DOI: 10.14714/CP96.1645

PEER-REVIEWED ARTICLE

Florence: a Web-based Grammar of Graphics for Making Maps and Learning Cartography

Ate Poorthuis
KU Leuven
ate.poorthuis@kuleuven.be

Lucas van der Zee
KU Leuven
luuc.vanderzee@kuleuven.be

Grace Guo
Georgia Tech
gguo31@gatech.edu

Jo Hsi Keong
Singapore University of Technology and Design
johsi.k@gmail.com

Bianchi Dy
Singapore University of Technology and Design
bianchi_dy@sutd.edu.sg

Online, web-based cartography workflows use a dizzying variety of software suites, libraries, and programming languages. This proliferation of mapmaking technologies, often developed from a software engineering rather than a cartographic foundation, creates a series of challenges for cartography education, research, and practice.

To address these challenges, we introduce a JavaScript-based open-source framework for web-based cartography and data visualization. It is built on top of existing open web standards that are already in intensive use for online mapmaking today, but provides a framework that is firmly based on cartographic and visualization theory rather than software engineering concepts. Specifically, we adopt concepts from Bertin's Semiology of Graphics *and Wilkinson's* Grammar of Graphics *to create a language with a limited number of core concepts and verbs that are combined in a declarative style of "writing" visualizations. In this paper, we posit a series of design guidelines that have informed our approach, and discuss how we translate these tenets into a software implementation and framework with specific use cases and examples. We frame the development of the software and the discussion specifically in the context of the use of such tools in cartography education.*

With this framework, we hope to provide an example of a software for web-based data visualization that is in sync with cartographic theories and objectives. Such approaches allow for potentially greater cartographic flexibility and creativity, as well as easier adoption in cartography courses.

KEYWORDS: cartography; geovisualization; web mapping: grammar of graphics; software; education

INTRODUCTION

Cartography has always relied on technology in the pursuit of making maps. The start of the twenty-first century is no different in that sense. But today's technology has enabled radical changes, and a proliferation in not only how we make, but also how we consume maps. A host of technologies that came along with Web 2.0 over the last twenty years has now changed significantly how we share and read (online) maps, even jumpstarting the concept of "viral" cartography (Muehlenhaus 2014; Robinson 2019; Shannon and Walker 2020). New media, such as web-based and other online maps, have also opened up new opportunities for readers to interact with the map (Roth and MacEachren 2016), and have quickly become one of the research frontiers in cartography and geovisualization (Griffin, Robinson, and Roth 2017).

Along with these changes in consumption, the actual practice of creating maps—the *how* of mapmaking—has

also shifted. Computer-based cartography in the late twentieth century was conducted primarily using a small number of desktop user interfaces—including mainstays such as ESRI's *ArcMap* GIS software and Adobe's vector editing software, *Illustrator*—that were employed to create static maps. In contrast, "new" workflows that are focused on creating web-based maps use a dizzying variety of software suites, libraries, and programming languages. As just one example, Roth et al. (2014) include 35 different technologies in their 2014 assessment of web mapping technologies. The six years since their tally have not seen a convergence of these technologies—far from it.

This proliferation of map consumption and production has brought many new people—not necessarily trained in cartography—to the practice of making maps, and has created a unique opportunity or even a renaissance for the academic field of cartography (MacEachren 2013). Certainly, (new) online mapping programs and certificates such as the University of Kentucky's New Maps Plus and Penn State's online geospatial education are an indication of a healthy interest from both academia and industry in all these new changes and possibilities. As a side-effect of this process, new cartographic software now draws from a wide range of software development paradigms, reflecting the variety of backgrounds of its developers. While this has created a welcome diversity in the cartographic software landscape, the drawback is that new technologies may speak less directly to a consistent set of cartographic principles.

This current software landscape creates a series of challenges for cartography education, research, and practice. First, the computer science paradigms on which many new technologies are based can be challenging for students without prior training in computer science or programming experience. This creates barriers to entry and can distract from teaching cartographic core principles and theory (Sack and Roth 2017; Sack 2018; Ricker and Thatcher 2017). Second, many new tools are less grounded in traditional cartographic theory, in ways that can limit their flexibility—the much-maligned use of the default Web Mercator projection in many mapping tools being a case-in-point (Battersby et al. 2014; Šavrič, Jenny, and Jenny 2016). Third, the absence of a single, canonical technology suite or paradigm limits the transferability of skills between all these different types of software and libraries.

In this paper, we speak to these challenges by introducing a JavaScript-based framework for web-based cartography and data visualization. It is built on top of existing open web standards that are already in intensive use for online mapmaking today, but provides a framework—or Application Programming Interface (API) in technical terms—that is based firmly on cartographic and visualization theory rather than software engineering concepts. Specifically, we adopt concepts from Bertin's *Semiology of Graphics* (2010) and Wilkinson's *Grammar of Graphics* (2013) to create a "language" with a limited number of core concepts and verbs that are combined with a declarative style of "writing" visualizations. With this framework, we hope to provide an example of software for web-based data visualization that is in sync with cartographic theories and objectives, and thus allows greater cartographic flexibility, lets users be more creative, and is potentially easier to adopt in cartography courses.

In the next section, we will first unpack in greater detail the aforementioned challenges surrounding web-based cartography. In doing so, we provide a survey of the current practice of online, interactive cartography. Subsequently, we outline the core tenets of our approach and describe the core elements of the framework. Finally, we will provide specific use cases and examples of how these core elements can be combined flexibly to create cartographic visualizations. We will end the paper by discussing our approach and looking ahead to potential future work.

CURRENT SOFTWARE AND PRACTICES FOR CREATING AND TEACHING WEB-BASED CARTOGRAPHY

THE LINK BETWEEN CARTOGRAPHIC THEORY AND MAPPING SOFTWARE

Cartography has a long-standing tradition of building theory around maps. What are maps? How do they represent and produce the world (Crampton 2010)? And how can we think systematically about their construction (MacEachren 2004; Kraak and Ormeling 2011)? At the heart of this systematic approach to cartography, we find

Bertin's idea of visual variables (Bertin 2010). Coined in his 1967 *Semiology of Graphics*, it is still influencing cartographic thought today (cf. the recent special issue on Bertin's legacy in *Cartography and Geographic Information Science*; Harvey 2019).

If we look beyond cartography to the related field of information visualization, Bertin's original ideas around visual variables have been further formalized into a system that Wilkinson dubbed the "grammar of graphics" (2013). Although there are different versions and interpretations of this system (cf. Munzner 2014 for a comprehensive treatment), it generally relies on a process of translating[1] data values (be they quantitative or categorical) into the visual variables[2] (e.g., position, size, or colour) of a graphical mark[3] (e.g., a point or a line). While Wilkinson's original publication was accompanied by a software implementation of the system, the grammar of graphics didn't significantly catch on in practice until Hadley Wickham implemented a version of it for the R programming language (Wickham 2010). Wickham's implementation—*ggplot2*—has helped transform R into one of the key languages for data visualization (including maps) used today. It has also inspired the adoption of its grammar-of-graphics API into a range of other programming languages.

This translation of theory into software brings us to the nexus of cartographic theory and its practice. The connection between the two is at the core of cartography (cf. Tobler 1959). After all, academic cartographers themselves often combine thinking and theorizing about maps with the act of making maps. A common approach to computer-based mapmaking uses desktop software with graphical user interfaces, often combining a GIS software (e.g. Esri's ArcMap) with a vector-editing graphics program (e.g., Adobe's Illustrator) in a single workflow. If not directly built on top of, these softwares are at least very much compatible with cartographic theory. For example, they provide straightforward workflows to build multiple layers in a map, assign data variables to visual variables, or change a map's projection. We see this reflected in the discipline's textbooks as well: many of the oft-used cartography textbooks cover the theory and practice of mapmaking *without* going into specific software-based how-to instructions (e.g., Kraak and Ormeling 2011; Slocum et al. 2009; Dent 2009). The "translation" to software is left to an instructor's own lab materials or a compendium book. This split of concerns seems to work reasonably well partly because the software and the theory are in sync.

THE ECOSYSTEM OF WEB MAPPING AND INTERACTIVE DATA VISUALIZATION

Building on top of cartography, the field of geovisualization, with its genesis in the 1980–1990s personal computer era, has capitalized significantly on the affordances of new (web) technologies to build interactive mapping interfaces. We see examples of this in early work focused on exploratory data analysis (Anselin, Kim, and Syabri 2004) to more recent examples focused on understanding output of specific algorithms (Fabrikant, Gabathuler, and Skupin 2015) or specific data sets (Pezanowski et al. 2018). Geovisualization systems have become powerful mapping and analytical tools for the end user. Despite their power, and often due to their bespoke design, they can be remarkably easy and convenient to use (e.g., Nost et al. 2017; Roth, Ross, and MacEachren 2015). In contrast, *creating* such geovisualization tools remains a complicated endeavour, often performed by experts. Although there is promising work focused on making geovisualizations easier to create, for example through no- or low-code software (e.g., Gahegan et al. 2002; Hardisty and Robinson 2011), geovisualizations seem to rely on the use of a wide variety of programming languages and software libraries, without a single, or even a small, set of canonical approaches emerging.

