Seminar Javascript technology 2015-2

Vol. 00 no. 00 2015
Pages 1–5

Data visualization using D3

Jesús Alejandro Valdés Valdés, and Philipp Müller

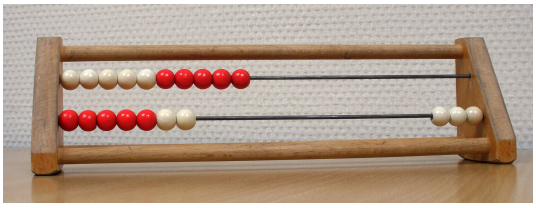

Fig. 1. Simple abacus

ABSTRACT

Motivation

Long before people started to collect data in a digital format, they depended on visual representations of data. One of the earliest devices to visualize the abstract concepts of the numerical system and basic arithmatics was the Abacus (figure 1). Numbers are represented by small spheres on a wire, shifting them from left to right you can add or subtract values and count the remaining balls. This simple visualization of the numerical system and basic arithmatics is still used today in pre-schools and elementary schools, because visualizations convey information in an universal manner and make it simple to share and communicate ideas with others.

As huge amounts of data are collected nowadays, data visualization software becomes more and more important. A visual representation enables us to find relevance of millions of variables, communicate concepts and even predict future behaviour based on patterns.

In our report we will discuss different visualization methods for data in a web based context, available JavaScript libraries, why we decided to use D3.js, and explain the core concepts of D3.

Availability

D3.js is a free JavaScript library which can also be customized according to your own needs. Further information how to download or link it can be found on their official website d3js.org.

Contacts

Jesús Alejandro Valdés Valdés: alejandro.valdesval@gmail.com
Philipp Müller: philipp.mueller@tum.de

1 INTRODUCTION

1.1 How can we visualize data in a browser

Before we move on what different libraries exist to represent data in web browsers we first have to talk about the different approaches how we can draw in our browser in general. There exist mainly three approaches to draw data in modern web browsers, namely:

1. HTML elements
2. Canvas
3. SVG

	Chrome	Internet Explorer	Firefox	Safari	Opera
Canvas	4.0	9.0	2.0	3.1	9.0
SVG	4.0	9.0	3.0	3.2	10.1

Table 1. Support table for Canvas and SVG, only the lowest browser versions are listed that fully support all features.

HMTL elements can be styled to represent data. For example a div could be used with different width, height and color style parameters to represent a bar in a bar chart. However building a framework solely on HTML elements and styling them is clunky and does not scale well. The reason why it doesn't scale well is that complex or round shapes are hard to do with HTML elements and come with performance issues. The performance issues are the result of hundreds or thousands pixel small HMTL elements you need to create and draw to be able to represent complex shapes. A big advantage of HTML elements is that they are supported in every browser. They also provide the same functionalities as DOM elements such as events and other behaviour. Another advantage of DOM elements is that they can be styled with CSS (Cascading Style Sheet) files.

Canvas is a HMTL5 element and can be seen as a via script drawable Bitmap surface. It provides different drawing methods that enable us to visualize every shape we want to. Canvas however doesn't have any knowledge about the drawn objects, it only draws pixels. This also means that there are no predefined event callbacks per element that we are used to from DOM elements. Another feature of Canvas that might be problematic is that Canvas redraws the visualization with a fixed update rate. This might put unnecessary strain on our computational resources. The browser support for Canvas can be found in table 1 for the main browsers. The lowest browser version is listed that supports all baisc features of Canvas.

SVG is also a HTML5 element and defines a scriptable drawing surface. SVG stands for Scalable Vector Graphic, which implies that it is resolution independent. Resolution independence means in this case that, independent of your zoom level, round shapes stay round and don't get pixelated. It also provides many drawable elements that can be modified by property parameters to visualize every shape we want them to. Different to Canvas SVG elements are DOM elements, this means we can easily define event callbacks on them. SVG also provides simple functions to animate elements. The support for SVG for the main browsers can also be found in table 1.

In comparison Canvas is performance wise faster than SVG if very many elements (approximatly more than thousand) have to be drawn. For our project we decided to use SVG over Canvas, because we wanted to create a interactive data visualization with only a small

© Oxford University Press 2015.

Library	SVG	Canvas	HTML elements
D3.js	✓	✓	✓
Raphäel.js	✓	-	-
vis.js	✓	✓	✓
Paper.js	-	✓	-
Chart.js	-	✓	-
and many more	⋮	⋮	⋮

Table 2. Small selection of available JavaScript data visualization libraries and their base method

number of elements. As mentioned it is very easy to define callback functions for different events such as mouse click or hover events on SVG elements. This helps us to create a more interactive web application in less time.

