Code as Cartography

Michael Bostock & Jason Davies

To cite this article: Michael Bostock & Jason Davies (2013) Code as Cartography, The Cartographic Journal, 50:2, 129-135, DOI: 10.1179/0008704113Z.00000000078

To link to this article: https://doi.org/10.1179/0008704113Z.00000000078

Published online: 18 Nov 2013.

Submit your article to this journal

Article views: 440

View related articles

Citing articles: 4 View citing articles
Code as Cartography

Michael Bostock\textsuperscript{1} and Jason Davies\textsuperscript{2}

\textsuperscript{1}The New York Times, USA. \textsuperscript{2}Freelance Programmer, UK.
Email: mbostock@gmail.com; jason@jasondavies.com

Keywords: dynamic cartography, programming, web standards, vector graphics, map projections, data formats

It is now common practice to distribute maps not as static images, but as source code that renders in a web browser (Figure 1). This trend reflects the expansion of computer-assisted cartography over the last several decades; as computers and web standards have become more capable, it is not merely feasible to realize maps as small, interpreted software programs, but often the most convenient method of development and deployment.

Furthermore, in a world awash with data and with powerful computers that fit in the palm of your hand, the demand for relevant information displays has never been greater. Relevance may dictate the construction of highly-personalized maps (e.g. the viewer’s immediate vicinity) or showing real-time information in geographic context (e.g. air traffic control); when needed views cannot be anticipated, dynamic mapping is the only solution. Even for use cases that are readily satisfied by a static map, interactivity can add value by allowing the viewer to zoom, filter and retrieve details on demand (Shneiderman, 1996).

Yet to talk of dynamic cartography on the web as a single technique is misleading, it takes diverse forms, each evolved to suit a particular need. The tile quadtree approach to mapping popularized by Google is well known (Google Maps, 2013). New specialized typefaces allow iconic geography to be embedded directly into text: Alaska \textcopyright, California \textcopyright, Hawaii \textcopyright, and so on (ProPublicaStateFace, 2013). Of course, tiling precomputed raster imagery and glyphs are but limited forms of mapping. Substantially, greater expressiveness (not to mention escaping web Mercator) can be achieved by rendering vector geometry on the client. This article gives a brief overview of some new techniques, and related considerations, in web cartography.

**Discovery and manipulation**

Finding relevant data persists as a huge challenge for non-domain experts. Despite recent positive examples of user-friendly data releases (The National Atlas of the United States of America, 2013; NASA Visible Earth, 2013) and increased awareness of the importance of usability in data releases, much critical data remains hard to find and use. Cryptically-named files and broken links to documentation – or failing that, commented FORTRAN – often lend a feeling of archeology to data discovery.

Fortunately, the collaborative nature of web development has given rise to secondary sources that provide more convenient access. Figure 2 is a simple and intuitive interface for downloading digital elevation data from the Shuttle Radar Topography Mission. The Natural Earth project (Natural Earth, 2013) provides high-quality global vector and raster data available for free. The authors likewise publish scripts (US Atlas TopoJSON, 2013; World Atlas TopoJSON, 2013) for retrieving cartographic boundaries from the US Census Bureau and other sources, building on top of free open-source tools (GDAL, 2013) for manipulating and transforming geospatial data.

Data releases that expose a simple application programming interface encourage the development of diverse user interfaces, and ultimately better satisfy the varied needs of data consumers. Interoperability and hyperlinking is, after all, the ethos of the web.

**Serialisation and deserialisation**

For geometric data to be used by a web browser, raw bytes must first be decoded into representative geometric objects, a process known as deserialisation. Since this code, too, must run in a browser, web developers favour formats that can be easily deserialized. The current format of choice is GeoJSON (Butler \textit{et al.}, 2013), a subset of JavaScript Object Notation (JSON); since JSON is supported natively by browsers, deserializing GeoJSON is trivially executed as JSON.parse. GeoJSON offers additional convenience for novice cartographers in that it can be edited in a basic text editor, enabling the removal of features or modification of properties without specialized software.
Binary formats, despite their ubiquity in desktop geographic information systems and government data releases, are almost never used directly on the web. Conversion to web-friendly formats would be a major hurdle, save for free tools developed by the open-source community (OSGeo, 2013). Still, this is inconvenient; wider adoption of open standards when releasing data would be greatly welcomed and ease integration of data from multiple sources.

