TUTORIAL: D3 (3)

Advanced

Christoph Kralj
christoph.kralj@univie.ac.at

Manfred Klaffenböck
manfred.klaffenboeck@univie.ac.at

Prerequisites

Previously you learned about how to use SVGs within HTML documents to produce visual outputs, how to select and manipulate elements via JavaScript and D3, and how to load and prepare data. We also discussed scales, axes, and the enter-update-exit pattern. You will need this knowledge to participate in this tutorial.

Previous tutorials:

http://vda.univie.ac.at/Teaching/Vis/17w/Tutorials/d3_tutorial1_17w.pdf

http://vda.univie.ac.at/Teaching/Vis/17w/Tutorials/d3_tutorial2_17w.pdf

Contents

Mouse Events

Often we want more interactivity than just manipulating our visualization via HTML GUI elements. For this D3 provides the *.on(type, listener)* function, which can be called on selections. The *type* is a string specifying which kind of event is listened for. Examples would be "click", "mouseover", "keydown". The second argument, the *listener*, is an anonymous function defining what to do when an event is caught. It gets passed the current datum, *d*, and its index, *i*. Moreover, you can access the element on which the event was triggered with *d3.select(this)* within the anonymous function.

```
d3.selectAll("circle").on("click", function(d) {
        console.log(d);
 });
```

More about the *.on()* method can be found here: https://github.com/d3/d3-selection/blob/master/README.md#handling-events

A list of possible event types can be found here: https://developer.mozilla.org/en-US/docs/Web/Events#Standard_events

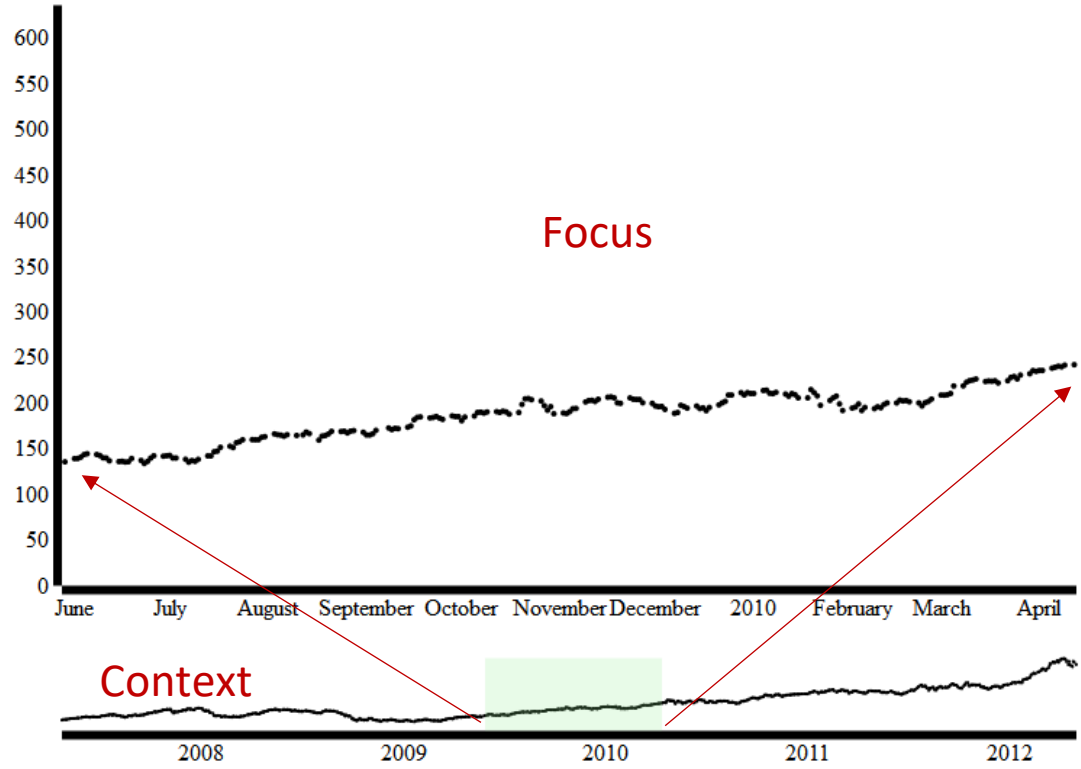
---

Exercise 3.1

- Get the template code http://vda.univie.ac.at/Teaching/Vis/17w/Tutorials/t3/exercise3_template.html
- Download the data http://vda.univie.ac.at/Teaching/Vis/17w/Tutorials/data/sp500.csv
- Make yourself familiar with the code
- Make the points change their color when clicked.

