Portable GPU-Accelerated Colour-Space Visualization

Kyle Lukaszek†
klukasze@uoguelph.ca
University of Guelph
Guelph, Ontario, Canada

Denis Nikitenko
dnikiten@uoguelph.ca
University of Guelph
Guelph, Ontario, Canada

David R. Flatla
dflatla@uoguelph.ca
University of Guelph
Guelph, Ontario, Canada

Abstract

Interactive colour-space visualization tools are still fragmented between legacy desktop plugins and modern web demos that prioritize visual polish over data completeness. In practice, this leaves researchers without a robust, inspectable, full-resolution workflow for exploring colour-volume structure and interior occupancy. This paper presents a WebGPU technical demo that computes colour-space point clouds on the GPU, caches generated point buffers, and renders either native 1-pixel points or instanced circle splats for denser displays. The system supports full RGB lattice generation up to 256^3 samples, image-driven unique-colour sets, multiple perceptual and cylindrical colour spaces, and interactive clip bounds for interior investigation. We describe the pipeline design, implementation tradeoffs, and how this approach addresses common limits in current tools.

CCS Concepts

• **Human-centered computing** → **Visualization systems and tools**; • **Computing methodologies** → *Graphics systems and interfaces.*

Keywords

colour spaces, WebGPU, point clouds, compute shaders, scientific visualization

ACM Reference Format:
Kyle Lukaszek†, Denis Nikitenko, and David R. Flatla. 2026. Portable GPU-Accelerated Colour-Space Visualization. In *Proceedings of Demo at Graphics Interface 2026 (GI '26).* ACM, New York, NY, USA, 2 pages.

1 Introduction

Colour-space visualization remains essential in graphics and imaging research, but tooling has not kept pace with modern GPU capabilities. Legacy tools persist because they expose point-wise colour distributions, while newer web demos often prioritize surface aesthetics over dense interior analysis. Our goal was a browser-native, point-first system with practical interactivity at high sample counts.

2 Tooling Gap

The ImageJ *Color Inspector 3D* plugin remains one of the most complete accessible tools for researchers, but its public documentation shows a development history rooted in 2004–2007 and a Java plugin workflow that reflects that era [1]. The ImageJ project pages also characterize this plugin line as not hardware accelerated [2]. In practice, this translates to constrained scale and limited fit with contemporary browser-based research demos.

Demo at Graphics Interface 2026 (GI '26), Waterloo, ON, Canada
2026.

At the same time, modern web colour-space visualizations such as *hueplot* [3] are visually strong but primarily mesh-based, which emphasizes outer geometry and reduces direct inspection of interior occupancy patterns. Point-based Three.js examples exist, but they typically rely on reduced sampling densities for interactivity, and point rendering itself is often subject to implementation caps and platform constraints [6].

These two ends of the spectrum expose a clear gap: researchers need an interactive, high-density, point-native colour-space tool that runs on current GPU APIs and supports interior analysis tasks.

3 WebGPU Implementation

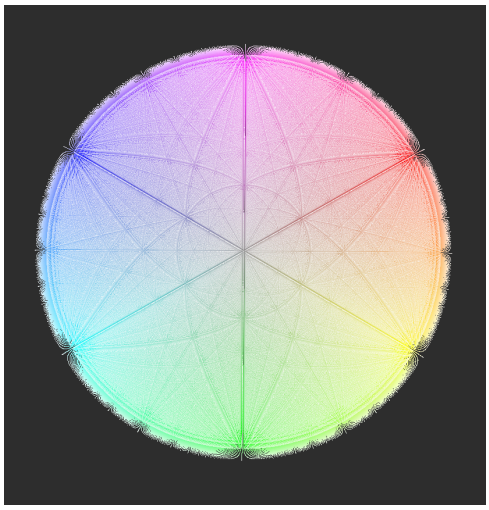


Figure 1: sRGB gamut rendered in HSL at a resolution of 256^3

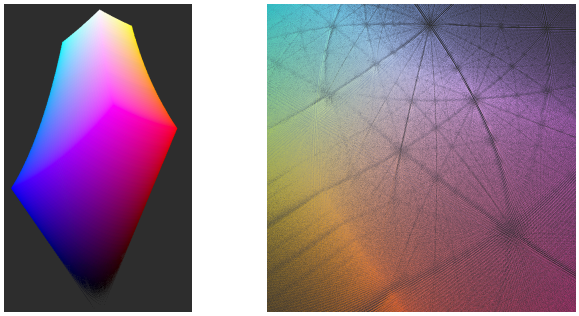
3.1 Data Generation

The demo supports two point sources:

(1) A regular RGB lattice with adjustable grid size (up to 256^3 points) as seen in Figure 1.
(2) Unique sRGB colours extracted from uploaded images (including PPM parsing).

For each point, a compute shader converts sRGB to a selected target space (RGB, CIELAB, CIELUV, HSL, HSV, Oklab, Oklch) and writes world-space position plus display colour to a storage buffer. In grid mode, this allows full-resolution colour-volume generation directly on the GPU. In image mode, it enables direct analysis of empirical colour distributions.

Dispatch uses a 256-thread workgroup and dynamically maps large workloads across 3D dispatch dimensions when 1D workgroup counts exceed device limits. This preserves correctness for



(a) Oklab global view (b) Oklab interior zoom

Figure 2: Zoomed-out and zoomed-in Oklab views for internal structure inspection.