This proliferation of different approaches is not due to a lack of promising work in (academic) cartography. For example, Nagel et al. (2013) developed a library, *Unfolding*, for writing interactive maps in Processing and Java, while Ledermann and Gartner (2015) provide a "cartographic" API for making maps with JavaScript (JS). More recently, Degbelo, Sarfraz, and Kray (2020) created *AdaptiveMaps*, a no-code semi-automatic approach to making thematic web maps that is based on Bertin's visual variables. This no-code or low-code approach also shares similarities with the full-stack (i.e., covering both spatial analysis and cartographic functionality) approaches to web cartography that commercial providers are now offering—CARTO,

1. Or encoding, or mapping (depending on the author)

2. Or channels, or aesthetics, or dimensions

3. Or geometries

Mapbox, and Esri are primary examples of companies in this space.

Nonetheless, many online web maps are still made by utilizing one or more smaller JavaScript libraries. For maps specifically, *Leaflet* (**leafletjs.com**) and *OpenLayers*(**openlayers.org**) are the go-to technologies for creating "slippy" maps. They use a basemap that can be panned and zoomed, similar to the map solutions by Apple, Google, and Microsoft that have become commonplace. These base maps can subsequently be overlaid with additional (thematic) map layers. Similarly, for larger datasets, WebGL approaches (which utilize a computer's Graphical Processing Unit for greater performance) such as *Deck.gl* (**visgl.github.io/deck.gl**) and *Mapbox GL* (**docs.mapbox.com/mapbox-gl-js/api**) exist. For cartographic work that goes beyond the "basemap + thematic overlay" paradigm, the collection of JavaScript modules collectively referred to as *D3* (Bostock, Ogievetsky, and Heer 2011; **d3js.org**) has become a commonly used tool.

The reference to D3 also brings us to the connection of cartography with the larger field of information visualization. D3, although used extensively for mapmaking, did not emerge from or for cartography specifically. Instead, its key contributors, Heer and Bostock, laboured to devise a system that makes it possible to design interactive data visualizations in a much more broader sense of the word (Bostock and Heer 2009; Heer and Bostock 2010). D3 is a relatively low-level implementation of their approach in JavaScript. More recently, Heer and colleagues have created the *Vega* system that operates at a higher abstraction level and is more squarely based on the grammar of graphics (**vega.github.io**). It is relatively language-agnostic, as it stores and describes visualizations with the interoperable JSON standard (Satyanarayan et al. 2017). Along similar lines, *Data Illustrator* merges the grammar of graphics with vector editing into a single system (Liu et al. 2018) through the automatic binding of data variables to visual components that designers can easily work with. Although many information visualization libraries also support the creation of maps, support for key cartographic principles (e.g., the selection of an appropriate projection) is often not a primary concern.

CHALLENGES FOR EDUCATORS

As we stated before, this splintered state of affairs in web mapping brings with it a few specific consequences. With so many different (programming) technologies available, it has become a formidable challenge to teach online web mapping—especially in the context of curricula that are not focused heavily on software engineering. As Sack (2018, 39) recently pointed out while taking the pulse of web mapping education in the United States: "*The two greatest challenges in teaching web mapping were, unsurprisingly, teaching students how to code and keeping up with rapid technology changes in the industry.*"

There are different responses possible to the challenge of teaching students how to code, which also depend strongly on the specific degree programme in which a web mapping course or module is offered. One such approach, partly supported by newer tools such as Esri's online suite, is to use low-code solutions. However, as creative or bespoke online cartography still requires manual coding, relying only on such solutions might be detrimental to the field at large—especially since programming skills are becoming increasingly useful in other parts of our discipline. Another approach is to treat courses that rely on programming as more advanced or upper-level and to set up specific prerequisites to enrolment. This has several potential downsides (cf. Ricker and Thatcher 2017)—one of which is an increase in the barriers to entry, which is especially poignant for domain experts for whom programming is often a means to an end.

Instead, we would like to argue for a continued emphasis on teaching programming to cartography and GIS students. The base technologies of the modern web (HTML, CSS, and JavaScript) have reached a level of maturity and consistency that makes learning them more straightforward now, compared to the state of affairs at the start of the millennium. We posit that it is mainly the "jungle" of web mapping software, built on top of those base technologies, that proves difficult to teach. There are several reasons for this. Different web mapping software and libraries do not operate from a consistent foundation and implement similar things in different ways. Furthermore, many technologies do not "sync" well with cartographic theory. This leads to situations where instructors need to reserve a significant amount of class time to teach the idiosyncrasies of a library rather than core cartographic principles. If we add to that the fact that many new tools seem to be using significantly different approaches, it is no wonder educators are hesitant in taking on this task.

This challenge is exacerbated by the fact that many of the current web mapping libraries are either developed outside of the discipline or have (design) goals that are not

necessarily in line with cartographic principles. A simple illustration: the production of a thematic choropleth map that uses a classification scheme to translate a quantitative variable to a limited set of colours on the map is a mainstay in any cartography class. However, it is a surprisingly complicated undertaking in most of the popular mapping libraries. For the Leaflet mapping library, it requires the developer to write a custom function to implement a classification scheme, and another custom mapping function that contains logic to translate data values to colours. And then we haven't even tried to use a non-Mercator projection! While this approach might be sensible or even preferred from a software engineering perspective, it becomes a pedagogical distraction in a cartography class—akin to asking a student in an introductory statistics class to write and implement their own fitting function for a linear regression.

CONSISTENT SOFTWARE DEVELOPMENT BY AND FOR CARTOGRAPHERS

Drawing parallels with the discussion around geocomputation and GIS (Harris et al. 2017; Gahegan 2018; Poorthuis and Zook 2020), we argue that the academic field of cartography can address the challenges around web mapping by playing a more prominent role in developing software for this purpose, and, in doing so, build stronger connections between visualization practice and cartographic and visualization theory.

Here we also draw inspiration from Wickham's development of the ggplot2 library for the R programming language (Wickham 2010), which adopts the grammar of graphics as its theoretical foundation. Somewhat in parallel with ggplot2, the R community has developed a constellation of libraries collectively referred to as the *tidyverse* (Wickham et al. 2019) that provides a consistent approach and design to common data science tasks, from data manipulation, to modelling, to visualization. It is predominantly developed by and for a community of domain experts and users, including efforts such as rOpenSci (Boettiger et al. 2015) that organize peer review of software. New libraries are continuously being developed and adhere to the same tidyverse design principles. This consistency is key: the adoption of additional libraries becomes much faster and easier for users as they don't need to grasp a new set of design principles or idiosyncrasies for every additional library. In an education context, this means that only a limited set of software design concepts needs to be taught and focus can otherwise remain on domain concepts.

Tidyverse-compatible libraries for mapping exist as well: *ggmap* (Kahle and Wickham 2013) and *tmap* (Tennekes 2018). In fact, the tidyverse ecosystem can be used very effectively by the modern cartographer, but is limited in its facility for interactive maps. Most interactive maps are created for the web, and creating content for the web is not (yet) one of R's core strengths—although possibilities do exist (Chang et al. 2019). We highlight the success of the tidyverse approach as an inspirational example and ask how we can translate these lessons to the field of web mapping.

A WEB-BASED GRAMMAR OF GRAPHICS FOR MAPMAKING

In this paper, we introduce Florence: a web-based mapping and visualization library that is aimed at addressing the challenges outlined in the previous section. To do this, we have used a specific set of core tenets as design guidelines (DG) in developing the library.

- **DG1:** A web mapping library should be built *on top of* modern web standards. Using and teaching Florence means teaching these technologies (CSS/HTML/JS), instead of replacing or hiding them. Florence is a relatively small convenience layer around those technologies, with its main purpose being to re-anchor web mapping on cartographic theory.
- **DG2:** A web mapping library should take note of and leverage the current generation of JavaScript frameworks. Such frameworks make web development faster and more convenient. Adopting them also allows for visualizations to integrate more seamlessly in larger web development projects. In addition, skills gained through using the library for the purpose of mapmaking will transfer to other domains (e.g., UI/UX design; Roth 2017).
- **DG3:** A web mapping library should be modular rather than one-size-fits-all. This means a reliance on small(er) building blocks that can be mixed together

creatively. Similarly, it should allow the user to extend and build their own modules for oft-used functionality or specific visualizations.

