DOI: 10.14714/CP81.1332

Mapping Temporal Datasets with D3

Patrick Butler
University of Nebraska at Omaha
pjbutler@unomaha.edu

INTRODUCTION

Many JavaScript libraries and APIs intended for mapping require long lines of code to perform even the simplest forms of animation. The Data-Driven Documents (D3) software library, however, offers a very simple option for quickly presenting a series of maps. D3's library contains various features that allow for geographic data to be bound to SVG objects in a webpage and presented chronologically. Animations can often be created with one short line of code.

ANIMATING WITH D3

D3 was developed primarily by Mike Bostock at Stanford University in 2011 with the intention of "bringing data to life." It offers a set of JavaScript functions for data visualization, which can be accessed by loading the *d3.js* library into an HTML document. These functions allow for the visualization of data sets by binding them to SVG objects and displaying them in a web browser. SVG is an XML-based format that allows for vector graphics to be grouped, styled, and transformed. After the data are tied to these objects, they may be animated using D3's *transition* method.

The *transition* method is a special type of selector. Selectors allow objects in a webpage to be selected and then manipulated. Objects may be selected based upon their properties, such as tags, classes, attributes, or unique identifiers. Once a selection is made, operators may then be applied. These operators may manipulate an object's attributes, styles, or text content. They may also join data to the selected objects. By itself, an ordinary selector applies the subsequent operations instantaneously. By using the *transition* method, instead, the changes will occur gradually over time as opposed to immediately. When multiple transitions are

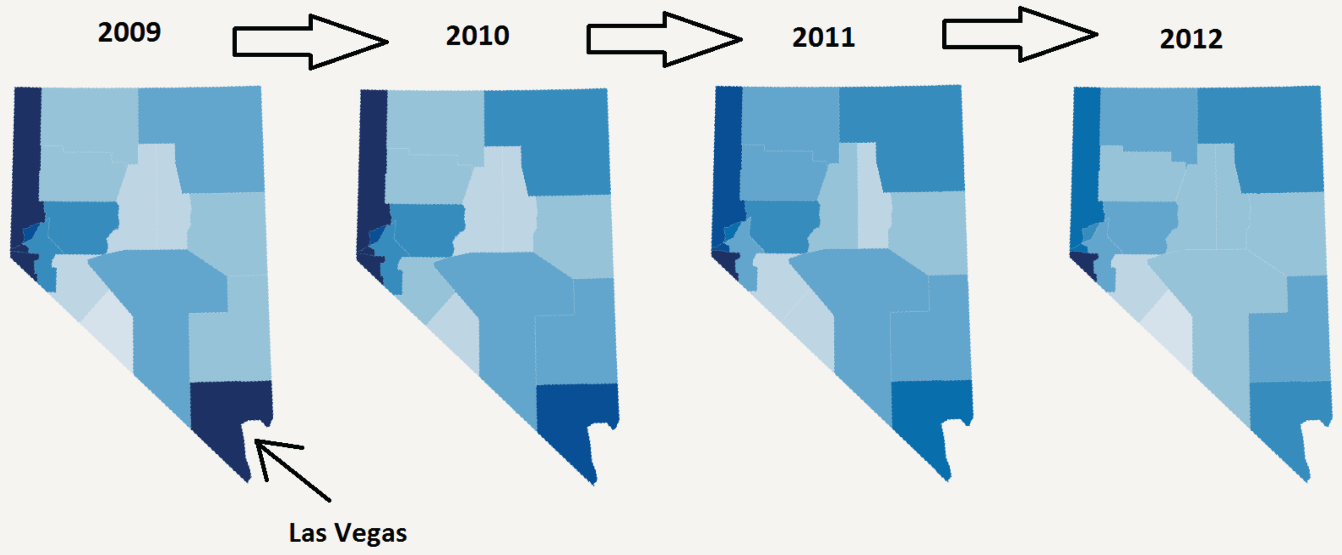

Figure 1. Nevada median home values from 2009 to 2012.

applied, the *delay* method may then be used to add separation between each transition. The *delay* constructor specifies, in milliseconds, how long the transition should wait to begin. By pairing each transition with a delay, the transformation may be specifically timed to string multiple maps together into a smooth animation (see Figure 1).

D3's user-friendly, minimalist approach allows for the easy borrowing of code. There are hundreds of examples of D3 maps on websites such as GitHub, and any example from the Web may be manipulated for use with another dataset with very little effort. Existing code examples demonstrate how to use D3 to take raw datasets and bind them to graphic elements for display in a browser (Figure 2). To add even more functionality, parts of various existing scripts can be combined.

D3 is very efficient, often requiring less code than other software libraries to accomplish the same task. The selection methods, for example, allow for elements of a webpage to be selected and manipulated either individually or all at once using a single line of code, whereas other libraries often require a *for* loop just to select an element and change one attribute. In addition to these efficiencies, D3 also offers options such as shape generators, scale constructors, and a variety of map projections.

