Chapter 2. Introducing D3

D3—also referred to as D^3 or d3.js—is a JavaScript library for creating data visualizations. But that kind of undersells it.

The abbreviation D3 references the tool's full name, *Data-Driven Documents*. The *data* is provided by you, and the *documents* are web-based documents, meaning anything that can be rendered by a web browser, such as HTML and SVG. D3 does the *driving*, in the sense that it connects the data to the documents.

Of course, the name also functions as a clever allusion to the network of technologies underlying the tool itself: the W3, or World Wide Web, or, today, simply "the Web."

D3's primary author is the brilliant Mike Bostock, although there are a few other dedicated contributors. The project is entirely open source and freely available on GitHub.

D3 is released under a BSD license, so you may use, modify, and adapt the code for noncommercial or commercial use at no cost.

D3's official home on the Web is *d3js.org*.

What It Does

Fundamentally, D3 is an elegant piece of software that facilitates generation and manipulation of web documents with data. It does this by:

- *Loading* data into the browser's memory
- *Binding* data to elements within the document, creating new elements as needed
- *Transforming* those elements by interpreting each element's bound datum and setting its visual properties accordingly

- *Transitioning* elements between states in response to user input

Learning to use D3 is simply a process of learning the syntax used to tell it how you want it to load and bind data, and transform and transition elements.

The *transformation* step is most important, as this is where the *mapping* happens. D3 provides a structure for applying these transformations, but, as we'll see, you define the mapping rules. Should larger values make taller bars or brighter circles? Will clusters be sorted on the x-axis by age or category? What color palette is used to fill in countries on your world map? All of the visual design decisions are up to you. You provide the concept, you craft the rules, and D3 executes it—without telling you what to do. (Yes, it's like the opposite of Excel's pushy "Chart Wizard.")

What It Doesn't Do

Here is a list of things D3 does not do:

- D3 doesn't generate predefined or "canned" visualizations for you. This is on purpose. D3 is intended primarily for *explanatory* visualization work, as opposed to *exploratory* visualizations. Exploratory tools help you discover significant, meaningful patterns in data. These are tools like Tableau and ggplot2, which help you quickly generate multiple views on the same data set. That's an essential step, but different from generating an *explanatory* presentation of the data, a view of the data that highlights what you've already discovered. Explanatory views are more constrained and limited, but also focused, and designed to communicate only the important points. D3 excels at this latter step, but is not ideal for the former. (For ideas on other tools, see the section "Alternatives" later in this chapter.)
- D3 doesn't even try to support older browsers. This helps keep the D3 codebase clean and free of hacks to support old versions of Internet

Explorer, for example. The philosophy is that by creating more compelling tools and refusing to support older browsers, we encourage more people to upgrade (rather than forestall the process, thereby requiring us to continue to support those browsers, and so on—a vicious cycle). D3 wants us to move forward.

- D3's core functionality doesn't handle bitmap map tiles, such as those provided by Google Maps or Cloudmade. D3 is great with anything vector—SVG images or GeoJSON data—but wasn't originally intended to work with traditional map tiles. (*Bitmap* images are made up of pixels, so resizing them larger or smaller is difficult without a loss in quality. *Vector* images are defined by points, lines, and curves —mathematical equations, really—and can be scaled up or down without a loss in quality.) This is starting to change, with the introduction of the d3.geo.tile plug-in. Prior to this plug-in, geomapping with D3 meant either going all-SVG and avoiding tiles or using D3 to create SVG visuals *on top of* a base layer of map tiles (which would be managed by another library, like Leaflet or Polymaps—see the section "Alternatives" later in this chapter). This question of how to integrate bitmap tiles and vector graphics comes up a lot in the D3 community. As of today, there is no super-simple and perfect answer, but I think you can expect to see lots of work done in this area, and possibly the new tile-handling methods integrated into the D3 core at some point in the future.
- D3 doesn't hide your original data. Because D3 code is executed on the client side (meaning, in the user's web browser, as opposed to on the web server), the data you want visualized must be sent to the client. If your data can't be shared, then don't use D3. Alternatives include using proprietary tools (like Flash) or prerendering visualizations as static images and sending those to the browser. (If you're not interested in sharing your data, though, why would you bother visualizing it? The purpose of visualization is to communicate the data, so you might sleep better at night by choosing openness and transparency, rather than having nightmares about data thieves.)

Origins and Context

The first web browsers rendered static pages; interactivity was limited to clicking links. In 1996, Netscape introduced the first browser with JavaScript, a new scripting language that could be interpreted *by the browser while the page was being viewed*.