- **DG4:** A web mapping library should perform as little magic or "black box" behaviour as possible. While such "one-click" solutions might entice new users, they generally inhibit an immediate understanding of how and why things work. In addition, black boxes can ultimately make creative, custom use more difficult. This is an explicit deviation from the low/no-code approach. We do not advocate for using less code but instead endeavour to make code easier to understand and reason about.

- **DG5:** A web mapping library should allow for the declarative authoring of visualizations, rather than the more common imperative approach. Imperative programming—giving step-by-step instructions that state *how* you build up to a final goal—for visualizations can often be difficult to reason about, as the reader/author needs to build up a mental picture of the visualization by running through all the imperative steps in the code. Declarative programming—stating *what* the final goal should look like—is a better fit for cartography and follows a larger trend in information visualization (Heer and Bostock 2010; Satyanarayan et al. 2016). Many recent JavaScript frameworks (cf. DG2) allow for the adoption of this approach.

- **DG6:** A web mapping library should be anchored explicitly on a theoretical foundation. Florence is based on the grammar of graphics (adapted for mapmaking purposes). This makes it easier to switch between spatial and non-spatial visualizations and build (linked) geovisualizations with graphs and maps using the same toolset.

- **DG7:** A web mapping library should provide easy ways to "escape" the software abstraction provided. If a user wants to get creative and go more low-level and use native SVG, or add on another visualization library, they should be able to do so. Similarly, if they prefer to work with a higher-level of abstraction, ready-made modules for commonly used visualizations should be provided or possible to create.

CORE ELEMENTS

With this set of design guidelines, we built Florence on top of Svelte: a reactive JavaScript framework that is notable for its simplicity and easy learning curve (**svelte.dev**). Svelte is structured around declarative "single file components" that combine the three core web technologies into a single file: HTML and SVG markup for layout; JavaScript for interaction and computation; and CSS for styling (DG1). Svelte files look very similar to regular HTML files, because they are effectively standard HTML files with a little bit of extra logic sprinkled in through a well-designed template syntax. Importantly, this syntax allows users to create connections between HTML (layout) and JavaScript (interaction and computation) to build declarative and reactive components and pages—which is exactly what is needed to build visualizations and web maps.

There are a number of additional, more technical advantages that Svelte provides over other JavaScript frameworks, but its main reason for adoption here is the ease with which Svelte can be learned and adopted (DG2). This is especially the case compared to other frameworks, such as React, that rely on powerful but complex software engineering concepts that are relatively difficult to learn for non-software engineers.

The central piece of Svelte's template syntax is the use of curly braces ({}) in HTML mark-up. Any JavaScript (or references to JS variables) inside such braces will be automatically evaluated. Importantly, updates to variables will automatically be reflected in the rendered page. As such, Svelte's "*Hello World*" is straightforward to understand, even for somebody who has not come into contact with the framework before.

```
<script>
      let name = 'world'
</script>
<h1>Hello {name}!</h1>
```

Built on this technical foundation, Florence provides a series of components that can be imported and combined to build visualizations (DG3, DG5), similar to how HTML elements are combined to build a web page. Figure 1

shows how these components relate directly to the various concepts of the grammar of graphics (DG6). In the next sections, we will discuss the core set of components. A deeper treatment, including documentation and more elaborate examples, can be found at the documentation website (**florence.spatialnetworkslab.org**). Florence can be installed into any JavaScript project through the Node Package Manager (npm) or by forking or extending any of the live code examples on the website. The source code for the software can be found on Gitlab (**gitlab.com/spatialnetworkslab/florence**).

GRAPHIC & SECTION

Every Florence visualization starts with a `Graphic`.[4] A user can think of this as a blank canvas that becomes available as a drawing space. Each `Graphic` has a specific width and height (measured in pixels in these examples, but it can also be made relative to the web page dimensions). The `Graphic` is like a supercharged SVG element—in fact, under the hood, drawing a `Graphic` will indeed draw an SVG element to the page.

In order to create an empty `Graphic` of 500 by 500 pixels, we can import the component from Florence and draw it to the page (Figure 2). Properties of components are specified in a syntax that is similar to HTML attributes. In this instance, we give the width and height properties of the `Graphic` a value of 500.

The `Graphic` has a sister component called a `Section`. As many visualizations consist of multiple panels, facets, and insets, `Section`s can be used to subdivide the `Graphic` for this purpose. Each `Section` has its own dimensions and position and—as we will see later—its own coordinate system. For example, we can draw non-overlapping left and right panels (Figure 3).

The same logic can be used to draw overlapping `Section`s, such as when multiple map layers need to be drawn on top of each other.

MARKS

To actually draw content, we rely on the grammar-of-graphics concept of the *mark*. A mark specifies a

4. Florence components are capitalized to distinguish them from HTML elements. We set them in monospaced type to make it clear when a reference to a component is made.

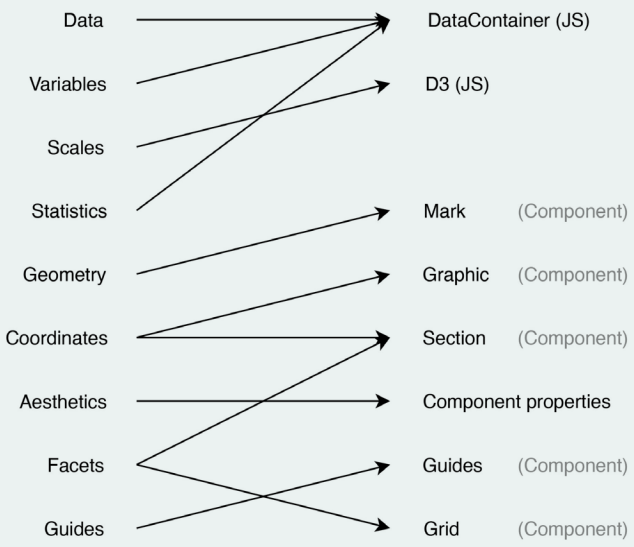
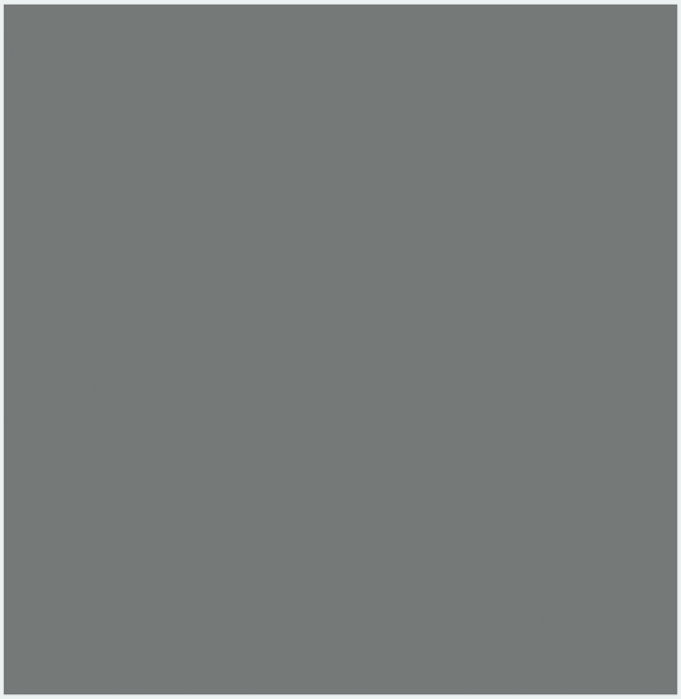


Figure 1. *Relation between Wilkinson's original grammar of graphics concepts and their implementation in Florence, after Wickham's (2010) comparison between Wilkinson and the ggplot2 approach.*

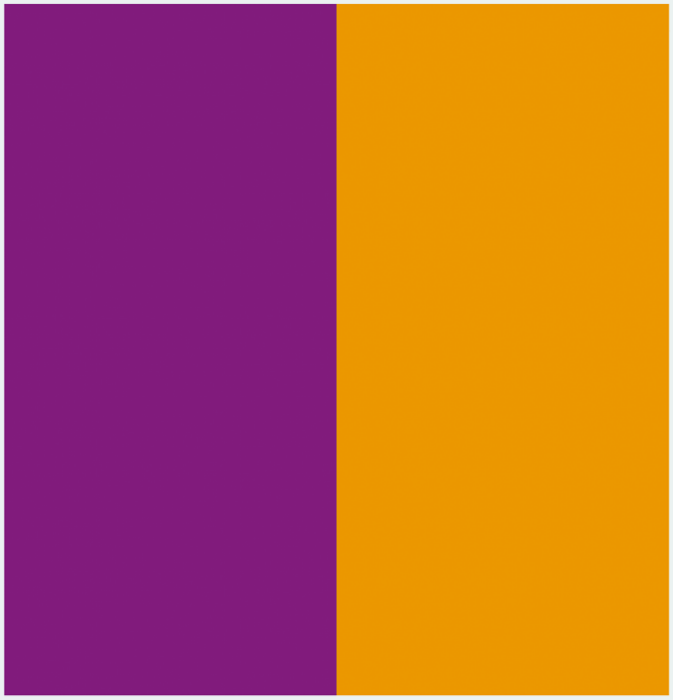


```
<script>
  import { Graphic } from '@snlab/florence'
</script>

<Graphic width={500} height={500} backgroundColor={'gray'}>

</Graphic>
```

