Visualization-D3.js

Bingyu Wang
rainicy925@gmail.com

Northeastern University

July 28, 2014

Introduction

- Why d3.js?
- Getting started
- Summary

Why d3.js?

- Most graphing packages take a configuration object.
- D3 is much more flexible and direct, and gets over the fact that the option you want are never available.
- Slower to get started but much more productive and flexible.
- Great documentation, examples community.

Fundamentals

Working with D3 requires an appreciation of the following concepts:

- HTML
- DOM
- CSS
- JavaScript
- SVG

HTML

Hypertext Markup Language is used to structure content for web browsers. The simplest HTML page looks like this:

```
<html>
    <head>
        <title>Page Title</title>
    </head>
    <body>
        <h1>Page Title</h1>
        <p>This is a really interesting paragraph.</p>
    </body>
</html>
```

Figure 1 : HTML

DOM

The Document Object Model refers to the hierarchical structure of HTML. Each bracketed tag is an element, and we refer to elements' relative relationships to each other in human terms: parent, child, sibling, ancestor, and descendant. In the HTML above, body is the parent element to both of its children, h1 and p (which are siblings to each other). All elements on the page are descendants of html.

DOM Example

For Example:

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Hello World</title>
5    </head>
6    <body>
7      <p>
8         <span style="color:red;">Hello World!</span>
9      </p>
10     <input type="button" value="Hello" onClick="alert('Hello World!');" />
11   </body>
12 </html>
```

The **p** element has a parent => **body**
The **p** element has a sibling => **input**
The **p** element has a child => **span**

Figure 2 : DOM

CSS

Cascading Style Sheets are used to style the visual presentation of HTML pages.

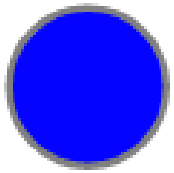
D3 uses CSS-style selectors to identify elements on which to operate, so it is important to understand how to use them.

JavaScript

JavaScript is a dynamic scripting language that can instruct the browser to make changes to a page after it has already loaded.

SVG

D3 is at its best when rendering visuals as Scalable Vector Graphics. SVG is a text-based image format. Meaning, you can specify what an SVG image should look like by writing simple markup code, sort of like HTML tags. For example:



```
<svg width="50" height="50">
    <circle cx="25" cy="25" r="22"
     fill="blue" stroke="gray" stroke-width="2"/>
</svg>
```

Figure 3 : SVG

Getting Started

1. Adding elements
2. Binding data
3. Using data
4. Drawing div
5. Drawing SVG

Adding elements

An example: **d3.select("body").append("p").text("New paragraph!");**
Let us walk through what just happened. In sequence, we:

1. Invoked D3's select method, which selects a single element from the DOM using CSS selector syntax. (We selected the body.)
2. Created a new p element and appended that to the end of our selection.
3. Set the text content of that new, empty paragraph to New paragraph!

Binding data

Data visualization is a process of mapping data to visuals. Data in, visual properties out.
With D3, we bind our data input values to elements in the DOM.
Binding is like attaching or associating data to specific elements, so that later you can reference those values to apply mapping rules.

Data

D3 is smart about handling different kinds of data, so it will accept practically any array of numbers, strings, or objects(themselves containing other arrays or key/value pair). It can handle **JSON** (and **GeoJSON**) gracefully, and even has a built-in method to help you load in **CSV** files. For example:

```
var dataset = [ 5, 10, 15, 20, 25 ];
```

Figure 4 : Data

Using Data

We use .data(datatset) for loading data. Once it is loaded. We can use High-functioning to parse each element in the whole dataset. For example:

High-functioning

In case you're new to writing your own functions (a.k.a. methods), the basic structure of a function definition is:

```
function(input_value) {
    //Calculate something here
    return output_value;
}
```

Figure 5 : High Function

Drawing divs

This is for starting drawing with data. Define a **div** as following:

```
div.bar {
    display: inline-block;
    width: 20px;
    height: 75px;   /* We'll override this later */
    background-color: teal;
}
```

Figure 6 : DIV

Drawing divs

The next step is using the div we just defined, named **bar**. The following is the example:

```
d3.select("body").selectAll("div")
      .data(dataset)
      .enter()
      .append("div")
      .attr("class", "bar");
```

Figure 7 : Using bar DIV

Drawing divs

The final step we want to apply our dataset = [5,10,15,20,25] to the bar's heights. Using as following:

```
.style("height", function(d) {
    return d + "px";
});
```

Figure 8 : Using Style

SVG

Drawing with divs and other native HTML elements is possible, but a bit clunky and subject to the usual inconsistencies across different browsers.
In fact D3 is most useful when used to generate and manipulate visuals as SVGs. Using SVG is more **reliable, visually consistent, and faster**.

SVG Shapes

There are a number of visual elements that you can include between those SVG tags, including:

- rect
- circle
- ellipse
- line
- text
- path

SVG Shapes Examples

```
<circle cx="25" cy="25" r="20" fill="rgba(128, 0, 128, 1.0)"/>
<circle cx="50" cy="25" r="20" fill="rgba(0, 0, 255, 0.75)"/>
<circle cx="75" cy="25" r="20" fill="rgba(0, 255, 0, 0.5)"/>
<circle cx="100" cy="25" r="20" fill="rgba(255, 255, 0, 0.25)"/>
<circle cx="125" cy="25" r="20" fill="rgba(255, 0, 0, 0.1)"/>
```

Figure 9 : SVG Example

Drawing SVGs

Now if we are familiar with the basic structure of an SVG image and its elements, how can we start generating shapes from our data?
We noticed that all properties of SVG elements are specified as attributes. That is, they are included as property/value pairs within each element tag, like this:

```
<element property="value"/>
```

Figure 10 : SVG Property

Fortunately, we have already used D3's handy **append()** and **attr()** methods to create new HTML elements and set their attributes. We can use these methods to generate SVG images.

Drawing SVGs

Two steps to use the data set generate SVG images.

1. Create the SVG
2. Data-driven Shapes

Create the SVG

First, we need to create the SVG element in which to place all our shapes. Example as following:

```
var svg = d3.select("body")
            .append("svg")
            .attr("width", 500)
            .attr("height", 50);
```

Figure 11 : Create a SVG

Data-driven Shapes

Time to add some shapes. Still use **var dataset = [5,10,15,20,25]**.
Then use data() to iterate through each data point, creating a circle for each one, as following:

```
var circles = svg.selectAll("circle")
                 .data(dataset)
                 .enter()
                 .append("circle");
```

Figure 12 : Creating Circles on SVG

Data-driven Shapes

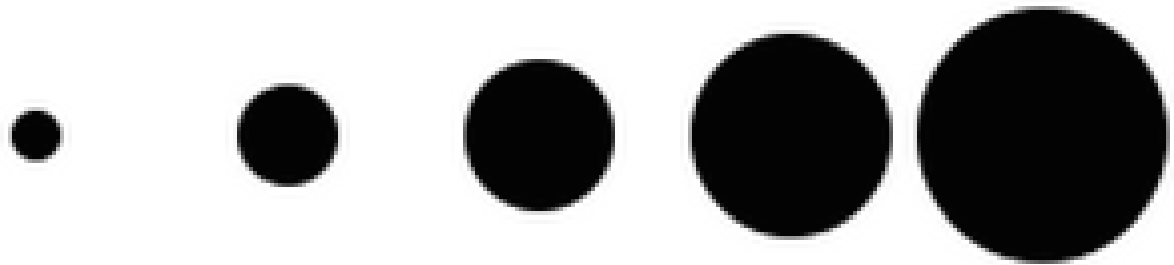
Look at the code in previous slide,

1. selectAll() will return empty references to all circles(which don't exist yet)
2. data() binds our data to the elements we're about to create
3. enter() returns a placeholder reference to the new element
4. append() finally adds a circle to the DOM

Data-driven Shapes

To continue play with circles in SVG. All these circles still need positions and sizes. Like:

```
circles.attr("cx", function(d, i) {
            return (i * 50) + 25;
        })
       .attr("cy", h/2)
       .attr("r", function(d) {
            return d;
       });
```

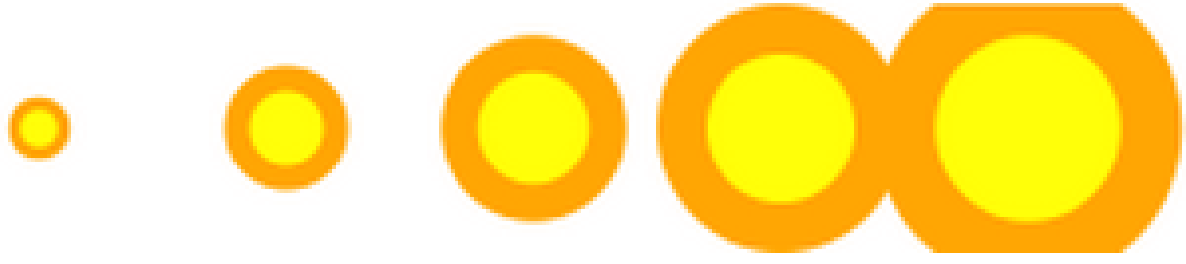


Data-driven Shapes

Do more color fills and strokes. Like:

```
.attr("fill", "yellow")
.attr("stroke", "orange")
.attr("stroke-width", function(d) {
    return d/2;
});
```

we get the following (see demo):



Summary

This is the basics we have learned so far:

- Make DOM objects, attach data to them, use CSS.
- The data can affect any attribute, shape, color, size, position, opacity, text, etc.
- All the attributes can have transitions applied.

There is more stuff to help us:

- Add and remove objects to dynamically update visualization.
- Helper functions for different charts: chord, histogram, hierarchy, pie, stack, tree, cluster, etc. [See more examples later]
- Drag and touch events, date formatting etc.

Helper Functions Examples

Chord `http://bost.ocks.org/mike/uberdata/`

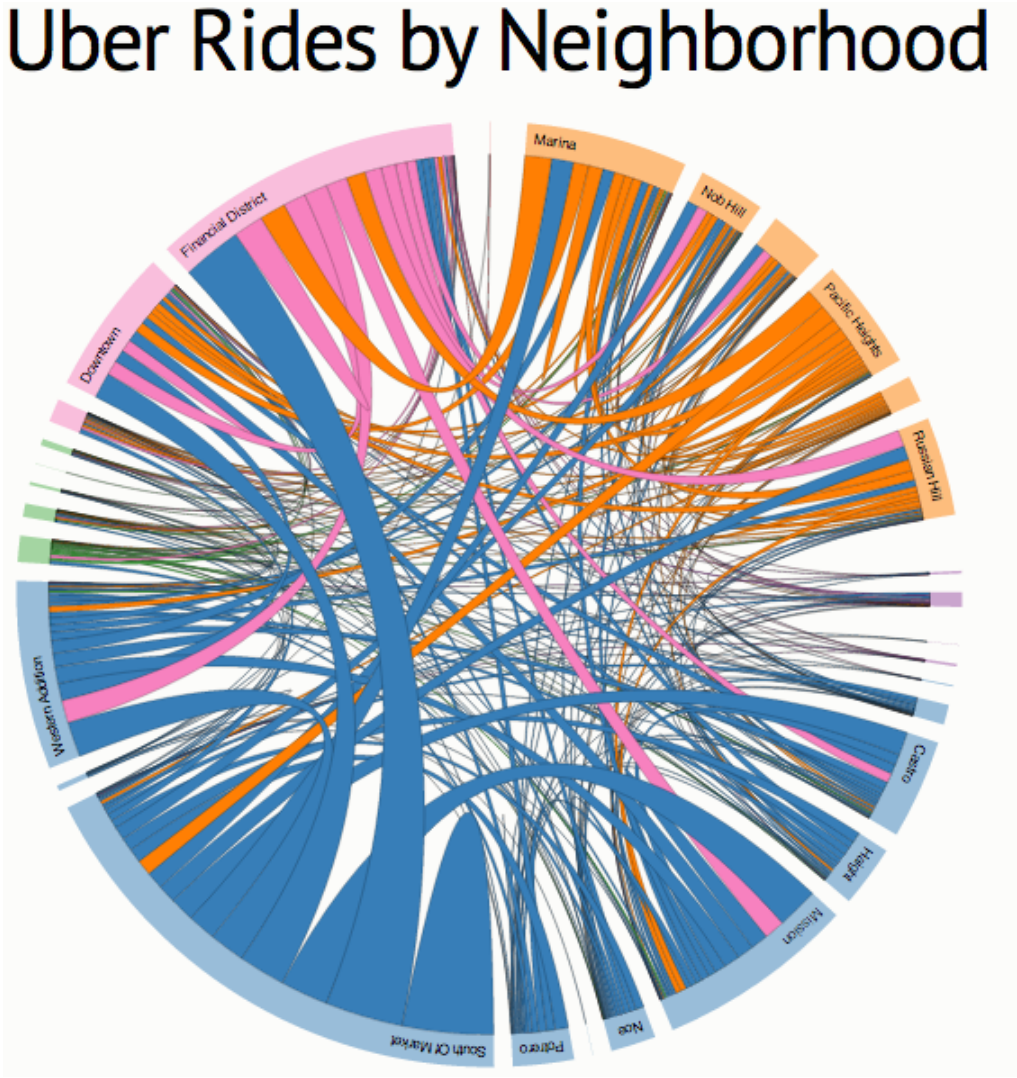


Figure 15 : Chord Examples

Stack `http://bl.ocks.org/mbostock/3943967`

