CSE 564
Visualization & Visual Analytics

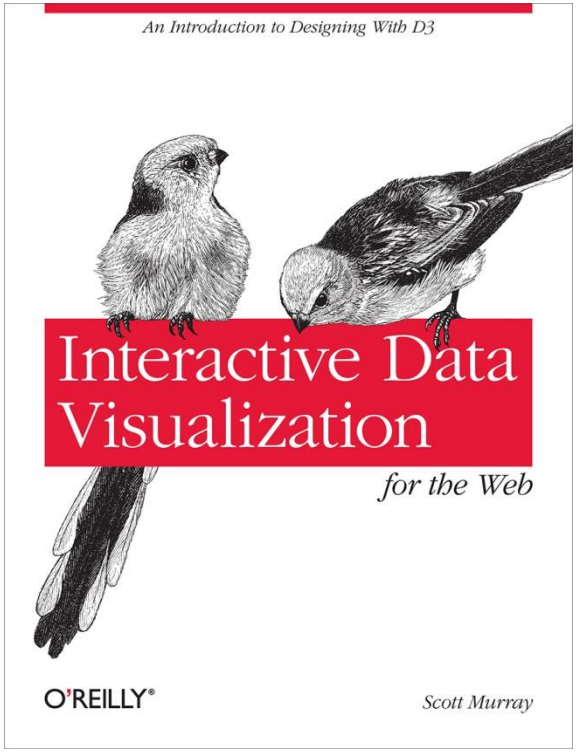
Introduction to D3

Klaus Mueller

Computer Science Department
Stony Brook University

Lecture	Topic	Projects
1	Intro, schedule, and logistics	
2	Applications of visual analytics	
3	Basic tasks, data types	Project #1 out
4	Data assimilation and preparation	
5	Introduction to D3	
6	Bias in visualization	
7	Data reduction and dimension reduction	
8	Visual perception	Project #2(a) out
9	Visual cognition	
10	Visual design and aesthetics	
11	Cluster analysis: numerical data	
12	Cluster analysis: categorical data	Project #2(b) out
13	High-dimensional data visualization	
14	Dimensionality reduction and embedding methods	
15	Principles of interaction	
16	Midterm #1	
17	Visual analytics	Final project proposal call out
18	The visual sense making process	
19	Maps	
20	Visualization of hierarchies	Final project proposal due
21	Visualization of time-varying and time-series data	
22	Foundations of scientific and medical visualization	
23	Volume rendering	Project 3 out
24	Scientific and medical visualization	Final Project preliminary report due
25	Visual analytics system design and evaluation	
26	Memorable visualization and embellishments	
27	Infographics design	
28	Midterm #2	

The material presented in these slides is derived from this book:



Also available online

What is D3.js?

D3 = Data Driven Documents

JavaScript library for manipulating documents based on data

- frequent tool to support *data journalism* (New York Times)

D3 helps you bring data to life using HTML, SVG, and CSS

- great library to construct animated visualizations (D3 website)

Runs in any modern web browser (Chrome, Firefox, IE)

- no need to download any software
- independent of OS (Linux, Windows Mac)

Makes Use Of

HTML Hypertext Markup Language

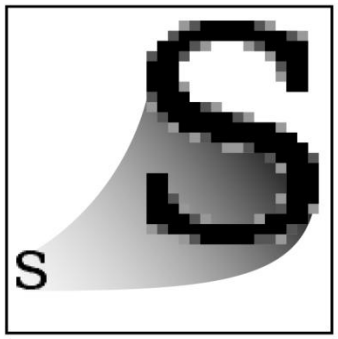
CSS Cascading Style Sheets

JS JavaScript

DOM The Document Object Model

- tree structured organization of HTML objects

SVG Scalable Vector Graphics



Raster

.jpeg .gif .png

Vector

.svg

What You Need

A text editor

- Visual; Studio Code, Atom, sublime text 2, or your browser
- need an editor with syntax highlighting. else it's easy to get lost

The d3 library

- from http://d3js.org

Data files for your code

A web server

- use python -m http.server 8000

A browser

- to run the code

Setup

Your folder structure should look like this:

```
project-folder/
    d3/
        d3.v3.js // D3 library
        d3.v3.min.js (optional) // minified D3 library
    index.html
```