Figure 2. `Graphic` *component. Here, and in Figures 3–7, the code is displayed on the bottom with the rendered visualization displayed on the top. Interactive version available at florence.spatialnetworkslab.org/examples/cp-figure2.*

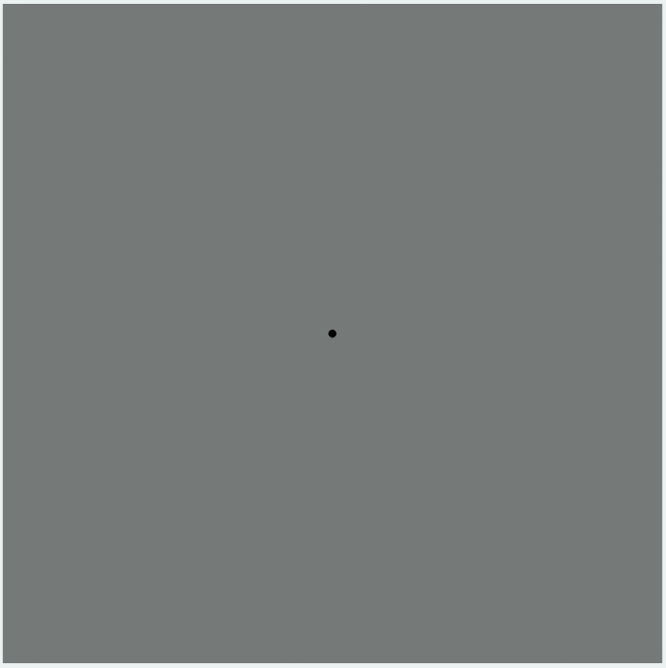
geometric object whose visual properties can encode data attributes. In this sense, marks are similar to the points, lines, and polygons central to vector cartography. Florence makes the following basic marks available: `Point`, `Symbol`, `Line`, `Rectangle`, `Area`, `Label`, and `Polygon`. Each mark supports a set of encoding channels through the properties of its components. These are categorized by position, shape, size, colour, textual attributes, transitional attributes, and interactivity. With these primitive marks, almost any visualization can be expressed. Figure 4 shows a `Point` mark drawn in the centre of our `Graphic` by setting the x and y positional properties of the component.

Of course, most visualizations need to draw not just a single mark but a larger set of them. Florence provides a `Layer` version of each mark for this purpose. Instead of providing a single value each to the x and y properties, we can simply provide an array of values, one for each mark.



```
<script>
  import { Graphic, Section } from '@snlab/florence'
</script>

<Graphic width={500} height={500}>

    <Section
      x1={0}
      x2={250}
      y1={0}
      y2={500}
      backgroundColor={'purple'}
    >
    </Section>

    <Section
      x1={250}
      x2={500}
      y1={0}
      y2={500}
      backgroundColor={'orange'}
    >
    </Section>

</Graphic>
```

Figure 3. `Graphic` *component with two non-overlapping* `Section` *components. Interactive version available at florence.spatialnetworkslab.org/examples/cp-figure3.*



```
<script>
  import { Graphic, Point } from '@snlab/florence'
</script>

<Graphic width={500} height={500} backgroundColor={'gray'}>
  <Point
    x={250}
    y={250}
  />
</Graphic>
```

Figure 4. *A simple* `Point` *mark, with an x and a y property, in the centre of a* `Graphic`*. Interactive version available at florence.spatialnetworkslab.org/examples/cp-figure4.*

The `Layer` will draw as many marks as there are values in the supplied array. For example, to draw three points:

```
<PointLayer
   x={[0, 250, 500]}
   y={[250, 250, 250]}
/>
```

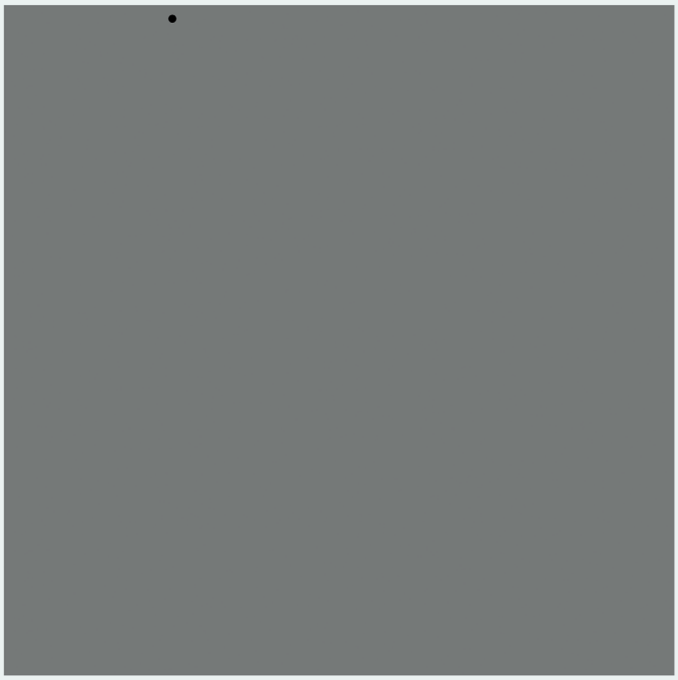
Apart from positioning a `Point` (or any other mark) with individual x and y coordinates, Florence also understands GeoJSON natively. Since GeoJSON is the de-facto standard for storing and sharing spatial data on the web, this is an important advantage for web mapping. Any mark can be given coordinates in GeoJSON format through the `geometry` property. For example, in Figure 5 we have a simple GeoJSON object with a single point representing the Dinagat Islands. Its geometry is directly "given" to the `Point` mark without any need for additional translation.

SCALES

If not specified, the coordinate system used inside the `Graphic` or `Section` will be based on the pixel dimensions. However, for most visualizations and maps we don't want to "think" in pixel coordinates. We might not even know the pixel coordinates in advance, as the visualization needs to grow or shrink dynamically depending on the available screen size. To enable this, we need a process to translate data values to positional values. In the context of the grammar of graphics, this process is most often referred to as *scaling*. In essence, a scaling function takes a data value as input, and outputs the appropriate location on the screen (i.e., a pixel coordinate) – mapping from "data space" to "pixel space."

In many software programs, this scaling is performed hidden from the user. In Florence, we take the opposite approach and make scaling explicit and transparent through user-supplied scaling functions (DG4). Florence is agnostic about the actual scaling functions used. A user can create their own functions, but they can also rely on the D3 scaling functions that have become close to an industry standard for data visualizations. Florence follows the D3 conventions for scaling functions for this reason.

Scaling functions can be passed to the `Graphic` or `Section`, where they will be used to create a "local coordinate" system by using information about the pixel dimensions of the component. Once such a local coordinate system is



```
<script>
  import { Graphic, Point } from '@snlab/florence'

  const myGeoJSON = {
    type: 'Feature',
    geometry: {
      type: 'Point',
      coordinates: [125.6, 10.1]
    },
    properties: {
      name: 'Dinagat Islands'
    }
  }
</script>

<Graphic width={500} height={500} backgroundColor={'gray'}>
  <Point
    geometry={myGeoJSON.geometry}
  />
</Graphic>
```

Figure 5. *A* `Point` *mark positioned with GeoJSON geometry. Interactive version available at florence.spatialnetworkslab.org/examples/cp-figure5.*

created, marks can be positioned in this local coordinate system or "data space," rather than with absolute pixels. This makes it much easier to reason about placing marks and annotations within the visualization, and it allows for the dynamic resizing of any visualization.

