Getting Started with D3

d3 interaction
Feb 2014

Visualizations are great tools to understand ideas and play with data. As Tim Brown from Ideo writes in this post

> *Visual thinking isn't limited to illustrations. It can take many forms. Mind maps, two-by-two matrices, and other visual frameworks can help explore and describe ideas in valuable ways that require little more than a few straight lines and some imagination.*

Especially for visualizing data in graphs, D3 might be helpful. D3 stands for "Data-Driven Documents" and is a JavaScript visualization library for HTML and SVG. This post is a short overview to help you get started.

Loading D3

The easiest way to load D3 is by loading the script from a CDN, such as:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/d3/3.4.1/d3.min.js"></script>
```

Alternatively, we can copy the library from the github repository or the project website.

By adding a script reference to the library, we obtain a "d3" object to work with.

The d3 object

The d3 object is somewhat similar to the "$" object in jQuery. This means, that we can "select" DOM nodes, and we can build nodes with "append". For drawing, we need an "svg" based canvas. Adding this "svg" is the first step to build a graph. Therefore, we define the following construct:

```
var vis = d3.select("#graph")
            .append("svg");
```

We can add attributes such as width and height of the graph with:

```
     var w = 900,
         h = 400;
     vis.attr("width", w)
        .attr("height", h);
```

We can also add text with:

```
     vis.text("Our Graph")
        .select("#graph")
```

This should look pretty familiar if you have worked with selectors in jQuery.

Placing nodes

To render data, we need to "join" data with DOM nodes. This is done with the data() command. The mapping of data to nodes can feel a bit magical, since this adds relationships based on a declarative syntax. If you end up somewhat confused (as I was), you can read some additional explanation here

To make a first mapping, we first define nodes:

```
 var nodes = [{x: 30, y: 50},
                {x: 50, y: 80},
                {x: 90, y: 120}]
```

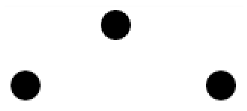
Since we start with a fresh canvas, all nodes will be new, and we can map these with “selectAll” - “data” - “enter”:

```
    vis.selectAll("circle.nodes")
       .data(nodes)
       .enter()
       .append("svg:circle")
       .attr("cx", function(d) { return d.x; })
       .attr("cy", function(d) { return d.y; })
```

And to actually see the circles, we must set the fill attribute:

```
    vis.selectAll("circle.nodes")
        .data(nodes)
        .enter()
        .append("svg:circle")
        .attr("cx", function(d) { return d.x; })
        .attr("cy", function(d) { return d.y; })
        .attr("r", "10px")
        .attr("fill", "black")
```

This should result into something like:



Since the circles are setup, we now can add and remove data, and the graph will update automatically: Try this codepen

Connecting the Dots

To render a graph, we need lines between the circles. Since we have the coordinates of the circles, let’s define the line data structure with:

```
  var links = [
    {source: nodes[0], target: nodes[1]},
    {source: nodes[2], target: nodes[1]}
  ]
```

We can use the line SVG shape for connecting the dots:

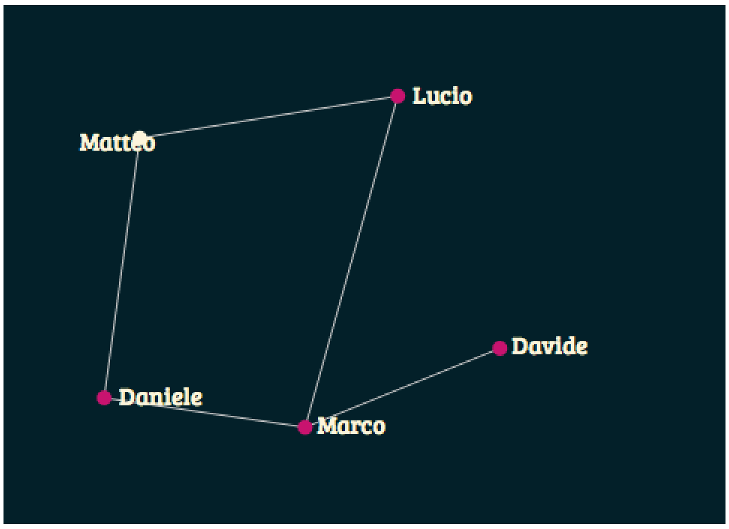
```
  vis.selectAll(".line")
     .data(links)
     .enter()
     .append("line")
     .attr("x1", function(d) { return d.source.x })
     .attr("y1", function(d) { return d.source.y })
     .attr("x2", function(d) { return d.target.x })
     .attr("y2", function(d) { return d.target.y })
     .style("stroke", "rgb(6,120,155)");
```

We have functions to translate the coordinates to line attributes. The result can be seen in this codepen To see some of the mechanics, you can add and remove data, and see how the graph changes.

Exploring further

Setting up graphs manually with nodes and edges might be interesting for small examples. However, if you often need to setup graphs, or the number of nodes and edges increases, a graph can be be setup with an algorithm too. This is where force layouts help.

With this, you can setup a graph like in this example:



D3 also supports force layout algorithms, and a nice place to start is here. Another option might be using D3 plugins, such as the graph plugin.

What are your experiences with D3 for graphs and the different approaches? Leave feedback here, or at (hackernews)[https://news.ycombinator.com/item?id=7210162].

Resources

If you want to explore further, here are some interesting links:

- D3 Tutorials by Scott Murray
- Fine tuning the display of circles: A question on SO
- An overview on basic shapes
- Another tutorial flow visualization
- SO question on using a force layout

And some graph examples with D3:

- Graph of Mobile Patent Suits
- State Diagram Editor
- Simple directed graph
- Discussion of force layout
- Drawing Graphs with D3

Leave me feedback

comments powered by Disqus

Follow me on Twitter here.

Tweet 79

g+1 6