File size and geometric complexity are crucial for web cartography. Cartographic boundaries in the tens or hundreds of megabytes are frequently used in desktop systems, but cause noticeable lag when downloaded by a web browser. Simplifying geometry to the displayed resolution is necessary not only to hasten file transfers but to achieve interactive framrates. GeoJSON’s simplicity, though convenient, fails to represent geometry compactly; GeoJSON also lacks topological information needed for applications such as geometry simplification (Bloch, 2007), colouring and cartograms (Dorling, 1996).

To address these issues, the authors have proposed a related format: TopoJSON (2013). Rather than representing geometries discretely, geometries in TopoJSON are stitched together from shared line segments called arcs. Combined with fixed-precision encoding for coordinates, or quantisation, TopoJSON files are often 80% smaller or more than their GeoJSON equivalents. Figure 3 shows a comparison of file sizes, with and without simplification.

**PROJECTION PIPELINE**

A dynamic map may involve changing geometries, changing projections, or both. A fast projection system is paramount, particularly as the performance characteristics of target devices and web browsers vary widely.

As part of the D3 visualisation library (Bostock et al., 2011), the authors have developed a general mapping framework in JavaScript, the native language of the web. D3’s *projection pipeline* executes serial geometric transformations to render vector data. To minimize the creation of temporary objects, input geometry is converted to a stream of events represented by procedural function calls; each stage in the pipeline can then transform events before passing them to the next stage. The output of the pipeline is expressible as Scalable Vector Graphics (SVG) (Ferraiolo et al., 2003), a declarative standard supported by modern browsers, or rendered immediately onto HTML5 Canvas (Cabanier et al., 2012). The modular nature of the pipeline allows web cartographers to compose geometric operations as needed.
Arbitrary aspects
To enable arbitrary aspects, the pipeline includes a stage to rotate geometry objects about the three orthogonal axes of the sphere. Rotation angles are specified in order corresponding to the projection’s origin \((\lambda, \phi)\) and azimuthal rotation \((\gamma)\). Projections formulated for the sphere can thus assume a single aspect with \((0^\circ, 0^\circ)\) at its centre, simplifying implementation and improving performance when rotation is not desired.

Cutting and clipping
Projecting individual geographic points to 2D Cartesian coordinates is a straightforward application of the projection’s mathematical definition. Implementations of a wide range of projections are readily available (PROJ.4, 2013).

Yet vector geometry also comprises lines and polygons, introducing significant additional challenges (Figure 5). Planar projections of the sphere are necessarily discontinuous; geometry objects that span a discontinuity are split into disparate pieces. For example, most non-azimuthal projections in their normal aspect have a discontinuity along the antimeridian at longitudes \(\pm 180^\circ\), causing a line crossing the antimeridian to become two lines when projected.

This issue is traditionally circumvented by inserting static cuts in the data for objects that cross the antimeridian. However, this approach presumes that the projection has its discontinuity exactly along the antimeridian, which is not
always the case, such as for polar aspects of the orthographic projection. For arbitrary oblique aspects constructed interactively through rotation of the sphere, as in Figure 2, static cuts are not only insufficient but counterproductive. An antimeridian cut in a polar polygon such as Antarctica produces a self-intersecting polygon in spherical coordinates, which is problematic due to ambiguity.

A more versatile solution is to clip objects on the sphere dynamically, given the aspect and discontinuities of the projection. Clipping a geometry object in an early stage of the projection pipeline may produce multiple objects as input to the following stage, or none if the object is entirely invisible. The objects can then be simply projected pointwise while avoiding the discontinuity.

This approach also works for incomplete projections. For example, only the near-side hemisphere is visible with the orthographic projection, else features on the opposite hemisphere would be coincident when projected. The invisible objects must be removed entirely, or clipped, while visible parts require interpolation along the edge of the clip region.

D3’s geographic clipping supports cuts along the antimeridian, allowing most non-azimuthal projections to be viewed in arbitrary aspects. For azimuthal projections, objects may instead be clipped against a small circle of arbitrary radius, centred at the projection’s origin. A second clipping stage post-projection clips objects against the 2D rectangular viewport, improving performance by excluding offscreen objects, and avoiding excessively large coordinates generated by projections such as Mercator.

The resampling stage is adaptive in that it tries to interpolate only when doing so reduces distortion (Figure 6), computing triangle areas of successive line segments as an estimator of complexity; in effect, this is the reverse of standard line simplification algorithms (Visvalingam and Whyatt, 1993). An additional benefit of resampling is that geodesics such as graticule meridians do not need to be pre-segmentized, as they are resampled as necessary when projected.

