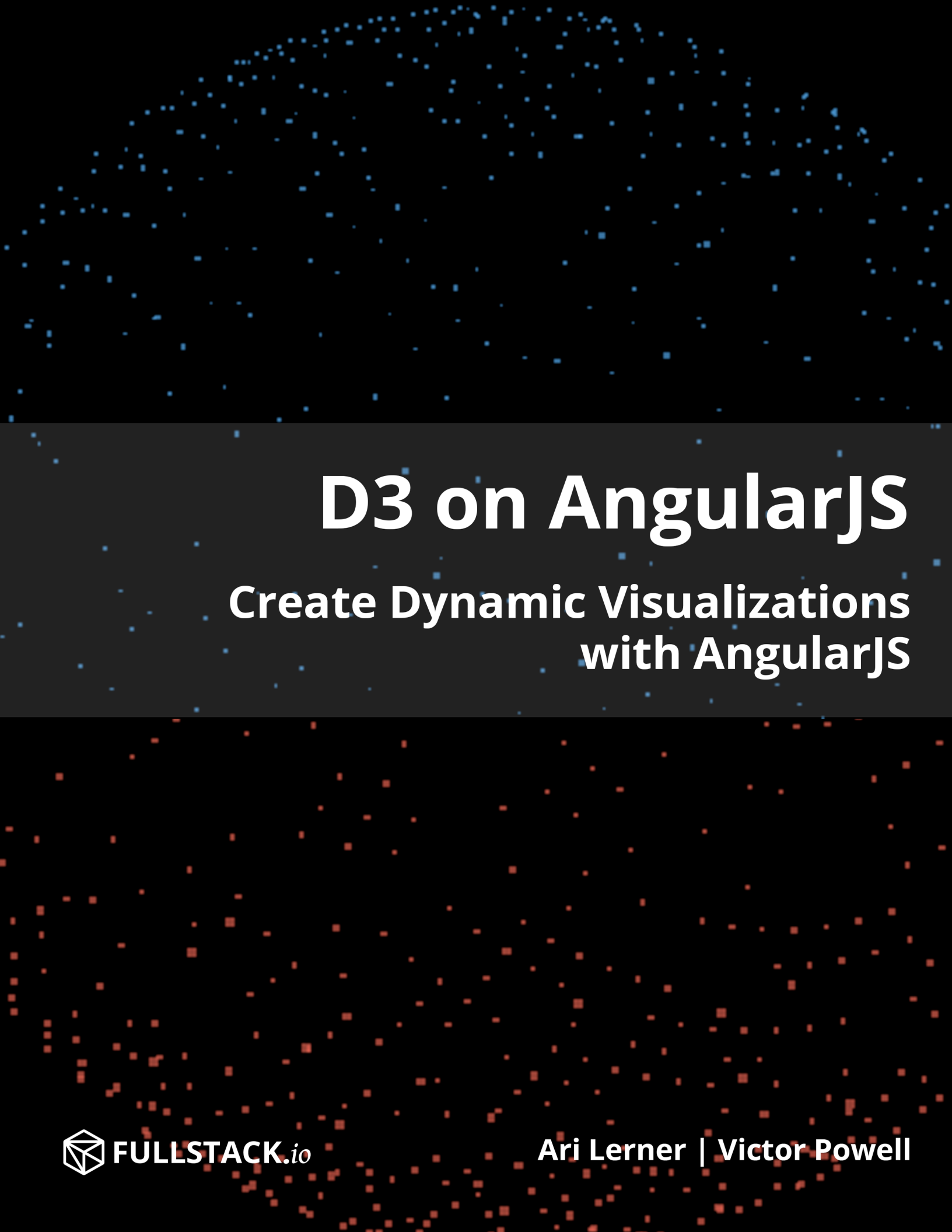


D3 on AngularJS

Create Dynamic Visualizations with AngularJS

FULLSTACK.io

Ari Lerner | Victor Powell

D3 on AngularJS

Create Dynamic Visualizations with AngularJS

Ari Lerner and Victor Powell

This book is for sale at http://leanpub.com/d3angularjs

This version was published on 2014-06-06

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

©2013 - 2014 Ari Lerner and Victor Powell

Tweet This Book!

Please help Ari Lerner and Victor Powell by spreading the word about this book on Twitter!

The suggested hashtag for this book is #d3angular.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search?q=#d3angular

Contents

Introducing D3. A simple example

In this chapter, we'll go over what D3 is and what makes it such a powerful tool for data visualization. We'll also introduce a simple 'Hello World' style example that shows how to get quickly get setup and running with D3.