1.2 Data visualization libraries

There is a huge amount of data visualization libraries available online. But because our time was limited we had to select only a few libraries of them for a closer inspection. Our criteria were mainly the size of the community, the support, dependencies and the documentation. The documentation and also availablility of tutorials were of special importance, because we had to learn everything from scratch in a limited time. The table 2 shows a small selection of libraries that mostly or completely satisfied our criteria.

To note here is that **D3.js** is a general purpose visualization JavaScript library. It is used for manipulating documents based on data without being coupling with a proprietary framework. That simply means that you don't have to learn a new visualization language or syntax if you already know SVG or Canvas.
The library **Raphäel.js** uses the SVG WC3 recommendation and VML (Vector Markup Language) to create graphics. This enables Raphäel to be compatible with older browsers as well as modern browers.
The **vis.js** library uses different visualization methods, depending on the called function to create a certain representation.
The **Chart.js** library uses the HTML5 Canvas element, but also provides polyfills in order to support older browers like Internet Explorer 7.
The Paper.js library uses the HTML5 Canvas element to draw the data.
In our application we decided to use D3.js, because it has a good, active and big community as well as a very good documentation with many great examples. You can also find many very well written tutorials from community members and other external sources which helped us to understand D3's core concepts and their practical application. Another important criteria in our case was time. Developing a nice looking interactive web application with D3 in conjunction with SVG is very easy and very fast, because SVG elements are DOM elements and D3 uses binds data to DOM elements. More on this subject in the following section 2.

2 D3 - DATA DRIVEN DOCUMENTS

As we have mentioned before, one of the most important libraries for data visualization and the main topic of this paper is the JavaScript library D3. D3 stands for Data Driven Documents and its main function is to bind data to DOM elements with the goal of creating stunning interactive data oriented visualizations suitable for web applications.
We want to start with a brief history of D3, before diving into the core concepts of it. D3 was developed in the year 2011 mainly by Mike Bostock and Jeffrey Heer, which were also the main developers for D3s predecessor Protovis.
Protovis was developed in the year 2009 amid a growing necessity of new technologies capable of creating visual content for the web, the strongest points of Protovis were that it was a Plugin-less library. Plugin-less means that it didnt need a plugin to work while most of the other visualization libraries strictly needed one to work. Protovis was also designed to be accessible to non-programmers as it was a learn by example library. Those design concept were kept for D3.

2.1 Loading of external data

It was previously stated that D3 stood for Data Driven Documents, but what exactly is data and how can a developer obtain it? The easiest way of representing data using JavaScript is through the use of a variable, of course a developer can also opt for more powerful data structures, such as an array, a matrix or even objects and arrays of objects. This enables the developer to create simple static data visualizations, or visualizations made using user input. A more useful way of using data in D3 is importing it from a file or through an API call, and D3 provides us methods to do this. Importing data is most often used in combination with API's and databases. Therefore a change in the backend system will result in an updated visualization without any additional programming effort. Using for example the csv, html, json, tsv, xml or xhr methods a developer could obtain data from different sources.

2.2 Selections

After a developer has found suitable data for his work, he now needs a way of binding said data to a selection of DOM elements in order to be visualized on a website. D3 provides a selection type which is a subtype of JavaScripts arrays and extends their functionality quite a bit. D3 selecitons can be acquired by using the select() and selectAll() methods. These methods will return an array of elements from the current document.

```
1 var selection = d3.selectAll("p");
```

In this small code example the selection will contain every paragraph in our current document. The selection is obtained via the selectAll method with the filter parameter for parapgraph DOM elements. Selections in D3 use CSS3 selectors to select elements which means that you can select DOM elements the following ways:

- Tag ("div")
- Class (".myClass")
- Id ("#thisID")
- Attributes ("[color=red]")
- Logical AND (".this.that")
- Logical OR (".this, .that")

Once a selection is retrieved, operators can be applied to it to modify it in many ways. For example, set or get attributes, styles, properties,

HTML and text content. Probably the most powerful thing to do with a selection is to join it to a data set, which will be described in more depth in section 2.3 about joins. This approach allows us to operate on entire selections at once and we therefore rarely need to use for loops. D3 also allows method chaining on its selections which makes code more readable and understandable. this looks like the next piece of code:

```
D3.select("body")
  .append("div")
  .attr("class","divClass")
  .append("p")
    .text("I am a paragraph")
    .attr("class","pClass");
```

This code snippet first selects our body element then it adds a div element to it. From this moment on the following methods will act on the new div element, so the new class name attribute will only modify our div element and not our body element. Inside the div it will add a paragraph element, and as before the following methods will only affect our newly appended p element. Because selections are based on DOM elements the attribute, style and text method calls on selections will also return the current self reference to the selection of DOM elements.