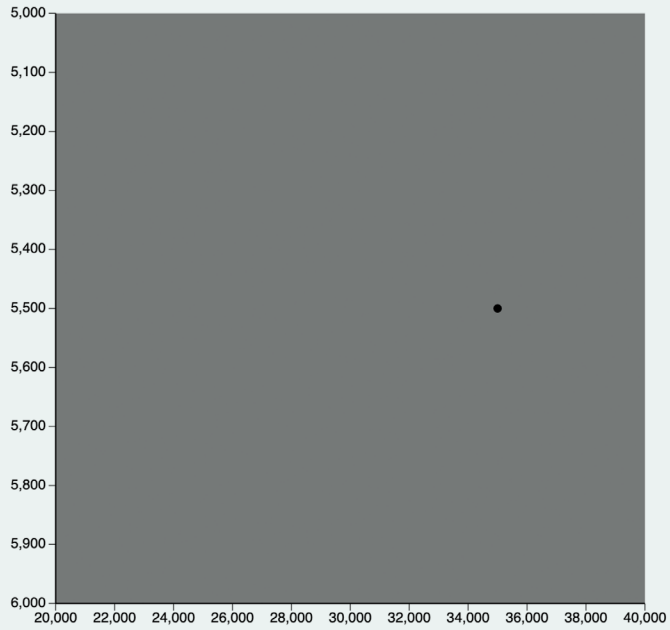
For example, in Figure 6 we create a `Graphic` with an x-axis based on a continuous variable that ranges from 20,000 to 40,000, and a y-axis with quantitative values ranging from 5000 to 6000. We then place a single point inside the `Graphic` at coordinates [35000, 5500] using the local

coordinate system. Note that `{scaleX}` is just a shorthand for `scaleX={scaleX}` in the Svelte framework.

As scaling of geographic coordinates is a special case (i.e., the x and y dimensions generally need to be scaled together to maintain the aspect ratio), Florence provides built-in scaling functions for geographic data. Figure 7 demonstrates how to scale two triangular polygons using their bounding box. The `createGeoScales` function returns an object with a `scaleX` and `scaleY`. The spread syntax (`{...geoScales}`) is a Svelte shorthand for `scaleX={geoScales.scaleX} scaleY={geoScales.scaleY}`.

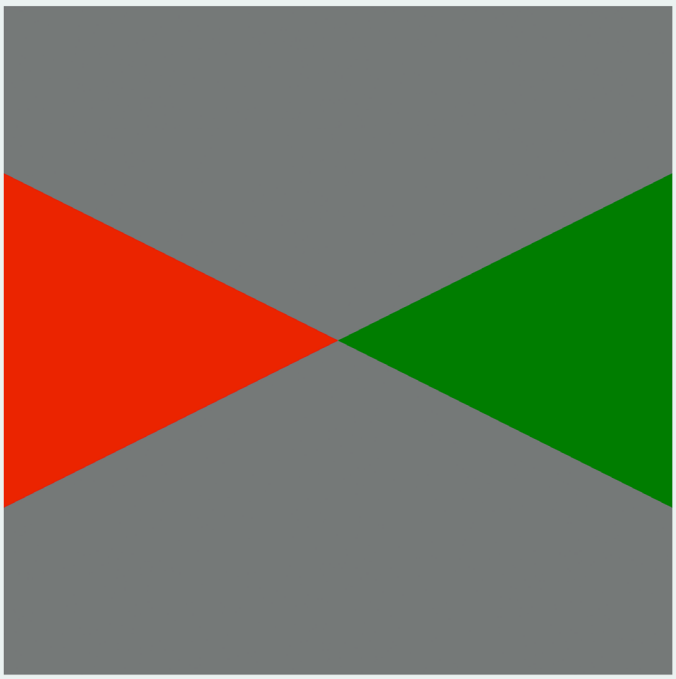
DEALING WITH DATA

Maps, like any visualization, often rely heavily on the transformation, aggregation and filtering of data. Conventional GIS programs offer a wide range of functions for this purpose. While most data transformations can be readily performed in JavaScript, this often requires a high level of JS software engineering knowledge. Moreover, since JS isn't designed as a data science language per se, the mental model for these transformations is much lower-level than ideal for cartography.




```
<script>
  import { Graphic, Point, XAxis, YAxis } from '@snlab/florence'
  import { scaleLinear } from 'd3-scale'

  const scaleX = scaleLinear().domain([20000, 40000])
  const scaleY = scaleLinear().domain([5000, 6000])
</script>

<Graphic
  width={500} height={500}
  padding={40}
  backgroundColor={'gray'}
  {scaleX} {scaleY}
>
  <Point
    x={35000}
    y={5500}
  />

  <XAxis />
  <YAxis />
</Graphic>
```

Figure 6. *A* `Point` *mark positioned in "data space." Interactive version available at florence.spatialnetworkslab.org/examples/cp-figure6.*



```
<script>
  import { Graphic, PolygonLayer, createGeoScales } from '@snlab/florence'

  const triangles = [
    { type: 'Polygon', coordinates: [[[0, 0], [5, 2.5], [0, 5], [0, 0]]] },
    { type: 'Polygon', coordinates: [[[10, 0], [5, 2.5], [10, 5], [10, 0]]] }
  ]

  const bbox = {
    x: [0, 10],
    y: [0, 5]
  }

  const geoScales = createGeoScales(bbox)
</script>

<Graphic
  width={500} height={500}
  backgroundColor={'grey'}
  {...geoScales}
>
  <PolygonLayer geometry={triangles} fill={['red', 'green']} />
</Graphic>
```

Figure 7. *Scaling polygons while maintaining the aspect ratio. Interactive version available at florence.spatialnetworkslab.org/examples/cp-figure7.*

To aid in this, we provide a sidecar data handling library that is designed to mirror the logic and concepts in tidyverse's *dplyr* "grammar of data manipulation" (Wickham et al. 2015) (DG6). In this way, any user familiar with the *tidyverse* approach will be able to adopt its logic quickly. The source code for the library, including documentation on all its functions, can be found on Gitlab at: **gitlab.com/spatialnetworkslab/florence-datacontainer**.

The library allows for loading row and column–oriented datasets, as well as GeoJSON data, into a consistent data structure referred to as a `DataContainer`. This `DataContainer` then offers familiar transformations such as:

- **Select:** for selecting a subset of columns
- **Filter:** for filtering a subset of rows
- **Mutate:** for creating new columns (based on some calculation)
- **Group by:** for aggregating data based on a specific column
- **Summarise:** for summarizing data about each aforementioned group

Many geovisualizations allow the end-user to interactively filter, subset, and aggregate data, so we consider these data transformations as essential ingredients in any web mapping toolset. A `DataContainer` also provides some shortcuts for accessing oft-used information in map design, such as the domain of a variable or its data type.

In addition, it has built-in support for binning with different classification schemes, functionality that is useful for both non-spatial histograms as well as the classification common in choropleth maps. It also allows for the reprojection of spatial geometry data. By building on top of the open source *proj4js* library (**github.com/proj4js/proj4js**), any projection supported by the proj4 ecosystem can be used to create visualizations. An example of this can be seen in Figure 8, which reads in an external GeoJSON file projected in a country-specific coordinate system. This custom projection works out-of-the-box with Florence and the map is automatically sized to fit the dimensions of the `Graphic`. Additional styling is provided through the use of the `fill`, `stroke`, and `strokewidth` component properties (aesthetics). In the second panel, the `fill` aesthetic is mapped to a categorical colour scheme (through the use of a scale provided by D3) based on the province name.

Commonly used elements such legends, graticules, and—for non-map visualizations—axes, can be created with built-in components or the user can create their own custom

```
<script>
  import { Graphic, Section, PolygonLayer, createGeoScales } from '@snlab/florence'
  import DataContainer from '@snlab/florence-datacontainer'
  import { provincesGeoJSON } from './provinces.js'

  const provinces = new DataContainer(provincesGeoJSON)
  const geoScales = createGeoScales(provinces.bbox())
</script>

<Graphic width={500} height={500} {...geoScales} flipY>
  <PolygonLayer
    geometry={provinces.column('$geometry')}
    fill={'rgb(230,230,230)'}
    stroke={'white'}
    strokeWidth={1}
  />
</Graphic>
```



```
<script>
  import { Graphic, Section, PolygonLayer, createGeoScales } from '@snlab/florence'
  import DataContainer from '@snlab/florence-datacontainer'
  import { scaleOrdinal } from 'd3-scale'
  import { schemeCategory10 } from 'd3-scale-chromatic'
  import { provincesGeoJSON } from './provinces.js'

  const provinces = new DataContainer(provincesGeoJSON)
  const geoScales = createGeoScales(provinces.bbox())

  const colorScale = scaleOrdinal()
    .domain(provinces.domain('statcode'))
    .range(schemeCategory10)
</script>

<Graphic width={500} height={500} {...geoScales} flipY>
  <PolygonLayer
    geometry={provinces.column('$geometry')}
    fill={provinces.map('statcode', colorScale)}
    stroke={'white'}
    strokeWidth={1}
  />
</Graphic>
```

Figure 8. *A map of Dutch provinces. Dutch spatial data is often provided in a country-specific projection and coordinate system ("Rijksdriehoeksstelsel"), which isn't compatible with most JavaScript mapping libraries that rely solely on WGS84. The bottom panel shows a categorical colour scheme applied to the province name. Interactive version available at florence.spatialnetworkslab.org/examples/cp-figure8a and florence.spatialnetworkslab.org/examples/cp-figure8b.*

implementation using the grammar of graphics (i.e., combine `Section`s with different marks).