Setup

Your initial webpage (index.html) should look like this:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>D3 Page Template</title>
        <script type="text/javascript" src="d3/d3.v3.js"></script>
    </head>
    <body>
        <script type="text/javascript">
            // Your beautiful D3 code will go here
        </script>
    </body>
</html>
```

Setting up a webserver

MAMP = My Apache, MySQL, PHP

- really only need Apache for now
- MS Windows = WampServer and XAMPP for Windows
- Mac = MAMP or XAMPP for Mac

Procedure

- install package (Linux has it already installed)
- find webserver folder (only files residing there will be served)
- put project files there
- open browser and point to http://localhost:8000/ or http://localhost:8000/project-folder/

Let's Use Some Simple Data

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

Chaining

Consider the following js code … all methods are chained:

```
d3.select("body").selectAll("p")
    .data(dataset)
    .enter()
    .append("p")
    .text("New paragraph!");
```

which gives this output

- how did this happen?

New paragraph!

New paragraph!

New paragraph!

New paragraph!

New paragraph!

Chaining

Consider the following js code … all methods are chained:

```
d3.select("body").selectAll("p") // selects all paragraphs in the DOM (none so far …)
    .data(dataset)   // counts and parses the data values
    .enter()  //  creates new, data-bound elements (placeholders) for the data values
    .append("p")  // takes the empty placeholder and adds a p-element
    .text("New paragraph!");  // takes the p-element and inserts a text value
```

which gives this output

- how did this happen?

New paragraph!

New paragraph!

New paragraph!

New paragraph!

New paragraph!

Using the Data

Change the last line to:

```
d3.select("body").selectAll("p")
    .data(dataset)
    .enter()
    .append("p")
    .text(function(d) { return d; });
```

which gives this output

- how did this happen?

5

10

15

20

25

Using the Data

Change the last line to:

```
d3.select("body").selectAll("p")
    .data(dataset)
    .enter()
    .append("p")
    .text(function(d) { return d; });  // used the data to populate the contents of each paragraph of the data-driven document
```

which gives this output

- how did this happen?

5

10

15

20

25

Using Functions

Change the last line to:

```
d3.select("body").selectAll("p")
    .data(dataset)
    .enter()
    .append("p")
    .text(function(d) { return "I can count up to " + d; });
```

which gives this output

- how did this happen?

I can count up to 5

I can count up to 10

I can count up to 15

I can count up to 20

I can count up to 25

Adding Aesthetics

Change the last line to:

```
d3.select("body").selectAll("p")
    .data(dataset)
    .enter()
    .append("p")
    .text(function(d) { return "I can count up to " + d; })
    .style("color", "red");
```

which gives this output

- how did this happen?

I can count up to 5

I can count up to 10

I can count up to 15

I can count up to 20

I can count up to 25

More Complex Functions

Replace the last line with:

```
d3.select("body").selectAll("p")
    .data(dataset)
    .enter()
    .append("p")
    .text(function(d) { return "I can count up to " + d; })
    .style("color", function(d) { if (d > 15) {return "red"; } else { return "black"; } });
```

which gives this output

- how did this happen?

I can count up to 5

I can count up to 10

I can count up to 15

I can count up to 20

I can count up to 25

Drawing With Data

Let's draw some bar charts

For this, put this embedded style in the document head

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

Simple Bar Chart

Run this code:

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

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

which gives this output

- five bars with no space between them
- how did this happen?

More Complex Bar Chart

Run this code:

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

d3.select("body").selectAll("div")
    .data(dataset)
    .enter()
    .append("div")
    .attr("class", "bar")
    .style("height", function(d) { return d + "px"; });
```

which gives this output

- how did this happen?

More Complex Bar Chart

Run this code:

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

d3.select("body").selectAll("div")
    .data(dataset)
    .enter()
    .append("div")
    .attr("class", "bar")
    .style("height", function(d) { return d + "px"; });   // adds text “px” to specify that the units are pixels → heights are 5px, 10px, 15px, 20px, and 25px
```

which gives this output

- how did this happen?

Make Bars Taller and Add Spaces

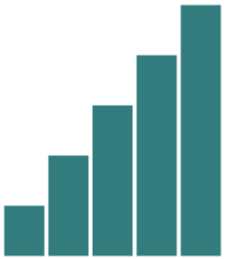
Run this code: (also add margin-right: 2px; to the css style)

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

d3.select("body").selectAll("div")
    .data(dataset)
    .enter()
    .append("div")
    .attr("class", "bar")
    .style("height", function(d) { var barHeight = d * 5; return barHeight + "px"; });
```

which gives this output

- how did this happen?