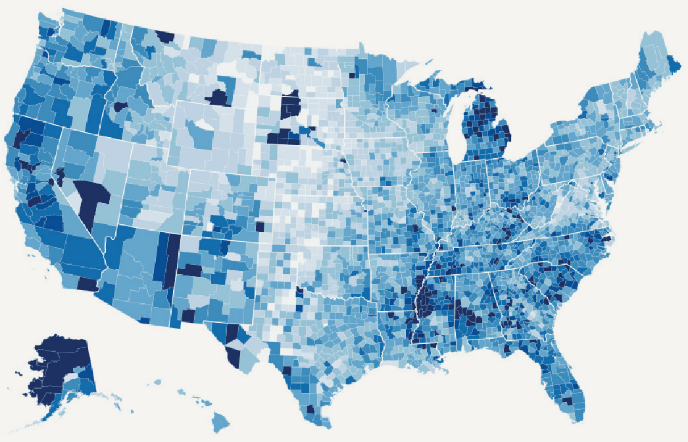

Figure 2. *Mike Bostock's choropleth map of US unemployment rates for 2008 demonstrates the binding of data to graphic elements using D3.*

DATA SOURCES

Data may be loaded into a webpage via D3 in multiple formats. D3 can accept GeoJSON and TopoJSON data, which store the coordinates of geographic features, and are variants on the JSON (JavaScript Object Notation) format, which uses simple text to pass attribute data in pairs. JSON files offer a simple way to organize and store data into variables and load them quickly in the background of a webpage. GeoJSON and TopoJSON can also include non-geographic attributes for each feature. Attribute data may also be stored separate from geographic data, in a comma-separated values (CSV) or tab-separated values (TSV) file. In Example 1, used to create Figure 2, a JSON file is loaded in which contains the outlines of US counties.

The United States Census Bureau's American FactFinder is a great resource for finding datasets with a temporal component, with thousands of different categories of statistical data at various levels of spatial resolution. Once downloaded, some manipulation is necessary for the data to be properly read by the specific code in Example 1: it is important to rename the FIPS code field to "id" and the statistic field title to "rate." The data can be saved as a tab-separated file using the extension ".tsv."

MAPPING AND ANIMATION

Once the data files are in the correct format, they may be bound to an SVG element and visualized using D3. SVG uses the *path* variable to bind it to the county geometry defined in the JSON file and draw the counties in a webpage. D3 then uses what it calls a *dictionary* to relate the unemployment values in the TSV file to their respective counties. After data classification using the *quantize* function, D3 uses inline CSS, a computer language used for manipulating the presentation of a webpage, to color each class (see Example 1). CSS offers many advantages to web design because it separates the content of a webpage from its styling.

The animation process begins by drawing the first map. After this is completed, the map may be redrawn multiple times and staggered with the delay function. This can be

```
//Creating choropleth map of U.S. unemployment rates

                                                        //Define fill colors for 9 possible classes
.q0-9 { fill:rgb(247,251,255); } .q1-9 { fill:rgb(222,235,247); } .q2-9 { fill:rgb(198,219,239); }
.q3-9 { fill:rgb(158,202,225); } .q4-9 { fill:rgb(107,174,214); } .q5-9 { fill:rgb(66,146,198); }
.q6-9 { fill:rgb(33,113,181); } .q7-9 { fill:rgb(8,81,156); } .q8-9 { fill:rgb(8,48,107); }
</style>
<body>
<script src="http://d3js.org/d3.v3.min.js"></script>    //Load D3 libraries
<script src="http://d3js.org/queue.v1.min.js"></script>
<script src="http://d3js.org/topojson.v1.min.js"></script>      //Imports external JavaScript file of
                                                                //reusable functions
<script>
  var width = 960, height = 600;                                        //Set size of map in pixels

  var rateById = d3.map();              //Create dictionary to relate counties in JSON to respective values in TSV

  var quantize = d3.scale.quantize()                    //Determine 9 possible classes
    .domain([0, .15])                                   //Set domain with maximum and minimum values of dataset
    .range(d3.range(9).map(function(i) { return "q" + i + "-9"; }));

  var projection = d3.geo.albersUsa()           //Set projection to Alber's Equal Area Conic
    .scale(1280)                                //Preserves area proportionally between two parallels
    .translate([width / 2, height / 2]);        //Ideal for large areas running east-west, such as U.S.

  var path = d3.geo.path()                      //Create path generator, for JSON geometries to be drawn
    .projection(projection);

  var svg = d3.select("body").append("svg")                     //Select body of DOM, append to SVG element
    .attr("width", width)
    .attr("height", height);

  queue()                                               //Load files, wait for ready function
    .defer(d3.json, "/mbostock/raw/4090846/us.json")
    .defer(d3.tsv, "unemployment.tsv", function(d) { rateById.set(d.id, +d.rate); })
    .await(ready);                                      //Populate dictionary with id and rate fields from TSV

  function ready(error, us) {                           //Call JSON file, bind to SVG
    svg.append("g")
    .attr("class", "counties")
    .selectAll("path")
    .data(topojson.feature(us, us.objects.counties).features)                   //Convert JSON to GeoJSON
    .enter().append("path")                                                     //Bind county data to path
    .attr("class", function(d) { return quantize(rateById.get(d.id)); })        //Determine each county's class
    .attr("d", path);                                                           //Draw counties

    svg.append("path")
    .datum(topojson.mesh(us, us.objects.states, function(a, b) { return a !== b; }))
    .attr("class", "states")                    //Mesh state boundaries to prevent duplicate paths for borders
    .attr("d", path); }                         //Draw states

  d3.select(self.frameElement).style("height", height + "px");
</script>
```

