

Recentering OpenSky Traffic: Arbitrary-Center, Multi-Projection Visualization on a Single GPU

Wataru Shida

Independent Researcher, Tokyo, Japan — mapshift.dev@gmail.com — <https://mapshift.dev>

Abstract

A projection preserves some quantities at the cost of others, so the map that is “correct” depends on the question being asked. Aviation asks about distance and bearing from a point, while everyday web maps preserve local angles in a fixed global frame. This paper reports the Arc Algo Engine (AAE), a single-GPU engine in which both the center and the projection are parameters: a spherical rotation places any point at the projection origin, and an inverse look-up table maps each output pixel back to source coordinates. Drawing the same nine routes under three projections shows that the line which comes out straight — a great circle through the center, a rhumb line, or neither — is different in each. The contribution is that recentering becomes an interactive operation rather than a configuration, not a new projection.

Keywords: ADS-B visualization; OpenSky Network API; map projection; GPU-accelerated visualization; azimuthal equidistant

1. Introduction

Any map of a sphere must give something up. Gauss’s *Theorema Egregium* rules out an isometry between the sphere and the plane, so a projection preserves some quantities at the cost of others. Which map is “correct” therefore depends on what the reader needs to be true.

Aviation needs distance and bearing from a point. Range from a receiver, distance to an alternate, the coverage of a ground station — these are questions asked *from somewhere*. A projection that answers them is centered on that place.

Web Mercator answers a different question. It preserves angle and tiles cleanly, which is why most interactive web maps are drawn in it. Panning moves the view, not the projection: its aspect is fixed, and the tile pyramid, the $z/x/y$ addressing and the client libraries all assume it. Choosing a projection — let alone a center — is not an operation the stack provides.

The Arc Algo Engine (AAE) is a geospatial visualization engine that draws base maps and live observations — aircraft, ships, ground stations, satellites, cyber events — under any of twenty projections, centered on any point. It makes both choices first-class: a spherical rotation places any point at the projection origin, and an inverse look-up table maps each output pixel back to source coordinates. I show what changes when the center moves, and what it costs.

2. Method

2.1 Why forward rasterization fails

Rotating the source geometry and rasterizing it is the conventional approach — PROJ exposes oblique aspects this way — and it is the one I tried first. It did not survive an interactive center, and the difficulty was structural.

An equirectangular map has edges: the antimeridian at the sides, the poles at top and bottom. When

the center moves, those edges move with respect to the data. A closed ring starts to cross the seam, the segment joining -179° to $+179^\circ$ is drawn straight across the map, and a filled polygon inverts land and sea because the fill rule no longer sees a closed boundary. Each case can be special-cased, but the case analysis depends on the center, so it has to be redone for every center. Six successive attempts at a clean forward rasterizer failed this way before the approach was abandoned.

2.2 Moving the edge cases to a fixed place

The engine does not remove the seam and pole handling. It moves it somewhere the center cannot reach.

Geometry is baked offline into an equirectangular sheet whose edges are always $\pm 180^\circ$ and $\pm 90^\circ$, independent of any viewing center. Rings are split at the antimeridian and the polar boundary is repaired during preprocessing; a ring that still crosses the seam does not fit in one width, so the fill runs on a canvas three times as wide and only the middle strip is kept, which leaves every such ring whole in at least one copy. This is done once: the viewer reads the result from a verified cache, so a map is generated offline, with no tile server or network connection; only live observations need one.

The projection is then applied by inverse mapping. For a given center, projection and output size, a look-up table maps each output pixel back to a source coordinate: the pixel is unprojected, rotated back by R^T (the inverse rotation), and converted to a source index, with the horizontal periodicity expressed as a modulo. Changing the center rebuilds R and the table, and nothing else.

Points and dynamic lines — observations, orbit tracks, graticules, geodesics — take the other direction: they are forward-projected with the same R and projection and composited onto the sampled frame. The two paths share no buffer.

2.3 Two independent axes

Because the center lives in a rotation matrix and the projection is a plane mapping applied afterwards, the two are independent: a projection class never sees a center, only R , so adding one means writing a forward and an inverse for the plane; twenty are registered. The same separation applies on the input side: the engine fetches observations itself — aircraft directly from the OpenSky Network API — and normalizes them, whatever the transport, into one record type before any projection is involved, so N sources (each an axis) and M projections meet as $N + M$ rather than $N \times M$ (Figure 2(b)); a new axis is one more input class, not a new pipeline.

2.4 General-purpose GPU computation

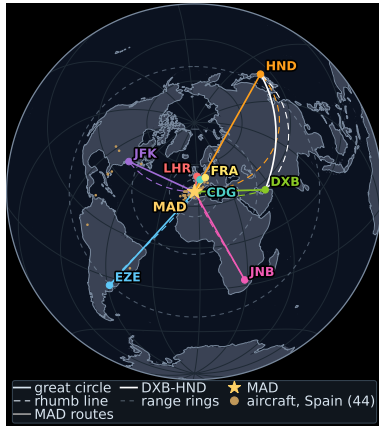
Rendering does not go through a graphics pipeline. No shader, texture unit or framebuffer is used: a scan of the codebase finds no import of any graphics API. The GPU work runs through CuPy, with custom CUDA kernels for the fill, the sampling and the compositing, and with wrap and clamp written explicitly rather than delegated to a sampler.