Get Ready to Draw Some SVGs

Optionally define some variable beforehand, e.g.:

```
// width and height
var w = 500;
var h = 50;
```

Define the svg object:

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

get Ready to Draw Some SVG Circles

Define the circles as variables for ease of reference:

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

But could so this just as well:

```
svg.selectAll("circle")
   .data(dataset)
   .enter()
   .append("circle"); // now circles are appended to the end of the SVG element
```

Now Draw The Circles

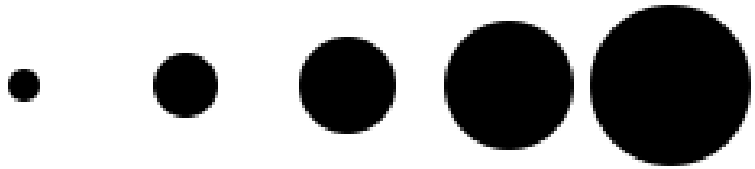
Run this code (still using var dataset = [5, 10, 15, 20, 25];)

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

or append it to the .append("circle") method

This gives this output

- how did this happen?



Now Draw The Circles

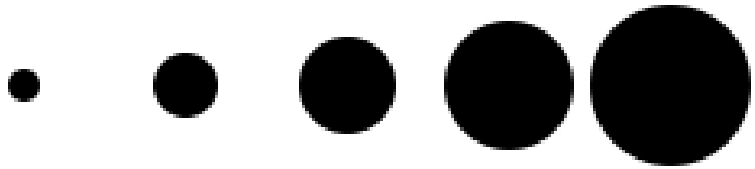
Run this code (still using var dataset = [5, 10, 15, 20, 25];)

```
circles.attr("cx", function(d, i) {return (i * 50) + 25;})  // i increments by 1 each time,
                                                            starting at 0
    .attr("cy", h/2)
    .attr("r", function(d) {return d;});
```

or append it to the .append("circle") method

This gives this output

- how did this happen?



Adding Colors

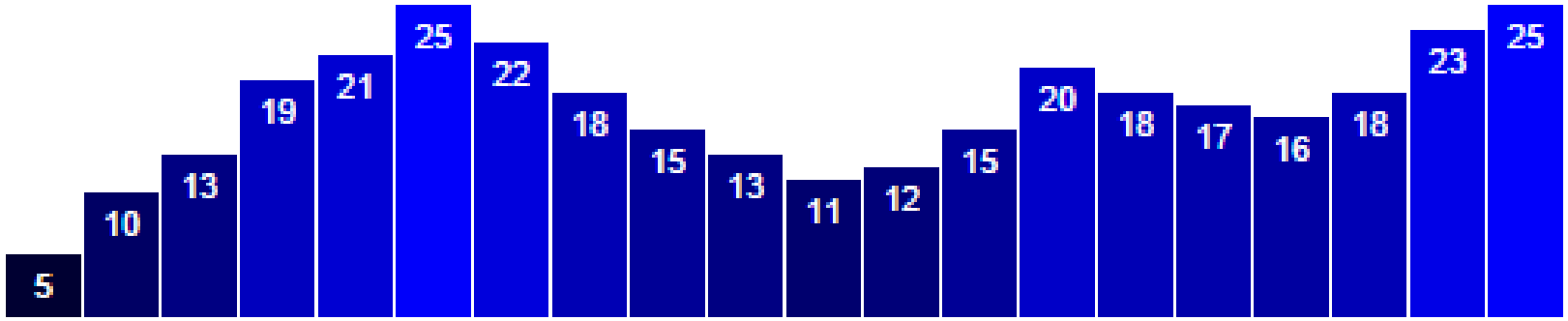
Run this code (still using var dataset = [5, 10, 15, 20, 25];)

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

This gives this output

- how did this happen?