Linking & Brushing

In more complex visualizations it is often required that the user is able to select a subset of the data – *brushing* – and these selections to be reflected in other views – *linking*. The case where there is one visualization acting as a detail view, and one as an overview, is often called Focus+Context.



Brushing is conveniently implemented in D3 via the *brush* component. This creates an element that works similar to an axis and allows the user to select subsets of the data with their mouse.

```
var brush = d3.brushX()
    .extent([[0, 0], [width, height]])
    .on("brush", brushed);
```

The brush can react to three events, again caught with the *.on()* method:

1. "*start*" is triggered on mousedown,
2. "*brush*" on mousemove after start, and
3. "*end*" on mouseup.

In the code example *brushed* is the name of a function implemented elsewhere in your code.

After the user triggered an event, the function passed to the .on() method is called. What is usually done here is that the domain of the focus view's x-axis is adapted to fit the domain selected by the user vie the brush. For this purpose, D3 provides the brush.extent() method that returns the selected region.

```
function brushed() {
        var selection = d3.event.selection;
        x1.domain(selection.map(x2.invert, x2));
        mainWindow.selectAll(".dot")
         .attr("cx", function(d) { return x1(d.date); })
         .attr("cy", function(d) { return y1(d.price); });
        mainWindow.select(".x-axis").call(xAxis1);
}
```

This example adapts the domain of the scale used for the x-axis of the focus plot to the selected region of the brush. If the user made no selection, it sets it to the domain to be equal to the one used by the context visualization – containing all data points.

Now the brush has to be added to an SVG or SVG group, just like an axis. Additionally, we add a rectangle, indicating the user's selection.

```
myGroup.append("g")
        .call(brush)
        .selectAll("rect")
        .attr("y", yOffset)
        .attr("height", height)
        .style("fill","lightgreen")
        .style("fill-opacity", ".2");
```

More about brushing here: https://github.com/d3/d3-brush

Exercise 3.2

- Reuse your previous code
- Implement a brush for the x axis. (Hint: You can place it in the space below the main visualization). The result should look like the image above. Remember, for the brush you need :
    - A group containing:
        - A brush component
        - An extra x-axis (like in the focus view)
        - An extra y-scale
        - A plot of the data-elements (like in the focus view)
        - Call the brush and the axis on their own a SVG groups
    - A function handling what happens once a brush event is triggered
        - Adapting the focus-scale
        - Redrawing the elements shown in the focus view

CSS

Often you want to set the style of your components for multiple parts of your visualization project at once. For this purpose you can use CSS (Cascading Stylesheets). These stylesheets can either be internal (in the head of the html document) or external (additional documents with a .css file extension).

We will discuss the external case in the next section, right now we will focus at the internal case.

In order to create an internal stylesheet, you have to place style-tags within between the head-tags of you document.

```
<body>
      …
      <style>
            …
            /* Your CSS-definitions here */
            …
```

```
    </style>
    …
<body>
```

In a stylesheet, you can define style properties via the selectors discussed in the first tutorial – "elementName", ".elementClass", "#elementID".

For instance, instead of calling .attr on our svg-variable to set "width" and "height", we set those in the head via CSS:

CSS:

```
svg {
  width: 1000px;
  height: 630px;
}
```

JS:

```
var svg = d3.select("body").append("svg");
```

Employing this strategy can help immensely to make your code more readable!

A good reference for CSS can be found here: http://www.w3schools.com/css/

Exercise 3.3

- Remove some styles in your file and replace them with a CSS style sheet.

Breaking up your code

For bigger projects, it is very helpful for better overview to put css-styles and javascript-code into individual files (with .css and .js extensions, respectively) and load them into your html-file.

To load a .css file, you use

```
<link rel="stylesheet" type="text/css" href="myStyle.css">
```

where "myStyle.css" is replaced by the name of your css-file. This call usually goes into the head-section of your html-document. Please note, that the external .css-file doesn't require the surrounding style-tags.

To load a .js file, you use

```
<script src="myCode.js"></script>
```

where "myCode.js" is replaced by the name of your js-file. This line can either go into the head- or the body-section of your html-document, where the general recommendation is though to place it at the end of your body-tags. Please note, that the external .js-file doesn't require the surrounding script-tags.

Here is an example of how the loading in the html-document could look like:

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="myStyle.css">
  <script src="./../src/d3.js"></script>
</head>
<body>
  <script src="myCode.js"></script>
</body>
</html>
```

Exercise 3.4

- Put your css-styles into a .css-file
- Put your js-code within the script tags into a .js-file (without the script-tags)
- Load those external files into your html-file.

Acknowledgements

Many thanks to Michael Opperman for all the help and material he provided.