What is it?

D3 (or Data-Driven Documents) is a library written by Mike Bostock[1] for "manipulating documents based on data." This means D3's primary job is to take data and produce structured documents such as HTML or SVG with respect to data. Unlike most visualization libraries, D3 is not a ready-made collection of common graphs and widgets. It's common to use D3 to make common graphs, like bar charts and pie charts, but the real power is in its flexibility and fine-grain control over the final result.

D3 works well with other established web technologies like CSS and SVG because it doesn't attempt to abstract away the DOM, like many other graphing libraries. This also means D3 will continue to be useful as browsers incorporate new features.

If we're just looking for a particular graph type like, say, a bar chart, and don't care how exact it ends up looking, D3 might not be the right library for the job. Several other ready-made libraries exist for creating simple, cookie-cutter charts, such as HighCharts[2] or Chart.js[3] or Google Charts API[4]. If, on the other hand, we have strong requirements for how our visualization should look and function, D3 is a great choice.

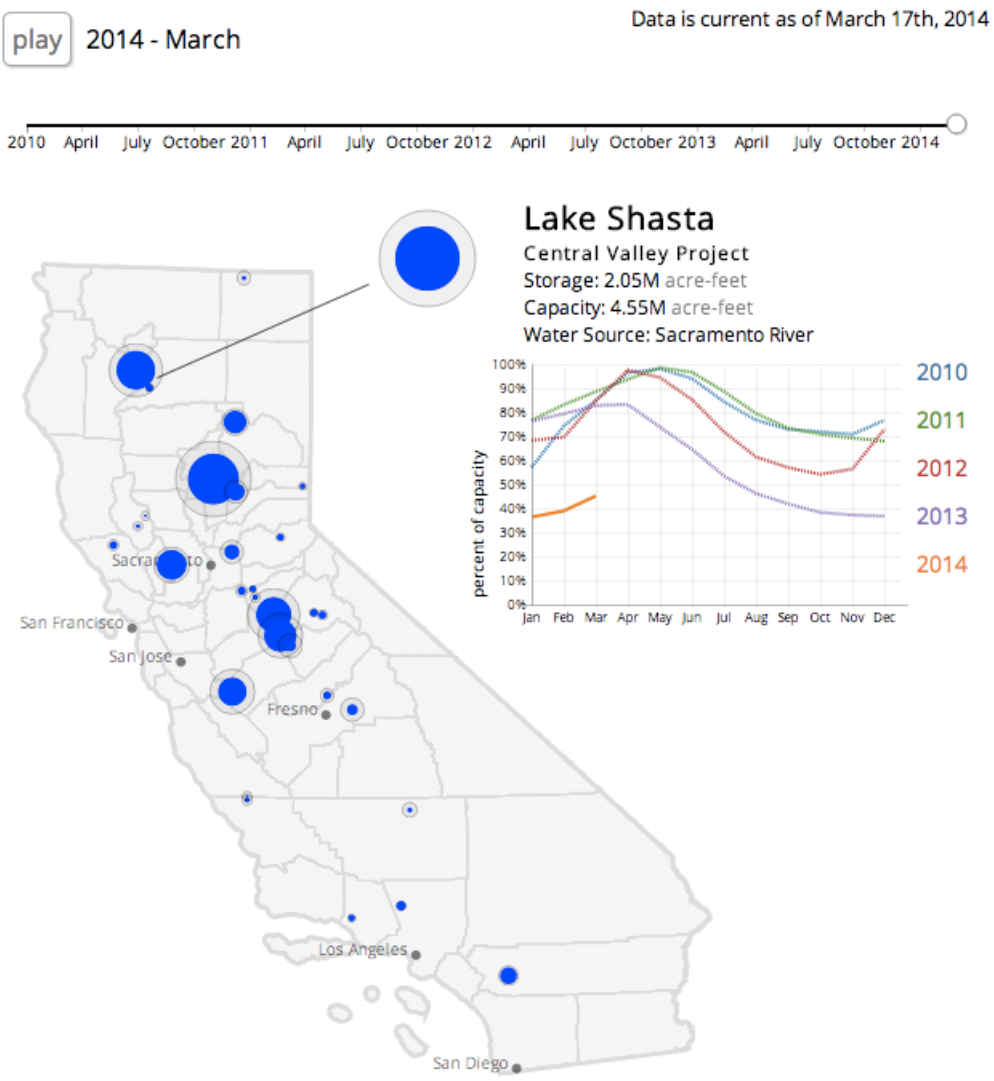
To quickly jump into a real world example of this, take the following interactive visualization produced by KQED's blog The Lowdown. It doesn't fall into a single-chart category and the different chart components need to communicate with each other, updating dynamically.