Bar Charts



Code

Updates

This will update the bar chart on a mouse click:

```
d3.select("p")
    .on("click", function() {

        //New values for dataset
        dataset = [ 11, 12, 15, 20, 18, 17, 16, 18, 23, 25, 5, 10, 13, 19, 21, 25, 22, 18, 15, 13 ];

        //Update all rects
        svg.selectAll("rect")
           .data(dataset)
           .attr("y", function(d) {
                return h - yScale(d);
           })
           .attr("height", function(d) {
                return yScale(d);
           });

    });
```

Adding Animated Transitions

Smooth animations are desirable:

```
svg.selectAll("rect")
   .data(dataset)
   .transition()
   .attr("y", function(d) {
        return h - yScale(d);
   })
   .attr("height", function(d) {
        return yScale(d);
   })
   .attr("fill", function(d) {
        return "rgb(0, 0, " + (d * 10) + ")";
   });
```

Control Duration

Now run this code:

```
svg.selectAll("rect")
   .data(dataset)
   .transition()
   .duration(1000)  // <-- Now this is new!
   .attr("y", function(d) {
        return h - yScale(d);
   })
   .attr("height", function(d) {
        return yScale(d);
   })
   .attr("fill", function(d) {
        return "rgb(0, 0, " + (d * 10) + ")";
   });
```

Interaction

Facilitated by event handlers (listeners), e.g.:

```
d3.select("p")
    .on("click", function() {
        //Do something on click
    });
```

others react on

- mouse hovering
- mouse over
- mouse out
- and others

Example

Directing Action to a Specific Item

Assume you selected a certain item by mouseover

```
.on("mouseover", function() {
        //Do something on mouseover of any bar
});
```

Keyword “this” maps the action to the selected item

```
.on("mouseover", function() {
        d3.select(this)
          .attr("fill", "orange");
});
```

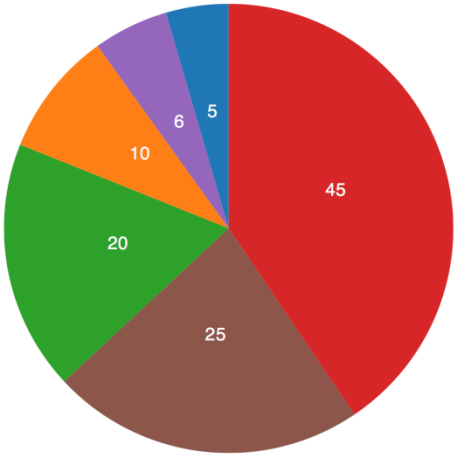
Layouts

D3 layouts take data that you provide

- remap or otherwise transform it
- and so generating new data that is more convenient for a specific visual task

The supported layouts are:

- Bundle and Chord
- Cluster
- Force
- Histogram
- Pack, Partition, and Pie
- Stack
- Tree and Treemap



Force-Directed Layout

```
var dataset = {
        nodes: [
                { name: "Adam" },
                { name: "Bob" },
                { name: "Carrie" },
                { name: "Donovan" },
                { name: "Edward" },
                { name: "Felicity" },
                { name: "George" },
                { name: "Hannah" },
                { name: "Iris" },
                { name: "Jerry" }
        ],
        edges: [
                { source: 0, target: 1 },
                { source: 0, target: 2 },
                { source: 0, target: 3 },
                { source: 0, target: 4 },
                { source: 1, target: 5 },
                { source: 2, target: 5 },
                { source: 2, target: 5 },
                { source: 3, target: 4 },
                { source: 5, target: 8 },
                { source: 5, target: 9 },
                { source: 6, target: 7 },
                { source: 7, target: 8 },
                { source: 8, target: 9 }
        ]
};
```

```
var force = d3.layout.force()
                     .nodes(dataset.nodes)
                     .links(dataset.edges)
                     .size([w, h])
                     .linkDistance([50])        // <-- New!
                     .charge([-100])            // <-- New!
                     .start();
```

Next, we create an SVG line for each edge:

```
var edges = svg.selectAll("line")
        .data(dataset.edges)
        .enter()
        .append("line")
        .style("stroke", "#ccc")
        .style("stroke-width", 1);
```

Note that I set all the lines to have the same stroke color and weight, but of course you could set this dynamically based on data (say, thicker or darker lines for "stronger" connections, or some other value).

Demo

Then, we create an SVG circle for each node:

```
var nodes = svg.selectAll("circle")
        .data(dataset.nodes)
        .enter()
        .append("circle")
        .attr("r", 10)
        .style("fill", function(d, i) {
                return colors(i);
        })
        .call(force.drag);
```