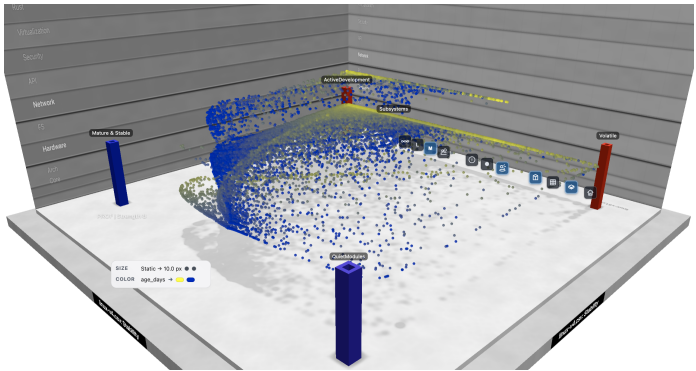
- Original 94,530-particle dataset: roughly **5,993 ms** JavaScript and **10 ms** unbatched WebGPU medians.
- Up to approximately **600**× lower recomputation time for the full original dataset.
- Web Workers keep the main thread responsive, but increase median recomputation time by around **15%**.

Conclusions

- The **WebGPU pipeline** performs all performance-critical computations of particle–magnet visualization.
- **Dependency-aware invalidation** reuses unaffected intermediate results during interactive updates.
- The architecture supports **different visualizations**, including 2D maps and 3D information landscapes.
- **Performance evaluation** demonstrates scalability beyond one million particles.
- **Further application domains** include tree inventories, 3D city models, and ERP/business intelligence analytics.

- **Future work:** temporal feature spaces, adaptive rendering of dense particle distributions, and advanced glyph logic.

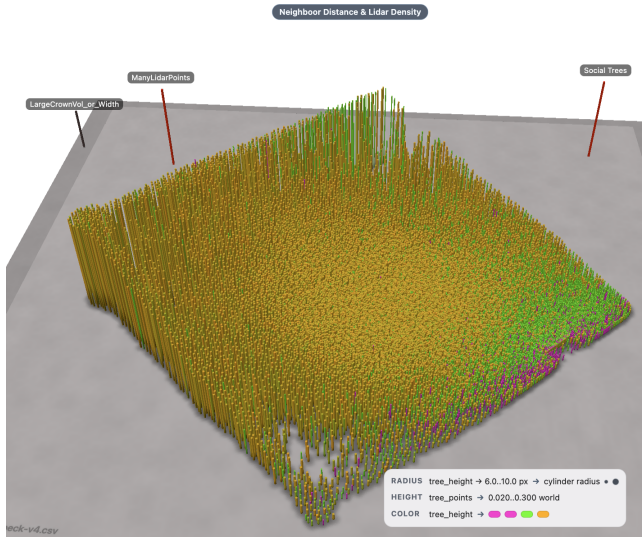
Thank You!



Questions?

Jürgen Döllner, doellner@uni-potsdam.de

Appendix: Tree Inventory User Case



Appendix: City Model User Case