2.3 Joins

D3s main goal is to visualize data. Since we already know how one can obtain data and how one can select certain DOM elements using D3, the concept of D3's data join can now be discussed. Joins are basically what makes D3 so powerfull and allows it to visualize data, using the *.data([values[, key]])* method. This method joins data specified in values with the selection the method is called upon, it returns a selection again. What this returned selection contains will be explained in the following section.
The value parameter can either be specified as an array containing any kind of information or a function that returns an array. The key parameter is optional and specifies a custom compare callback function. This function will be used to join selection elements and data elements. The following code example shows a simplification how the join could look like:

```
for(var i=0; i<data.length; i++) {
  for(var j=0; j<selection.length; j++) {
    if(key(data[i]) === key(selection[j])) {
      // match found! Resulting pair is data[i]
          and selection[j]
    }
  }
}

function key(d) {
  return d.comparedProperty;
}
```

If no function is defined as key parameter D3 will revert to the default function that joins selection elements and data based on their indices. A simplified join function could look like the following code:

```
// pseudocode
for(var i=0; i<data.length; i++) {
  for(var j=0; j<selection.length; j++) {
    if(i === j) {
      // match found! Resulting pair is data[i]
          and selection[j]
    }
  }
}
```

2.4 The Update, Enter, and Exit sub selections

In this section the figure 2 shows a venn diagram which might help you to understand the following explanation better. The circle with the label *Elements* represents all DOM elements of our selection. The selection on the other hand represents the current state of our document. The data values are the circle with the label *Data*. Because we want our document state to be driven by our data values, we have to compare them. In order to compare the current state (selection) with our wanted state (data values) we perform the join operator. When we join data on a selection D3 provides us with three different sub selections as seen in figure 2. These sub selections can be defined as:

1. The **update** sub selection is obtained by calling the *data(values)* operator on a selection. This operator will also save the other two sub selections as references in the update sub selection. The update sub selection contains all elements for which there exists matching data.

```
var updateSubSelection = d3.select("svg").
    selectAll("rect").data([values]);
```

   In this example we join our data values on a selection that contains all rectangles in the first SVG element we find in our current document.

2. The **enter** sub selection can be accessed by calling the operator *enter()* on the update sub selection. The enter sub selection contains all data values that did not find a match in our currently displayed selection of elements. We can initialize the data sub selection by appending to it. Otherwise changes to the sub selection won't persist.

```
var enterSubSelection = updateSubSelection.
    enter().append("rect");
```

   As mentioned before, in order to add the data to our document we have to add DOM elements containing our data into the document. In this example we used the append method to do so. Interesting to note here is that after calling the append operator the variable updateSubSelection will also contain our enterSubSelection. Therefore we must be careful in what order we call our operators on the update and enter sub selections.

3. The exit sub selection can be accessed by calling the operator *exit()* on the update sub selection. The exit sub selection contains all elements of the selection that did not find a match in the data values. Often we want to remove those elements from the document. This can be done via remove call on the exit sub selection as shown in the following example:

```
var exitSubSelection = selection.exit();
exitSubSelection.remove();
```

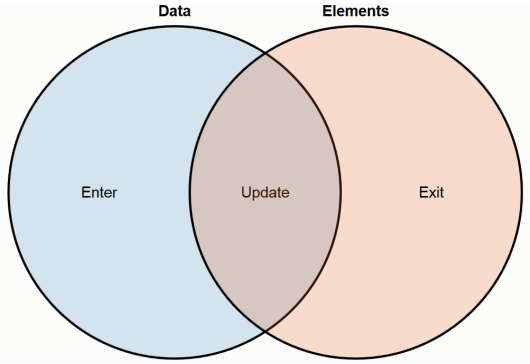

Fig. 2. Venn diagram

This seperation of the join into three disjunktive sets allows us to specify precisely what operations have to be run on which selections. Due to its disjunktive natur this can also improve the performance because we don't have to handle all sub selections, but pick the sub selections we are interested in. Which brings us back to the meaning of D3, Data Driven Documents. As we have seen we can transform documents based on data, e.g. the creation and destruction of DOM elements by calling operators on sub selections, which we obtain via data joins on selections. We can also modify them by calling different available operators such as .attr(), .style() and .text() on selections. Changes in those selections can be animated via the .transition() operator that we will discuss in the section 2.6.