This doesn't sound as groundbreaking as it turned out to be, but this enabled web browsers to evolve from merely passive *browsers* to dynamic frames for interactive, networked experiences. This shift ultimately enabled every intrapage interaction we have on the Web today. Without JavaScript, D3 would never exist, and web-based data visualizations would be limited to prerendered, noninteractive GIFs. (Yuck. Thank you, Netscape!)

Jump ahead to 2005, when Jeffrey Heer, Stuart Card, and James Landay introduced prefuse, a toolkit for bringing data visualization to the Web. prefuse (spelled with all lowercase letters) was written in Java, a compiled language, with programs that could run in web browsers via a Java plug-in. (Note that *Java* is a completely different programming language than *JavaScript*, despite their similar names.)

prefuse was a breakthrough application—the first to make web-based visualization accessible to less-than-expert programmers. Until prefuse came along, any datavis on the Web was very much a custom affair.

Two years later, Jeff Heer introduced Flare, a similar toolkit, but written in ActionScript, so its visualizations could be viewed on the Web through Adobe's Flash Player. Flare, like prefuse, relied on a browser plug-in. Flare was a huge improvement, but as web browsers continued to evolve, it was clear that visualizations could be created with native browser technology, no plug-ins required.

By 2009, Jeff Heer had moved to Stanford, where he was advising a graduate student named Michael Bostock. Together, in Stanford's Vis

Group, they created Protovis, a JavaScript-based visualization toolkit that relied exclusively on native browser technologies. (If you have used Protovis, be sure to reference Mike's introduction to D3 for Protovis users.)

Protovis made generating visualizations simple, even for users without prior programming experience. Yet to achieve this, it created an abstract representation layer. The designer could address this layer using Protovis syntax, but it wasn't accessible through standard methods, so debugging was difficult.

In 2011, Mike Bostock, Vadim Ogievetsky, and Jeff Heer officially announced D3, the next evolution in web visualization tools. Unlike Protovis, D3 operates directly on the web document itself. This means easier debugging, easier experimentation, and more visual possibilities. The only downside to this approach is a potentially steeper learning curve, but this book will make that as painless as possible. Plus, all the skills you gain while learning about D3 will prove useful even beyond the realm of datavis.

If you're familiar with any of these groundbreaking tools, you'll appreciate that D3 descends from a prestigious lineage. And if you have any interest in the philosophy underlying D3's elegant technical design, I highly recommend Mike, Vadim, and Jeff's InfoVis paper, which clearly articulates the need for this kind of tool. The paper encapsulates years' worth of learning and insights made while developing visualization tools.

Alternatives

D3 might not be perfect for every project. Sometimes you just need a quick chart and you don't have time to code it from scratch. Or you might need to support older browsers and can't rely on recent technologies like SVG.

For those situations, it's good to know what other tools are out there. Here is a brief, noncomprehensive list of D3 alternatives, all of which use web-standard technologies (mostly JavaScript) and are free to download and use.

Easy Charts

DataWrapper
: A beautiful web service that lets you upload your data and quickly generate a chart that you can republish elsewhere or embed on your site. This service was originally intended for journalists, but it is helpful for everyone. DataWrapper displays interactive charts in current browsers and static images for old ones. (Brilliant!) You can also download all the code and run it on your own server instead of using theirs.

Flot
: A plotting library for jQuery that uses the HTML canvas element and supports older browsers, even all the way back to Internet Explorer 6. It supports limited visual forms (lines, points, bars, areas), but it is easy to use.

Google Chart Tools
: Having evolved from their earlier Image Charts API, Google's Chart Tools can be used to generate several standard chart types, with support for old versions of IE.

gRaphaël
: A charting library based on Raphaël (see later in this chapter) that supports older browsers, including IE6. It has more visual flexibility than Flot, and—some might say—it is prettier.

Highcharts JS
: A JavaScript-based charting library with several predesigned themes and chart types. It uses SVG for modern browsers and falls back on VML for old versions of IE, including IE6 and later. The tool is free only for noncommercial use.

JavaScript InfoVis Toolkit
: The JIT provides several preset visualization styles for your data. It

includes lots of examples, but the documentation is pretty technical. The toolkit is great if you like one of the preset styles, but browser support is unclear.

jqPlot
: A plug-in for charting with jQuery. This supports very simple charts and is great if you are okay with the predefined styles. jqPlot supports IE7 and newer.

jQuery Sparklines
: A jQuery plug-in for generating sparklines, typically small bar, line, or area charts used inline with text. Supports most browsers, even back to IE6.