[1] http://bost.ocks.org/mike/

[2] http://www.highcharts.com/

[3] http://www.chartjs.org/

[4] https://developers.google.com/chart/



Live version of the above interactive visualization[5]

It was created using a combination of Angular and D3. (source code is available here[6]).

To illustrate this point, we'll also walk though a little thought experiment. Imagine we're working with a ready-made visualization library. Typical visualization libraries might have a `BarChart` class to create a new bar chart which works fine until we want to do something the library didn't allow to be configured. For example, say we wanted to change the background color of the legend in our bar chart. We could take their code and try and modify it to add our needed feature, but that can quickly

[5] http://blogs.kqed.org/lowdown/2014/03/18/into-the-drought-californias-shrinking-reservoirs/

[6] https://github.com/vicapow/water-supply

get very messy and cumbersome. We might have to create a new subclass of the `BarChart` or it could be that the original BarChart class wasn't written in a way to be easily extensible. Alternatively, we can use D3 to quickly, and in a only few lines, create our own custom bar chart were we can do whatever we'd like to our legend or any other component.

As another example, say we wanted to create a new type of visualization that doesn't even exist yet (or at least not yet in JavaScript.) This is a perfect example of when we would want to use D3. In this sense, D3 is a sort of "meta-library"; the kind of library one would want to have if they were creating a library of new data visualizations. It does this by using a new way of thinking about data visualizations (but more on that later.) In short, if we're going to be creating data visualizations, we'll typically be writing an order of magnitude less code if we use D3 than without.

When it comes to configuring the look and feel of our visualization, we can very easily utilize our existing knowledge of CSS, so long as we use classes when creating the components that make up our bar chart. Continuing our thought experiment, something along these lines would be enough to change the background color of our legend.

```
.graph .legend{
  color: white;
}
```

D3's functional, declarative style permits us to write less code. Less code allows us to make changes faster and reduces the cognitive load required to remember all the code we've written. When code is shorter, we can remember it better and read it faster later.

Consider the following example that changes all `<circle>` nodes in an `<svg>` to be positioned horizontally occurring to the `data` array.

Don't worry if this is confusing right now. We'll walk through how this works in later chapters. This sample is only to demonstrate D3's brevity.

Using straight JavaScript, we'll need to select the `<svg>` element and select all of the `<circle>` elements (assuming they are on the DOM already) and modify their `cx` attribute, like so:

```
var data = [ 10, 20, 30, 40];
var svg = document
  .getElementsByTagName('svg')
  .item(0);
var circles = svg.getElementsByTagName('circle');
for(var i = 0; i < circles.length; i++){
  var circle = circles.item(i);
  circle.setAttribute('cx', data[i]);
}
```

Using D3 allows us to accomplish the same using less code:

```
d3.select('svg').selectAll('circle')
  .data([10, 20, 30, 40])
  .attr('cx', function(d){ return d; });
```

'Hello World' D3 style

To dive right in, below is a simple 'Hello World' style D3 example which simply appends an `<h1>` with the text `Hello World!` to the `<body>` using D3.

```
<!DOCTYPE html>
<html>
  <head>
    <script src="http://d3js.org/d3.v3.min.js" charset="utf-8"></script>
  </head>
  <body>
    <script>
      d3.select('body')
        .append('h1')
        .text('Hello World!');
    </script>
  </body>
</html>
```

Live version: http://jsbin.com/uhEmuJI/1/edit[7]

We just created our first D3 app.

Although this example does not do very much, it highlights the structure we'll build with our D3 apps; the foundation for most of the examples that follow.

The resulting HTML after our D3 code has executed looks like this.

[7] http://jsbin.com/uhEmuJI/1/edit

```
<html>
  <head>...</head>
  <body>
    <script>...</script>
    <h1>Hello World!</h1>
  </body>
</html>
```

We can inspect the source of the resulting HTML using the fantastic Chrome developer tools[8] `Elements` tab. We'll use these tools throughout the book to deeply inspect our running code.

Common gotcha

When not using Angular or jQuery, make sure to put any code that depends on the `<body>` inside the `<body>` and *not* before it (ie., in the `<head>`.) If the code executes before the body element, then it will simply fail as the `<body>` will not have been created yet.

[8] https://developers.google.com/chrome-developer-tools/