2.5 Dynamic properties

As mentioned in the section before D3 enables us to modify selections dynamically. The syntax has many similarities with jQuery, however there exists a big difference. Properties such as stlyles, attributes and text can not only be specified as a constant but also as a function of data, and the index associated with the element that calls the method. To better understand this concept the following example shows how we can use dynamic properties. In the next code snippet we set the name and font size of each paragraph to a constant variable. We assume that there exists at least three paragraphs in our HTML body.

```
1 d3.select("body").selectAll("p")
2     .text("See")
3     .style("font-size", "8px");
```

This however might not be want we want. If we want to set the name of each paragraph to a different value and also increase the font size with increasing indice of the paragraph, the code could look something like this:

```
1 d3.select("body").selectAll("p")
2     .data(["See", "The", "World"])
3     .text(function(d) { return d; })
4     .style("font-size", function(d,i) { return (i
          *4+8) + "px"; });
```

Here we join the selection with data values which represents our paragraph names. Further we call our text and style operators on the update sub selection that will contain all three paragraph elements, because we join based on array indices. As you can see the operators have a function defined in them. If you define a function D3 will pass this function the self reference of the element (this), the associated data value as first parameter and the global index of the element in the array as second parameter. Both parameters are optional. Based on those informations we define for example the font size in the style property by the base value of 8 pixels plus four times the index of the element.

2.6 Transitions

In the sections before we have seen how a developer could select an element and change its attributes, however these changes would take place immediately. To create a more interactive and responsive application we also want to have animations. Luckily D3 provides transitions that enable us to apply these exact changes over time via interpolation, thus achieving an animation effect. The transitions have the following life cycle:

1. The transition is scheduled. (Gets schedule with selection.transition() call in JS)
2. The transition starts. (After the delay, if delay was specified)
3. The transition runs.
4. The transition ends

It is important to notice that transitions are exclusive and only act one given element, so only one transition can be executed at a given time, any new transitions will immediately stop any transitions that were running. You can also define custom interpolation functions for transitions, they have to take a parameter $t \in [0, 1]$.

2.7 Our web application - See the world

To demonstrate a small portion of the amazing powers of D3, we developed a small web application that would allow us to get a feeling for the core concepts of D3 and what it meant to be working with it. The application is called See the World and is meant to be used to create social conscience for the difference between countries. Using an interactive map users can select any country in the world and click on it to see a grahpical representation of data like how many citizens out of 100 have internet access, or the average yearly income per capita.
All of the visual elements were created with HMTL5 SVG and D3. The data gets pulled from the world bank API, which means an internet connection is required. To see the source code or simply check out the application, go to https://github.com/PhilippMueller1991/JS-Visualization

2.8 Discussion

2.8.1 Disadvantages of D3

D3 is a very powerful library however it does have disadvantages. For example, it has a steep learning curve and as it is largely used with SVG elements one could say that it is not IE8 compatible. D3 can be used to create charts, but it is not a graphing library, the user should not expect really easy out of the box methods that would do all of the work, such as makePieChart(50,50), D3 requires a little more creativity than that. Another big disadvantages is that because it works with the DOM elements response time can become problem on very large documents.

2.8.2 Advantages of D3

However D3 also has a lot of advantages. One main advatage is that it has a great community and that because it is not a graphing nor a charting library it gives the user total creative freedom to visualize whatever they want, however they want. Other advantages are that it uses commonly known selectors. D3s capabilities to animate selections via transitions is also a advantage. D3 in combination with SVG also results in fast development cycle and empowers users to protoype ideas very fast.

3 CONCLUSION

Throughout this report we have learned about one of the biggest visualization library for JavaScript which in the last couple of years has been used by data visualization programmer and experts all over the world to represent data in ways that break past barriers of simple charts and plots. We have learned about the backbone of D3 and how it allows us not only to create static content, but also dynamic real time applications. With the help of its great community and documentation, anyone with some previous JavaScript knowledge can start creating his or her own visualizations, that is if they dedicate enough time to invest some time to learn D3 and overcome the steep learning curve of it.

To finish this report it is necessary to point out in which cases it would be better to opt for other technologies.

3.1 When not to use D3

1. If you have no time to learn new concepts
2. If advanced big data operations are needed, however D3 can be combined with another library to neutralize this disadvantage.
3. Real graphing software is needed and you don't total control over creative aspect of graphs. You just want some graphs really fast with nearly no effort.

3.2 When to use D3

1. If you want is a Plugin-less visualization library. Which means it does not require any other libraries to work. You don't have to worry about other libraries creeping in your application like it is often the case with libraries that are based on jQuery.
2. If you want the visualization not be bound on any specific visualization framework or HTML5 element such as Canvas or SVG.
3. If you want total creative freedom.
4. If you want to quickly develope a prototype.