INTERACTION

Although higher levels (a "grammar," if you will) of abstraction for web mapping interactions exist (Roth 2013; Roth et al. 2014), we have chosen to rely on a slightly lower-level abstraction that is consistent with native browser event listeners for both desktop (mouse) and mobile (touch) events (DG1, DG4). We use this approach so that knowledge gained with HTML/JavaScript will transfer easily to Florence and vice versa. We consider this a useful trade-off for geovisualizations because they often need to include interactions with both visual elements (e.g., click on a map element) in addition to more "conventional" page elements (e.g., clicking on a button). The same event-listener approach can be used for both types of elements.

Florence uses an R-tree based spatial index (**github.com/mourner/rbush**) for detecting "hits" in an efficient manner that scales up to large datasets. Listeners for different user events can be set on both `Graphic`s and `Section`s as well as on individual marks. With these basic building blocks, any of the common geovisualization interactions (e.g., pan, zoom, highlight, brush, select, linked views, etc.) can be achieved. Importantly, Florence provides useful information about the mark being interacted with, including its identifier and its location in both "data space" and "pixel space" (see Figure 9 for an example).

SPECIFIC USE CASES AND EXAMPLES

Florence eases the execution of many common tasks in cartography through its flexible combination of `Section`s and marks. For example, map insets—often used to show an overview or different parts of non-contiguous countries, can be created by simply creating a separate `Section` for each inset and giving that section its own scale/bounding box (and thus its own coordinate system). This is difficult to achieve with web mapping libraries like Leaflet, and requires the use of a composite, custom projection in D3[5]. A common scenario is to display the contiguous United States, with Alaska and Hawaii as separate insets. Each would have their own, appropriate projections and bounding boxes. To achieve this with Florence, GeoJSON data for all states can be filtered into three separate `DataContainers` (one for the contiguous United States, one for Alaska, and one for Hawaii) and

5. e.g., github.com/d3/d3-geo/blob/master/README.md#geoAlbersUsa

```
<script>
  import { Graphic, Section, PolygonLayer, createGeoScales } from '@snlab/florence'
  import DataContainer from '@snlab/florence-datacontainer'
  import { provincesGeoJSON } from './provinces.js'

  const provinces = new DataContainer(provincesGeoJSON)
  const geoScales = createGeoScales(provinces.bbox())

  let selectedIndex = null
</script>

<Graphic width={500} height={500} {...geoScales} flipY>
  <PolygonLayer
    geometry={provinces.column('$geometry')}
    fill={({ index }) => index === selectedIndex ? 'yellow' : 'rgb(230,230,230)'}
    stroke={'white'}
    strokeWidth={1}
    onMouseover={e => { selectedIndex = e.index }}
    onMouseout={e => { selectedIndex = null }}
  />
</Graphic>

<h1>
  {selectedIndex ? provinces.row({ index: selectedIndex }).statnaam : 'No province selected'}
</h1>
```


Figure 9. *An example of a hover-based interaction. When the user hovers over a province, the province lights up in yellow, and its name is displayed beneath the map. Interactive version available at florence.spatialnetworkslab.org/examples/cp-figure9.*

each `DataContainer` is then used to set up a `Section` with its own projection and scaling.

Similarly, small multiples—a grid of smaller maps, each showing a different variable (e.g., one for each year in a dataset)—can be achieved in an automated fashion by using Svelte's `{#each}` syntax to repeat a separate `Section` for each variable. In the code example below, `years` is an array of numbers that represent the years in a dataset. Through Svelte's slot property syntax (`let:cells`), the `Grid` component makes an object called `cells` available to all components inside of it. This object contains the x and y coordinates for each "cell" or `Section` so they can be automatically arranged into a grid formation.

```
<Grid names={years} let:cells>
    {#each years as year}
        <Section {...cells[year]} {...geoScales}>
            <PolygonLayer
                geometry={data.column('$geometry')}
                fill={data.map(year, someScale)}
            />
        </Section>
    {/each}
</Grid>
```

Similar logic can also be applied to create, for example, atlas-like functionality, in which a map is created for each province in a dataset.

One approach somewhat unique to cartography is the visualization of multi-dimensional spatial datasets through small pie charts or other such "micro diagrams" (Gröbe and Burghardt 2020) that are displayed at specific locations on a map to visualize some additional information about that specific location. Depending on the complexity of the type of diagram, these can be challenging to implement with web mapping software. However, with the grammar-of-graphics approach, we can think of each micro diagram as an individual `Section` (with its own coordinate system) that we can simply position at the right geographic coordinates. An example of this, replicating Mathieu Rajerison's approach (Rajerison 2020) for "map sparklines" can be seen in Figure 10.

Figure 10. *"Map sparklines" as an example of micro diagrams. Used here to show the evolution of COVID-19 cases in different Dutch provinces in an animated manner. Code and interactive version available at florence.spatialnetworkslab.org/examples/cp-figure10.*

TEACHING WITH FLORENCE

In the spring of 2020, we used the framework as a core library to teach an introductory course in interactive data visualization at the Singapore University of Technology and Design. The course had no specific prerequisites and attracted students from a wide variety of backgrounds. Most students had no significant programming experience and only three students had worked with HTML before. None had prior training in cartography. We include a short discussion of our experience teaching with Florence here as an initial pilot study of the potential effectiveness of our approach, pending a more formal and systematic assessment (see Discussion & Future Work).

The first half of the course built a foundational understanding of HTML/JS/CSS and the Svelte reactive framework, by recreating charts produced by Du Bois and his colleagues for the 1900 Paris Exhibition (Battle-Baptiste and Rusert 2018) using each of those technologies. The second

half of the course introduced the grammar of graphics and its implementation with the Florence framework. Students then created a series of visualization dashboards and interactive maps during class exercises, while simultaneously working on an independent project.

The main challenge for students in the course was to learn the foundational computing concepts within HTML/JS/CSS, as well as the Svelte framework. After that, picking up the grammar of graphics, and by extension the Florence library, seemed natural to students. From a pedagogical point of view, it is interesting to note that many students did not fully realize that they were actually using an external software library. Rather, they were just writing JavaScript based on the core concepts from the grammar of graphics, such as marks and scales.

As highlighted before, Florence serves as a convenience layer on top of core web technologies. This enabled students to branch out creatively in their final projects, combining and linking different visual ways to analyse and present their data, using everything from maps and graphs to regular UI elements such as form elements and text (DG5, DG6). In some cases, students built relatively bespoke and complex web applications, in which the use of Florence was observed to be helpful in easing the path to linking to "low-level" approaches, which can otherwise be challenging to achieve with libraries that provide more out-of-the-box, "one-click" solutions (DG3, DG4).

Importantly, Florence's easy interoperability with other libraries (DG7) allowed student projects to merge the grammar-of-graphics foundation taught in the class with libraries such as *d3-force* to display network data and *mapbox-gl* to display a pannable basemap under a Florence visualization. This was also useful for students who came to the course with the expectation of learning D3 or some other existing library and were initially disappointed to learn Florence instead. The foundation of the grammar of graphics allowed them to quickly adopt other libraries and approaches in their final projects. While the library in its current state is not without its limitations, we are encouraged by this initial use case, which shows clear promise as a teaching tool for web-based cartography and data visualization.

DISCUSSION & FUTURE WORK

With a flexible combination of the core components discussed here, many (spatial) visualizations can be created. Importantly, once the grammar-of-graphics approach is adopted, a user can employ the concepts to "think through" a visualization, breaking it down in its constituent marks and scales even before starting the actual coding process. In our experience teaching with Florence, the easy transition from HTML to the use of Svelte and Florence—as well as the declarative approach to writing visualizations—works well for students that do not come from a software engineering background. By design, Florence does not have much embedded "magic" and, in some cases, requires relatively verbose code (DG4). We argue that this should not be seen as a downside as it leads to greater understanding and easier customization and adaptation in student projects.