Example 1. *Code for Mike Bostock's U.S. unemployment map.*

done by creating an update function that includes some of the same code from Example 1, but with a different year's dataset (see Example 2). The *transition* method creates a pause between the drawings of the two maps. The standard transition time is 250 milliseconds. By using the *delay* method after each transition, that time can be lengthened, adjusting the speed of the animation.

The addition of the *transition* and *delay* components to each update create the animation. The data update function may be repeated for the number of time periods present in the time series. It is important to include the minimum and maximum values that would be appropriate for the entire series of datasets in the original *quantize* function, so that each set is classified in the same way. The delay time must also be incremented in equal intervals to form a consistent animation.

The animated map still needs to be paired with a dynamic title. To do this, the *span* HTML element can be used (see Example 3). *Span*, short for spanning, is similar to the *div* element, as both are used for organizing and styling particular pieces of a webpage. The *div*, or division, element is typically for larger areas of a webpage, and may be made up of many different spans. The *span* element is for smaller areas of text. Both allow for specific parts of an HTML page to be grouped together and easily referred to later in the document. By adding a span to the *h2* element,

```
//Creating update function

function updateData() {

  queue()
  .defer(d3.json, "us.js")
  .defer(d3.tsv, "unemployment2.tsv", function(d) { rateById.set(d.id, +d.rate); })
  .await(ready);                    //Update script with new dataset

function ready(error, us) {

  svg.append("g")
  .attr("class", "counties")
  .selectAll("path")
  .data(topojson.feature(us, us.objects.counties).features)
  .enter().append("path")
    .transition()                          //Create a brief pause between redrawing of map
    .delay(750)                            //Extend pause to 3/4 of a second, increase by
                                           //equal intervals each update

  .attr("class", function(d) { return quantize(rateById.get(d.id)); })
  .attr("d", path);

  svg.append("path")
  .datum(topojson.mesh(us, us.objects.states, function(a, b) { return a !== b; }))

  .attr("class", "states")
  .attr("d", path);

  updateData();                     //Add to each ready function preceding an update function

}
```

Example 2. *By creating an update function that includes a transition and delay constructor, the map may be continually redrawn using a new dataset.*

```
<h2>
    <span> </span> //Add to beginning of body
</h2>

d3.select("h2 span").text("U.S. Unemployment Rates 2010"); //Add to end of ready function
```

Example 3. *By adding a span to the h2 element, the space may be selected in each update function and a dynamic title may be inserted.*

which indicates header text, a new title may be inserted each time the update function is called. In order to create the same animated effect displayed by the map itself, the transition and delay parameters must be added to each title as well. By using the same delay timings from each update function, the titles stagger at the same intervals as their respective maps.

CONCLUSION

D3 NOT ONLY transforms raw datasets into static graphics, but also graphics with movement and interactivity. By using D3's animation capabilities, a sense of change over time may be conveyed. Applying a custom dataset to one of the many D3 map examples on the web is easy; an update function can then be assembled to reload the existing script with new data. The *transition* method is used to redraw the map after the update, and the *delay* operator adds timing to the animation to generate a smooth progression of graphics. The library is simple enough to use so that anyone with a basic understanding of HTML and JavaScript can easily turn almost any time series dataset into an animated map.

A sample animated map, based upon this article, can be viewed at **http://pjbutler.podserver.info/test.html**.

SUGGESTED RESOURCES

Bostock, Mike. 2012. "Choropleth." *Bl.ocks.org*. **http://bl.ocks.org/mbostock/4060606**.

Maps on D3 — Tutorial. 2013. *Social Innovation Simulation*. **http://socialinnovationsimulation.com/2013/07/11/tutorial-making-maps-on-d3**.