The treatment of line segments as geodesics is by no means standard. Geometry algorithms for an infinite 2D plane are simpler than for a finite spherical surface; thus, many tools assume plate carrée coordinates, with line segments interpolated linearly. Linear interpolation between line endpoints in this way is not rotationally invariant, which causes problems when rotating features to obtain arbitrary projection aspects. Geodesics are preferable since they may be interpolated and rotated in any order without issue, and concisely specify the shortest paths between two points on a sphere.

Composite projections
The minimal definition of projections in D3, as shown in Figure 4, facilitates the derivation of hybrid projections with arbitrary aspects and clip regions. One projection can be smoothly blended with another using linear interpolation (Figure 7). The generic design of adaptive resampling extends to hybrid projections as well. A practical application of composite projections is that they can be used to switch seamlessly between projections at different zoom levels, latitudes and viewport aspect ratios, balancing various desirable criteria (Jenny, 2012) while preserving object constancy (Heer and Robertson, 2007) and continuity. Contrast with naïve per-point interpolation between projected geometries, which requires the source and destination to have the same number of points, and precludes the use of dynamic resampling to reduce distortion during the transition.
Inverse projections

For interaction, inverse projections are needed to convert the mouse or touch position from screen coordinates back to geographic coordinates. A projection without an inverse may therefore be unusable for interactive web cartography. Many popular projections include formulae or procedures for inverse transformation. The more esoteric projections may not have published inverse transformations, but in most cases such a transformation can be computed efficiently using Newton–Raphson (Ipbüker and Bildirici, 2012).

function d3_geo_mercator(λ, ϕ) {
  return [λ, Math.log(Math.tan(π / 4 + ϕ / 2))];
}

d3_geo_mercator.invert = function(x, y) {
  return [x, 2 * Math.atan(Math.exp(y)) - π / 2];
};

Figure 4. D3’s spherical Mercator projection, and its inverse. This raw projection is typically composed with D3’s other projection features, including three-axis rotation, adaptive resampling, antimeridian cutting and viewport clipping.

Inverse projections

For interaction, inverse projections are needed to convert the mouse or touch position from screen coordinates back to geographic coordinates. A projection without an inverse may therefore be unusable for interactive web cartography. Many popular projections include formulae or procedures for inverse transformation. The more esoteric projections may not have published inverse transformations, but in most cases such a transformation can be computed efficiently using Newton–Raphson (Ipbüker and Bildirici, 2012).

CREATION VERSUS CONFIGURATION

The tools we employ influence the artefacts we produce by making certain tasks easier or harder. It is tempting to empower a wide audience of web cartographers by simply enumerating common map types in a point-and-click interface or as parameterized components. Yet an exhaustive typology of maps, as with charts in information visualisation (Wilkinson and Wills, 2005), is impossible. Such an approach may satisfy common needs but ignores the long tail of diverse applications.

Simply put, there is no substitute for writing code. The creation of maps through code affords the composition of techniques and interfaces in ways wholly unanticipated by the toolmaker, greatly increasing the cartographer’s ability for self-expression. A sample of the diverse possibilities is shown in Figure 8. This is not to suggest that every map

Figure 5. Several examples of clipping an arbitrary aspect of Albers’ equal-area conic projection

Figure 6. A comparison of resampling methods, shown with a 10° graticule on the Cassini projection. From left to right: no resampling, uniform resampling, and adaptive resampling. The adaptive method concentrates samples in areas of high curvature, improving the appearance of discrete geometry without penalizing performance.
should be built from scratch. While not appropriate for every application, the authors have tried to produce a tool that balances efficiency (the effort required to specify a visualisation) with expressiveness (the diversity of possible visualisations).

**BIODIGNAPICAL NOTES**

Mike Bostock is a graphics editor for *The New York Times*. He is also the creator of D3.js, an open-source library for visualizing data using web standards. Before *The New York Times*, Mike was a visualisation scientist for Square and a computer science PhD student at Stanford University. Mike received the BSE degree in computer science in 2000 from Princeton University.

Jason Davies is a freelance programmer. His work and interests involve combining visualisation with mathematics, data structures and algorithms. He is a co-author of D3.js, an open-source visualisation library. Jason graduated with a BA degree in Computer Science from the University of Cambridge in 2005.

**REFERENCES**