Three things follow. The engine is not constrained to what a browser can do — its security layer binds a listening UDP socket to receive pushed events, which a sandboxed page cannot. And although it is written in Python, observations are projected as array operations on the GPU, so one frame can place 100,000 points in 3.1 ms at $1,920 \times 1,080$ (median of 20 frames, GPU copies included); what limits the map is legibility, which is why Figure 1(a) draws 44 of the 8,514 aircraft in its snapshot. And because sampling is not a fixed-function unit, categorical layers can be forced to nearest-neighbor, so that reprojection introduces no color absent from the source — the case Jenny et al. [1] leave open. Their per-fragment inverse mapping in WebGL is the origin of this approach. I moved it out of the browser for two reasons: WebGL did not give me the performance I needed with many layers and

observations at once, and building every layer as a web application was more than one person could maintain. The difference is the implementation layer, not the idea.

3. Results

3.1 What changes when the projection changes



(a) Azimuthal equidistant, centered on Madrid Barajas



(b) Mercator, central meridian at Madrid's longitude

Figure 1. Nine routes and the 44 Spanish-registered aircraft of one OpenSky snapshot (23:27 UTC, 23 September 2026). Solid: great circles; dashed: rhumb lines; dotted: 500 to 10,000 km from Madrid. In (a) the routes from the center are straight and the rings true circles; in (b) the rhumb lines are straight and the rings distorted.

Figure 1(a) shows nine routes between airport pairs and the aircraft of one snapshot, drawn on an azimuthal equidistant projection centered on Madrid Barajas. Great circles through the center appear as straight lines radiating from it, and the range rings are true circles, so the distance of each aircraft from Madrid can be read off the rings.

Figure 1(b) draws the same on Mercator centered on Madrid's longitude: the rhumb lines are straight and the great circles bow poleward. A cylindrical projection has no latitude of center, so the rings distort: they still mark equal distance from Madrid, but no single scale measures it. Neither figure is wrong.

3.2 The center is not a general-purpose fix

Eight of the nine routes begin at the center, so their appearance in Figure 1(a) is not informative on its own: a great circle through the center of an azimuthal projection is straight by construction. The ninth route, Dubai to Tokyo, does not pass through Madrid. In Figure 1(a) it is a curve, departing from its chord by 1,050 km, or 11 % of the 9,240 km chord.

This is the limit of the azimuthal choice: distance and bearing are correct from one point, not between points. A projection that renders every great circle straight exists — the gnomonic — but it cannot show a hemisphere; this implementation caps the visible radius at 60°, and three of the eight destinations fall outside that cap from Madrid. Choosing a projection is choosing which question to answer well.

3.3 A third axis

On Equal Earth (Figure 2(a)) neither curve is straight; what it preserves is area.

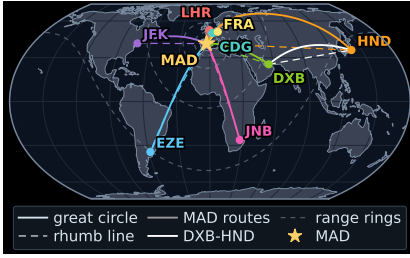
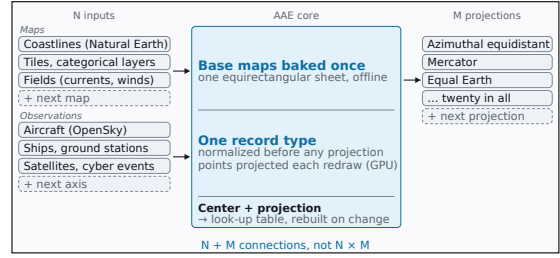
(a) Equal Earth, centered at 0° (b) N inputs, one core, M projections

Figure 2. (a) The same routes on Equal Earth, centered at 0° : neither curve is straight. (b) Maps are baked once; observations are normalized to one record type; every projection is one class behind the same interface, and a change of center or projection rebuilds only the look-up table.

From a change of center to a frame ready for display takes 16.4 ms (median of 30 changes, each to a new center so that no table is reused; the look-up table alone is 6.0 ms). Conditions: azimuthal equidistant, $1,920 \times 1,080$, coastlines only, RTX 4000 SFF Ada, viewer rendering functions only (texture upload and presentation excluded). The center is a control, not a configuration setting.

Limitations

The comparison is between projections, not tools: ADS-B displays use Web Mercator because angle preservation and rectangular tiling are what they need, and nothing here shows that another choice would serve them better. The engine is a single-author local application, not a service, and not every projection has been checked against every feature.

Open data statement

Aircraft state vectors are from the OpenSky Network [2] under its non-commercial research terms; the underlying records, identifiers and derived datasets are not redistributed with this work. Airport coordinates are from Natural Earth 10m airports v5.0.0 (GCS_WGS_1984); coastlines are Natural Earth 110m; land fill is GSHHG 2.3.7 (Wessel & Smith).

Reproducibility statement

The figures come from a standalone script that calls the engine’s projection modules (sphere of radius 6,371 km) and have been checked against an independent computation (the poster uses a later snapshot). The engine’s source code is not publicly released; the data sources above suffice to reproduce the figures with other tools.

Use of AI

Generative AI assistants (ChatGPT, Gemini, Claude) were used for language proofreading and $\text{\LaTeX}$ typesetting only; the author verified every claim and is solely responsible for the manuscript.

References

- [1] B. Jenny, B. Šavrič, and J. Liem. “Real-time raster projection for web maps”. In: *International Journal of Digital Earth* 9.3 (2016), pp. 215–229. DOI: [10.1080/17538947.2014.1002867](https://doi.org/10.1080/17538947.2014.1002867).
- [2] M. Schäfer, M. Strohmeier, V. Lenders, I. Martinovic, and M. Wilhelm. “Bringing up OpenSky: A large-scale ADS-B sensor network for research”. In: *Proc. 13th Int. Symp. on Information Processing in Sensor Networks (IPSN)*. IEEE. 2014, pp. 83–94.