The modular, component-based approach aids in this flexibility as well. Although the framework only provides a limited set of primitive marks, they can be easily expanded (DG3, DG7). For example, a box plot is an example of a visual element that is not a single mark but rather a collection of different marks that indicate the different quartiles and outliers. This collection of marks can be turned into its own higher-level "boxplot" component and can subsequently be re-used across a project and shared with other users or projects. In this way, higher-order layers can be created—to the point of entire pre-defined maps that can serve as templates. As an example, the sparklines seen in Figure 10 can be saved as a component as well—allowing the user to create one for any country by passing a reference, via the component properties, to a GeoJSON file for the spatial polygons as well as a table of x/y data for the actual sparklines.

Although our initial use of the library in teaching showed promise, a more thorough evaluation in an educational context is warranted. Such an evaluation could take two specific approaches. First, the extent to which the design goals are achieved, and the library's impact on a student's learning of web-based cartography skills and concepts can be formally assessed in subsequent iterations of the course. Roth and Sack's (2017) methodology provides a clear and structured evaluation approach for this purpose through

employing instructor observation logs, student feedback compositions, and exit surveys. However, by its very nature, such an approach evaluates the library only in the context of the course in which it is used.

To provide a more direct comparison to other commonly used libraries in web mapping (e.g., D3, Leaflet), a survey with an experimental design could be conducted with cartography practitioners that have a working understanding of web technology but might use varying tools in their day-to-day practice. A few common mapping scenarios could be implemented, supplemented with small lessons, across different technologies to measure the effectiveness of those technologies in relation to the aforementioned design goals. Since both the research population and likely sample size will be small, such a survey could be combined with qualitative exit interviews as well.

There are some obvious limitations in the implementation of the first version of the library as well. For example, currently Florence only supports rendering in SVG. However, its rendering backend is written to allow for different rendering approaches. For larger datasets, SVG has certain limitations. In future work, we would like to explore expansion to both HTML canvas and WebGL rendering. The latter is an especially promising technology for creating geovisualizations of very large datasets. Although some more general WebGL visualization libraries exist (Ren, Lee, and Höllerer 2017), to the best of our knowledge no convenient approach currently exists for creative cartography with WebGL.

Similarly, it would be fruitful to build on our current implementation of interactions to provide a higher-level "grammar" of interactions (cf. Roth 2013). In relation to this, Florence currently does allow for a basic set of transitions and animations, including tweening. Animation has been a long standing interest in cartography (Karl 1992; Lobben 2003), but recent work has called for caution around the use of animation to facilitate change detection in choropleth maps (Fish, Goldsberry, and Battersby 2011). We believe extending the grammar to interactions and animations (cf. the R library *gganimate*; gganimate.com), and thus easing its use, will enable a wider variety of use cases for animation in web mapping, beyond the common case of mapping temporal change to frames in an animation.

In evaluating its approach and current capabilities, we put Florence forward as an example of software designed for web-based data visualization that is speaking directly to the discipline of cartography, and cartography education in particular. We are optimistic that such approaches and a concerted effort around developing software for cartography have the potential to not only open up new ways of creative mapmaking but also help address the significant challenges in teaching web-based mapping in our cartography curricula.

DATA & CODE AVAILABILITY

The source code repository for the software can be found on Gitlab: **gitlab.com/spatialnetworkslab/florence**. A deeper treatment, including documentation and more elaborate examples, can be found at the documentation website: **florence.spatialnetworkslab.org**.

FUNDING DETAILS

This work was supported by the Singapore Ministry of Education Tertiary Education Research Fund, under Grant 3 TR 20.

REFERENCES

Anselin, Luc, Yong Wook Kim, and Ibnu Syabri. 2004. "Web-Based Analytical Tools for the Exploration of Spatial Data." *Journal of Geographical Systems* 6 (2): 197–218. https://doi.org/10.1007/s10109-004-0132-5.

Battersby, Sarah E., Michael P. Finn, E. Lynn Usery, and Kristina H. Yamamoto. 2014. "Implications of Web Mercator and Its Use in Online Mapping." *Cartographica* 49 (2): 85–101. https://doi.org/10.3138/carto.49.2.2313.

Battle-Baptiste, Whitney, and Britt Rusert, eds. 2018. *W. E. B. Du Bois's Data Portraits: Visualizing Black America*. Amherst, MA: Princeton Architectural Press.

Bertin, Jacques. 2010. *Semiology of Graphics: Diagrams, Networks, Maps.* Redlands, CA: Esri Press.

Boettiger, Carl, Scott Chamberlain, Edmund Hart, and Karthik Ram. 2015. "Building Software, Building Community: Lessons from the ROpenSci Project." *Journal of Open Research Software* 3 (1): e8. http://doi.org/10.5334/jors.bu.

Bostock, Michael, and Jeffrey Heer. 2009. "Protovis: A Graphical Toolkit for Visualization." *IEEE Transactions on Visualization and Computer Graphics* 15 (6): 1121–1128. https://doi.org/10.1109/TVCG.2009.174.

Bostock, Michael, Vadim Ogievetsky, and Jeffrey Heer. 2011. "D3 Data-Driven Documents." *IEEE Transactions on Visualization and Computer Graphics* 17 (12): 2301–2309. https://doi.org/10.1109/TVCG.2011.185.

Chang, Winston, Joe Cheng, J. Allaire, Yihui Xie, and Jonathan McPherson. 2019. "Shiny: Web Application Framework for R." *R Package Version* 1 (5). https://cran.r-project.org/package=shiny.

Crampton, Jeremy W. 2010. *Mapping: A Critical Introduction to Cartography and GIS, 1st Edition*. Malden, MA: Wiley-Blackwell.

Degbelo, Auriol, Saad Sarfraz, and Christian Kray. 2020. "Data Scale as Cartography: A Semi-Automatic Approach for Thematic Web Map Creation." *Cartography and Geographic Information Science* 47 (2): 153–170. https://doi.org/10.1080/15230406.2019.1677176.

Dent, Borden D. 2009. *Cartography: Thematic Map Design*. New York: McGraw-Hill Higher Education.

Fabrikant, Sara Irina, Cedric Gabathuler, and André Skupin. 2015. "SOMViz: Web-Based Self-Organizing Maps." *KN - Journal of Cartography and Geographic Information* 65 (2): 81–91. https://doi.org/10.1007/BF03545092.

Fish, Carolyn, Kirk P. Goldsberry, and Sarah Battersby. 2011. "Change Blindness in Animated Choropleth Maps: An Empirical Study." *Cartography and Geographic Information Science* 38 (4): 350–362. https://doi.org/10.1559/15230406384350.

Gahegan, Mark. 2018. "Our GIS Is Too Small." *The Canadian Geographer* 62 (1): 15–26. https://doi.org/10.1111/cag.12434.

Gahegan, Mark, Masahiro Takatsuka, Mike Wheeler, and Frank Hardisty. 2002. "Introducing GeoVISTA Studio: An Integrated Suite of Visualization and Computational Methods for Exploration and Knowledge Construction in Geography." *Computers, Environment and Urban Systems* 26 (4): 267–92. https://doi.org/10.1016/S0198-9715(01)00046-1.

Griffin, Amy L., Anthony C. Robinson, and Robert E. Roth. 2017. "Envisioning the Future of Cartographic Research." *International Journal of Cartography* 3 (sup1): 1–8. https://doi.org/10.1080/23729333.2017.1316466.

Gröbe, Mathias, and Dirk Burghardt. 2020. "Micro Diagrams: Visualization of Categorical Point Data from Location-Based Social Media." *Cartography and Geographic Information Science* 47 (4): 305–320. https://doi.org/10.1080/15230406.2020.1733438.

Hardisty, Frank, and Anthony C. Robinson. 2011. "The Geoviz Toolkit: Using Component-Oriented Coordination Methods for Geographic Visualization and Analysis." *International Journal of Geographical Information Science* 25 (2): 191–210. **https://doi.org/10.1080/13658810903214203**.

Harris, Richard, David O'Sullivan, Mark Gahegan, Martin Charlton, Lex Comber, Paul Longley, Chris Brunsdon, Nick Malleson, Alison Heppenstall, Alex Singleton, et al. 2017. "More Bark than Bytes? Reflections on 21+ Years of Geocomputation." *Environment and Planning B: Urban Analytics and City Science* 44 (4): 598–617. **https://doi.org/10.1177/2399808317710132**.

Harvey, Francis. 2019. "Jacques Bertin's Legacy and Continuing Impact for Cartography." *Cartography and Geographic Information Science* 46 (2): 97–99. **https://doi.org/10.1080/15230406.2019.1533784**.

Heer, Jeffrey, and Michael Bostock. 2010. "Declarative Language Design for Interactive Visualization." *IEEE Transactions on Visualization and Computer Graphics* 16 (6): 1149–1156. **https://doi.org/10.1109/TVCG.2010.144**.