Peity
: A jQuery plug-in for very simple and very *tiny* bar, line, and pie charts that supports only recent browsers. Did I mention that this makes only very *tiny* visualizations? +10 cuteness points.

Timeline.js
: A library specifically for generating interactive timelines. No coding is required; just use the code generator. There is not much room for customization, but hey, timelines are really hard to do well. Timeline.js supports only IE8 and newer.

YUI Charts
: The Charts module for the Yahoo! User Interface Library enables creation of simple charts with a goal of wide browser support.

Graph Visualizations

A "graph" is just data with a networked structure (for example, B is connected to A, and A is connected to C).

Arbor.js
: A library for graph visualization using jQuery. Even if you never use this, you should check out how the documentation is presented as a graph, using the tool itself. (It's so *meta*.) It uses the HTML canvas, so it works only in IE9 or current browsers, although some workarounds

are available.

Sigma.js
: A very lightweight library for graph visualization. You have to visit this website, move your mouse over the header graphic, and then play with the demos. Sigma.js is beautiful and fast, and it also uses canvas.

Geomapping

I distinguish between *mapping* (all visualizations are maps) and *geomapping* (visualizations that include geographic data, or geodata, such as traditional maps). D3 has a lot of geomapping functionality, but you should know about these other tools.

Kartograph
: A JavaScript-and-Python combo for gorgeous, entirely vector-based mapping by Gregor Aisch with must-see demos. Please go look at them now. I promise you've never seen online maps this beautiful. Kartograph works with IE7 and newer.

Leaflet
: A library for tiled maps, designed for smooth interaction on both desktop and mobile devices. It includes some support for displaying data layers of SVG on top

of the map tiles. (See Mike's demo "Using D3 with Leaflet".) Leaflet works with IE6 (barely) or IE7 (better!) and of course all current browsers.

Modest Maps
: The granddaddy of tiled map libraries, Modest Maps has been succeeded by Polymaps, but lots of people still love it, as it is lightweight and works with old versions of IE and other browsers. Modest Maps has been adapted for ActionScript, Processing, Python, PHP, Cinder, openFrameworks…yeah, basically everything. File this under "oldie, but goodie."

Polymaps

A library for displaying tiled maps, with layers of data on top of the tiles. Polymaps relies on SVG and thus works best with current browsers.

Almost from Scratch

These tools, like D3, provide methods of drawing visual forms, but without predesigned visual templates. If you enjoy the creative freedom of starting from scratch, you might enjoy these.

Processing.js
: A native JavaScript implementation of Processing, the fantastic programming language for artists and designers new to programming. Processing is written in Java, so exporting Processing sketches to the Web traditionally involved clunky Java applets. Thanks to Processing.js, regular Processing code can run natively, in the browser. It renders using canvas, so only modern browsers are supported.

Paper.js
: A framework for rendering vector graphics to canvas. Also, its website is one of the most beautiful on the Internet, and their demos are unbelievable. (Go play with them now.)

Raphaël
: Another library for drawing vector graphics, popular due to its friendly syntax and support for older browsers.

Three-Dimensional

D3 is not the best at 3D, simply because web browsers are historically two-dimensional beasts. But with increased support for WebGL, there are now more opportunities for 3D web experiences.

PhiloGL
: A WebGL framework specifically for 3D visualization.

Three.js
: A library for generating any sort of 3D scene you could imagine, produced by Google's Data Arts team. You could spend all day exploring the mind-blowing demos on their site.

Tools Built with D3

When you want to use D3 without actually writing any D3 code, you can choose one of the many tools built on top of D3!

Crossfilter
: A library for working with large, multivariate datasets, written primarily by Mike Bostock. This is useful for trying to squeeze your "big data" into a relatively small web browser.

Cubism
: A D3 plug-in for visualizing time series data, also written by Mike Bostock. (One of my favorite demos.)

Dashku
: An online tool for data dashboards and widgets updated in real time, by Paul Jensen.

dc.js
: The "dc" is short for *dimensional charting*, as this library is optimized for exploring large, multidimensional datasets.

NVD3
: Reusable charts with D3. NVD3 offers lots of beautiful examples, with room for visual customizations without requiring as much code as D3 alone.

Polychart.js
: More reusable charts, with a range of chart types available. Polychart.js is free only for noncommercial use.

Rickshaw
: A toolkit for displaying time series data that is also very customizable.

Tributary
: A great tool for experimenting with live coding using D3, by Ian

Johnson.