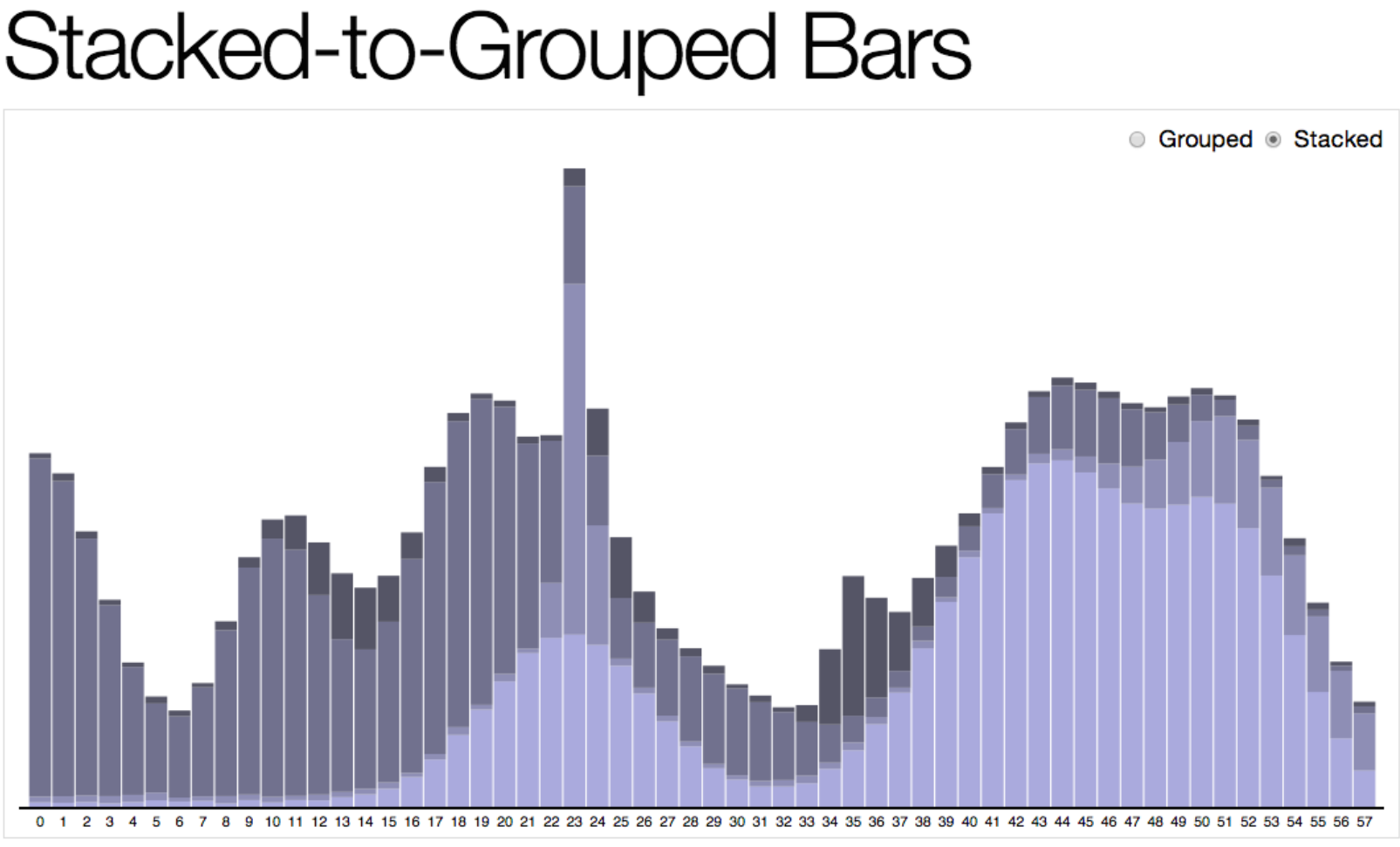


Figure 16 : Stack Examples

Cluster `http://bl.ocks.org/gmamaladze/9320969`

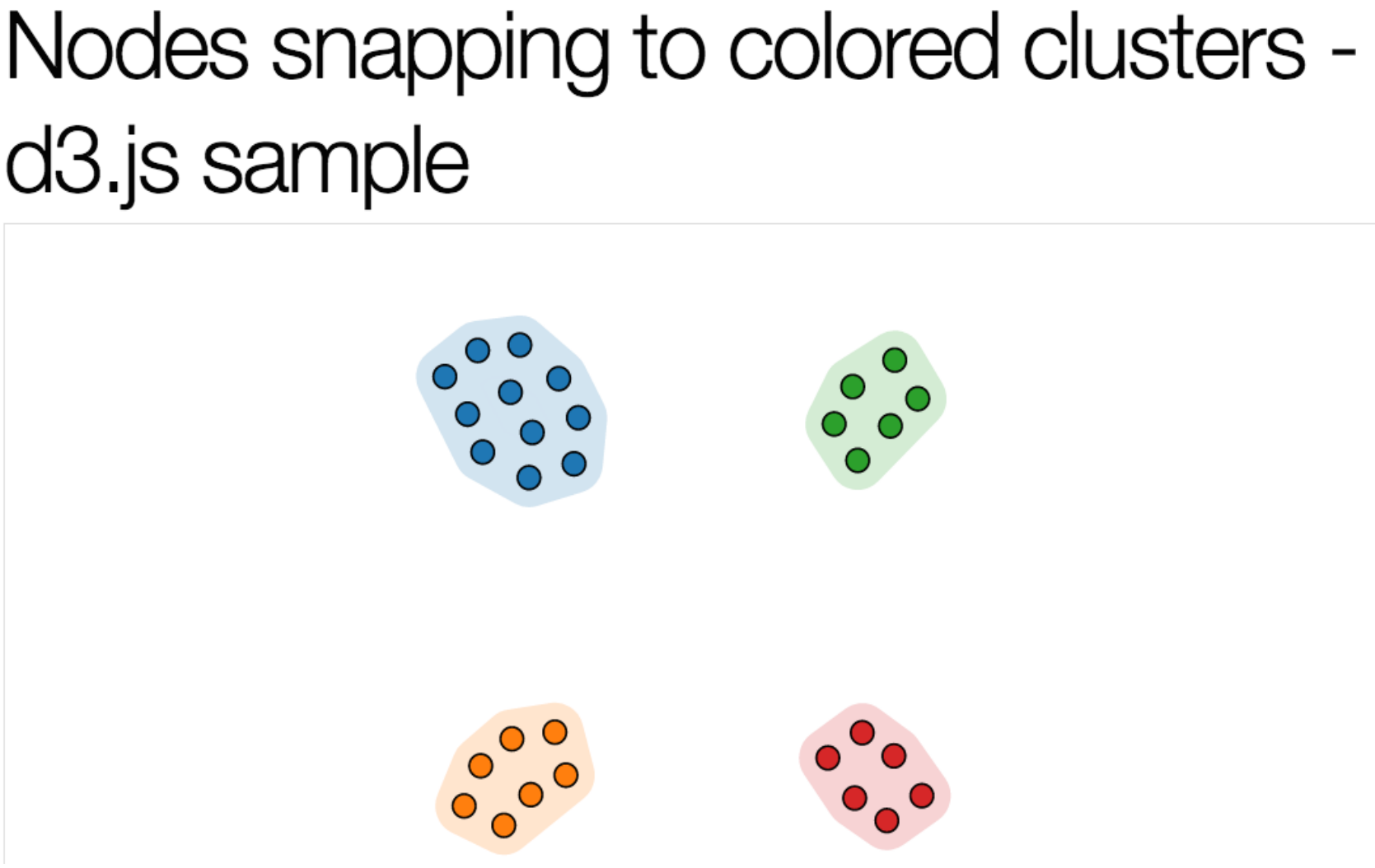


Figure 17 : Cluster Examples

Tree

http://mbostock.github.io/d3/talk/20111018/tree.html

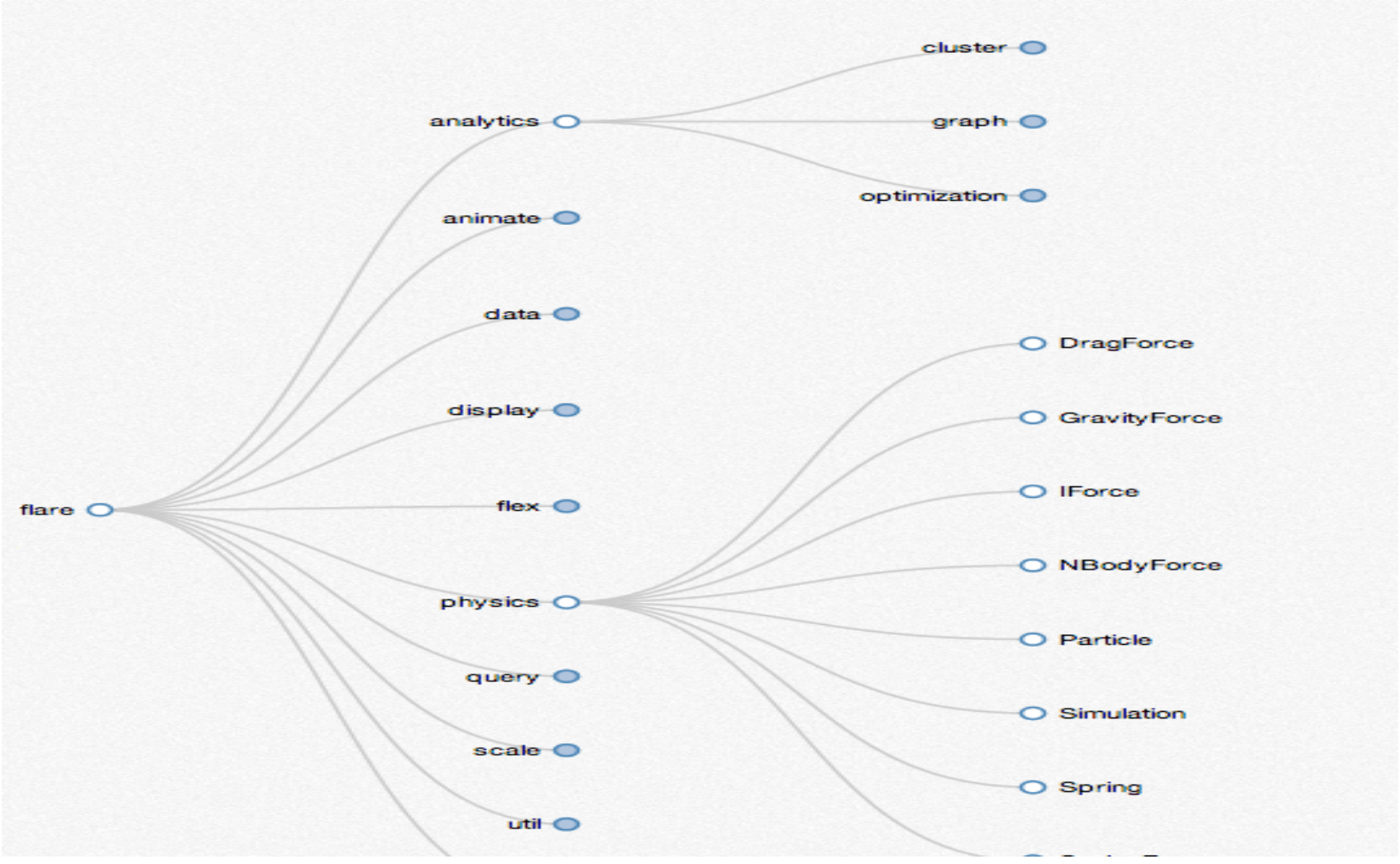


Figure 18 : Tree Examples

References

- http://alignedleft.com/
- http://lws.node3.org/
- https://www.dashingd3js.com/
- http://d3js.org/