Kahle, David, and Hadley Wickham. 2013. "ggmap: Spatial Visualization with ggplot2." *The R Journal* 5 (1): 144–161.

Karl, Doris. 1992. "Cartographic Animation: Potential and Research Issues." *Cartographic Perspectives* 13: 3–9. **https://doi.org/10.14714/CP13.999**.

Kraak, Menno-Jan, and Ferjan Ormeling. 2011. *Cartography, Third Edition: Visualization of Spatial Data*. New York: Guilford Press.

Ledermann, Florian, and Georg Gartner. 2015. "mapmap.js: A Data-Driven Web Mapping API for Thematic Cartography" *Revista Brasileira de Cartografia* 67 (5): 1043–1054. **http://www.seer.ufu.br/index.php/revistabrasileiracartografia/article/view/44626**.

Liu, Zhicheng, John Thompson, Alan Wilson, Mira Dontcheva, James Delorey, Sam Grigg, Bernard Kerr, and John Stasko. 2018. "Data Illustrator: Augmenting Vector Design Tools with Lazy Data Binding for Expressive Visualization Authoring." In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems,* 1–13. New York: ACM Press. **https://doi.org/10.1145/3173574.3173697**.

Lobben, Amy. 2003. "Classification and Application of Cartographic Animation." *The Professional Geographer* 55 (3): 318–328. **https://www.tandfonline.com/doi/abs/10.1111/0033-0124.5503016**.

MacEachren, Alan M. 2004. *How Maps Work: Representation, Visualization, and Design*. New York: Guilford Press.

———. 2013. "Cartography as an Academic Field: A Lost Opportunity or a New Beginning?" *The Cartographic Journal* 50 (2): 166–170. **https://doi.org/10.1179/0008704113Z.00000000083**.

Muehlenhaus, Ian. 2014. "Going Viral: The Look of Online Persuasive Maps." *Cartographica* 49 (1): 18–34. **https://doi.org/10.3138/carto.49.1.1830**.

Munzner, Tamara. 2014. *Visualization Analysis and Design*. Boca Raton, FL: CRC Press.

Nagel, Till, Joris Klerkx, Andrew Vande Moere, and Erik Duval. 2013. "Unfolding – A Library for Interactive Maps." In *Human Factors in Computing and Informatics*, edited by Andreas Holzinger, Martina Ziefle, Martin Hitz, and Matjaž Debevc, 497–513. Berlin, Heidelberg: Springer. **https://doi.org/10.1007/978-3-642-39062-3_31**.

Nost, Eric, Heather Rosenfeld, Kristen Vincent, Sarah A. Moore, and Robert E. Roth. 2017. "HazMatMapper: An Online and Interactive Geographic Visualization Tool for Exploring Transnational Flows of Hazardous Waste and Environmental Justice." *Journal of Maps* 13 (1): 14–23. **https://doi.org/10.1080/17445647.2017.1282384**.

Pezanowski, Scott, Alan M. MacEachren, Alexander Savelyev, and Anthony C. Robinson. 2018. "SensePlace3: A Geovisual Framework to Analyze Place–Time–Attribute Information in Social Media." *Cartography and Geographic Information Science* 45 (5): 420–437. **https://doi.org/10.1080/15230406.2017.1370391**.

Poorthuis, Ate, and Matthew Zook. 2020. "Being Smarter about Space: Drawing Lessons from Spatial Science." *Annals of the American Association of Geographers* 110 (2): 349–359. **https://doi.org/10.1080/24694452.2019.1674630**.

Rajerison, Mathieu. 2020. "A Sparkline Map of COVID-19 (Or any Name you'll Prefer)." Billet. *Datagistips*. 2020. **https://datagistips.hypotheses.org/488**.

Ren, Donghao, Bongshin Lee, and Tobias Höllerer. 2017. "Stardust: Accessible and Transparent GPU Support for Information Visualization Rendering." *Computer Graphics Forum* 36 (3): 179–188. **https://doi.org/10.1111/cgf.13178**.

Ricker, Britta, and Jim Thatcher. 2017. "Evolving Technology, Shifting Expectations: Cultivating Pedagogy for a Rapidly Changing GIS Landscape." *Journal of Geography in Higher Education* 41 (3): 368–382. **https://doi.org/10.1080/03098265.2017.1315533**.

Robinson, Anthony C. 2019. "Elements of Viral Cartography." *Cartography and Geographic Information Science* 46 (4): 293–310. **https://doi.org/10.1080/15230406.2018.1484304**.

Roth, Robert E. 2013. "An Empirically-Derived Taxonomy of Interaction Primitives for Interactive Cartography and Geovisualization." *IEEE Transactions on Visualization and Computer Graphics* 19 (12): 2356–2365. **https://doi.org/10.1109/TVCG.2013.130**.

———. 2017. "User Interface and User Experience (UI/UX) Design." *Geographic Information Science & Technology Body of Knowledge*. **https://doi.org/10.22224/gistbok/2017.2.5**.

Roth, Robert E., Richard G. Donohue, Carl M. Sack, Timothy R. Wallace, and Tanya M. A. Buckingham. 2014. "A Process for Keeping Pace with Evolving Web Mapping Technologies." *Cartographic Perspectives* 78: 25–52. **https://doi.org/10.14714/CP78.1273**.

Roth, Robert E., and Alan M. MacEachren. 2016. "Geovisual Analytics and the Science of Interaction: An Empirical Interaction Study." *Cartography and Geographic Information Science* 43 (1): 30–54. **https://doi.org/10.1080/15230406.2015.1021714**.

Roth, Robert E., Kevin S. Ross, and Alan M. MacEachren. 2015. "User-Centered Design for Interactive Maps: A Case Study in Crime Analysis." *ISPRS International Journal of Geo-Information* 4 (1): 262–301. **https://doi.org/10.3390/ijgi4010262**.

Sack, Carl M. 2018. "The Status of Web Mapping in North American Higher Education." *Cartographic Perspectives* 89: 25–43. **https://doi.org/10.14714/CP89.1429**.

Sack, Carl M., and Robert E. Roth. 2017. "Design and Evaluation of an Open Web Platform Cartography Lab Curriculum." *Journal of Geography in Higher Education* 41 (1): 1–23. **https://doi.org/10.1080/03098265.2016.1241987**.

Satyanarayan, Arvind, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. "Vega-Lite: A Grammar of Interactive Graphics." *IEEE Transactions on Visualization and Computer Graphics* 23 (1): 341–350. **https://doi.org/10.1109/TVCG.2016.2599030**.

Satyanarayan, Arvind, Ryan Russell, Jane Hoffswell, and Jeffrey Heer. 2016. "Reactive Vega: A Streaming Dataflow Architecture for Declarative Interactive Visualization." *IEEE Transactions on Visualization and Computer Graphics* 22 (1): 659–668. **https://doi.org/10.1109/TVCG.2015.2467091**.

Šavrič, Bojan, Bernhard Jenny, and Helen Jenny. 2016. "Projection Wizard – An Online Map Projection Selection Tool." *The Cartographic Journal* 53 (2): 177–185. **https://doi.org/10.1080/00087041.2015.1131938**.

Shannon, Jerry, and Kyle E. Walker. 2020. "Ventures into Viral Cartography: Waffle House, Educational Attainment, and the Social Life of Maps." *The Professional Geographer* 72 (1): 66–77. **https://doi.org/10.1080/00330124.2019.1653774**.

Slocum, Terry A., Robert B. McMaster, Fritz C. Kessler, and Hugh H. Howard. 2009. *Thematic Cartography and Geovisualization, Third Edition*. Upper Saddle River, NJ: Pearson Prentice Hall.

Tennekes, Martijn. 2018. "Tmap: Thematic Maps in R." *Journal of Statistical Software* 84 (1): 1–39. **https://doi.org/10.18637/jss.v084.i06**.

Tobler, Waldo R. 1959. "Automation and Cartography." *Geographical Review* 49 (4): 526–534. **https://doi.org/10.2307/212211**.

Wickham, Hadley. 2010. "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics* 19 (1): 3–28. **https://doi.org/10.1198/jcgs.2009.07098**.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy McGowan, Romain François, Garrett Grolemund, Alex Hayes, Lionel Henry, Jim Hester, et al. 2019. "Welcome to the Tidyverse." *Journal of Open Source Software* 4 (43): 1686. **https://doi.org/10.21105/joss.01686**.

Wickham, Hadley, Romain Francois, Lionel Henry, and Kirill Müller. 2015. "Dplyr: A Grammar of Data Manipulation." *R Package Version 0.4* 3. **https://cran.r-project.org/package=dplyr**.

Wilkinson, Leland. 2013. *The Grammar of Graphics*. Berlin: Springer Berlin Heidelberg.