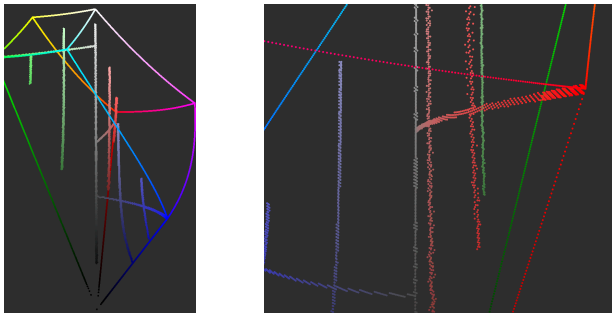
high point counts while remaining portable across hardware. GPU-side caching avoids recomputation during interaction, enabling stable navigation at high point counts.

3.2 Rendering Modes

The renderer exposes two draw paths: native WebGPU `point-list` rendering (1 px primitive footprint), and instanced circle splats (two triangles per point) for larger projected marks. The first path is exact and inexpensive; the second improves perceptual continuity in sparse or zoomed views, with interactive switching through a point-size parameter.

4 Interaction for Interior Investigation

A key requirement was interior exploration rather than boundary-only viewing. The demo includes an interactive axis-aligned clip-bounds widget with per-axis min/max controls and draggable face handles. Users can progressively peel the volume and inspect internal structures that are difficult to observe in mesh-only gamut surfaces.



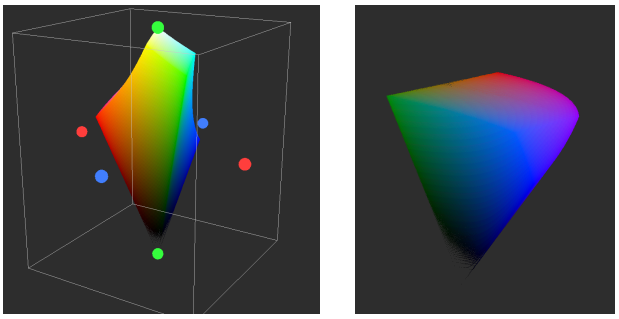
(a) Image-mode point cloud (b) Interior of image with 10px points

Figure 3: Colour gamut used in a recent HCI study rendered in Oklab, shown at overview and detail scales.

Figure 2(a) provides the full-gamut context, while Figure 2(b) shows what becomes visible after zooming into the same space. In the interior view, individual samples resolve into spoke-like radial fans, seam-aligned ridges, and repeated star-node intersections rather than a smooth mesh proxy. This is the primary analysis task the tool supports: move from a global colour-volume view to localized, point-level structure inspection without changing data representation.

The clip interface and image-driven mode support the same investigation workflow on both synthetic full-lattice data and empirical colour sets from input images.

Figure 3 shows image mode, where unique image colours form sparse vertical trajectories and curved boundary traces in the same analytic space. Figure 4(a) shows the clip-bounds workflow explicitly: the gamut sits inside a wireframe volume with per-axis min/max face handles (red, green, blue) used to peel and isolate interior regions. Figure 4(b) shows the corresponding clipped Oklab volume after peeling.



(a) Clip-bounds controls (b) Clipped Oklab view

Figure 4: Clip-bounds interaction and resulting clipped volume used for structure inspection.

5 Discussion and Limitations

The WebGPU implementation makes full-resolution, browser-native point-cloud colour-space inspection practical, but limits remain: memory usage scales linearly with point count, and large splat sizes become fill-rate bound at high density, so the 1-pixel point path is the practical interactive mode for very large datasets. We also prototyped similar workflows in Makie and PyVista [4, 5]; interactivity dropped sharply as resolution increased, reinforcing the compute-buffer caching and dual render-path design used here.

6 Conclusion

This technical demo contributes a practical WebGPU compute-splatter pipeline for interactive, high-density colour-space analysis. By combining GPU-side full-resolution point generation, explicit caching, and dual rendering modes with clip-based interior exploration, it provides a modern alternative to legacy colour-space tools and a stronger foundation for research-facing visualization workflows.

References

[1] Kai Uwe Barthel. *3D Color Inspector/Color Histogram (ImageJ plugin documentation).* https://imagej.net/ij/plugins/color-inspector.html. Accessed April 29, 2026.

[2] ImageJ. *Color Inspector 3D (archived plugin page).* https://imagej.net/imagej-wiki-static/Color_Inspector_3D. Accessed April 29, 2026.

[3] Ardo van Rangelrooij. *Hueplot.* https://hueplot.ardov.me/. Accessed April 29, 2026.

[4] Simon Danisch and Julius Krumbiegel. *Makie.jl: Flexible high-performance data visualization for Julia.* Journal of Open Source Software, 6(65):3349, 2021. https://doi.org/10.21105/joss.03349.

[5] Bane Sullivan and Alexander Kaszynski. *PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK).* Journal of Open Source Software, 4(37):1450, 2019. https://doi.org/10.21105/joss.01450.

[6] Three.js authors. *PointsMaterial documentation.* https://threejs.org/docs/pages/PointsMaterial.html. Accessed April 29, 2026.