Data visualization

Web-based visualization with D3.js

David Hoksza
http://siret.ms.mff.cuni.cz/hoksza

Outline

- Web visualization and D3.js
- SVG
- Selections
- Data binding
- Scales
- Arrays
- Layouts
- Geography
- Colors
- Behavior
- Charting libraries
- Time series libraries
- Graph libraries

Web visualization

- **Multiple technologies** for **multiple data types**
  - **Page content** – HTML
  - **Aesthetics** – CSS
  - **Vector graphics** – SVG
  - **Interactivity** – JavaScript
  - **Integration** enabled by shared page representation – **Document Object Model (DOM)**
    - Exposing the hierarchical structure of page content enabling reference and manipulation

Interactivity

- **Static** visualizations can only offer **precomputed views** → multiple views might be needed to rely the story (especially for multi-dimensional data)

- Interactivity is required when the goal is data exploration (EDA)
  - **Visual Information-Seeking Mantra**: ”***Overview first, zoom and filter, then details-on-demand*** “ [Ben Shneiderman (1996) The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations]

- Interactivity **helps to engage** an audience who might not be interested in the topic otherwise

Visualization toolkits

- There are several **libraries** making the development of visualization easier → **visualization toolkits**

- Usually, toolkits encapsulate DOM with scenegraph abstractions → intermediate representations
  - Decrease **accessibility**
    - Some of the internal structures might not be accessible
  - Diminish **expressiveness**
    - Some of the tasks enabled by the underlying representation might not be exposed by the toolkit
  - Impose runtime **overhead**
    - Fast incremental scene changes

Data-Driven Documents (D3)

- An embedded **domain-specific language** for **transforming** the **document object model based on data**
  - Client-based JavaScript library
- **Declarative framework** for **mapping data** to visual **elements**
  - Binding input data to arbitrary document elements to both generate and modify content → no toolkit-specific lexicon of graphical marks
- **Thin layer** on top of **standard modern web technologies**
  - JavaScript, SVG, HTML5, CSS3
  - D3 is a **visualization "kernel"** rather than a framework

D3 Origins

- **1996** – First browser (Netscape) with scripting language interpreted by the browser - **JavaScript**
- **2005** – Visualization toolkit based on Java run in Java plugin - **prefuse** → web-based visualization accessible to less-than-expert programmers
- **2007** – ActionScript (Adobe Flash Player) toolkit similar to prefuse (developed by Jeff Heer, one of the programmers of prefuse) – **Flare**
- **2009** - JavaScript-based visualization toolkit relying exclusively on native browser technologies → **Protovis** (developed by Michael Bostock, Heer's student at Stanford)
- **2011 – D3.js** – evolution of Protovis operating directly on the web document itself (Protovis used an abstract representation layer)

D3 features

- **Focused on explanatory** visualization rather then exploratory
  - Although with tools like Observable, it can also serve the exploratory purpose
- No support for older browsers
- **Original data** are **not hidden**, but sent to the client → if the data can't be shared, using D3 might be problematic
  - Still, there are applications to extract data even from static images, e.g., DataThief

D3 data process

- D3's goal is to generate and manipulate web documents with data
  - **Loading** data into browser
  - **Binding** data to existing elements (or create/remove), based on the incoming data
  - **Transforming** the elements using visual encodings based on the bound data
  - **Transitioning** elements based on user's input (interactivity)

D3 library overview

- **Selections**
  - Atomic operands which return filtered sets of elements from a document
- **Operators**
  - Act on selections and modify their content
- **Data joins**
  - Bind input data to elements, enabling functional operators that depend on data
- **Transitions**
  - Procedures like operators, but they do not apply attributes and styles immediately, but interpolate them smoothly over time
- **Visualization modules**
  - Help to simplify common visualization tasks (layouts, geography, scales, …)

SVG Interlude

SVG

- **S**calable **V**ector **G**raphics
- http://www.w3.org/TR/SVG/
- XML-based 2D vector graphics format
- Supported by all major web browsers
- Developed as an open standard by W3C
- All SVG elements are part of the DOM

- Advantages
  - can be created and edited using any text editor
  - can be searched, indexed, scripted, and compressed
  - **scalable**
  - can be printed with high quality at any resolution
  - **zoomable** (the image can be zoomed without degradation)
  - open standard
  - pure XML

SVG elements (1)

SVG Container (canvas)

```
<svg width="1000" height="1000">
    <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4" fill="yellow"/>

    <rect x="50" y="20" rx=" 20" ry="20" width="150" height="150"
style="fill:rgb(0,0,255);stroke:pink;stroke-width:5;fill-opacity:0.1;stroke-
opacity:0.9" />

    <ellipse cx="240" cy="100" rx="220" ry="30" style="fill:purple" />

    <line x1="0" y1="0" x2="200" y2="200" style="stroke:rgb(255,0,0);stroke-
width:2" />

    <polygon points="100,10 40,198 190,78 10,78 160,198"
style="fill:lime;stroke:purple; stroke-width:5;fill-rule:evenodd;" />
</svg>
```

SVG elements (2)

Path commands

```
<svg width="1000" height="1000">
      <polyline points="0,40 40,40 40,80 80,80 80,120 120,120
120,160"
  style="fill:white;stroke:red;stroke-width:4" />
      <path id="lineAB" d="M 100 350 l 150 -300" stroke="red"
  stroke-width="3" fill="none" />
      <path id="lineBC" d="M 250 50 l 150 300" stroke="red"
  stroke-width="3" fill="none" />
      <path d="M 175 200 l 150 0" stroke="green" stroke-width="3"
  fill="none" />
  <path d="M 100 350 q 150 -300 300 0" stroke="blue"
  stroke-width="5" fill="none" />
      <text x="0" y="15" fill="red" transform="rotate(30 20,40)">I
love SVG</text>
</svg>
```

moveto (absolute)

lineto (relative)

quadratic Bezier curveto (relative)

Pen commands		
M / m	x, y	Move the pen to a new location. No line is drawn. All path data must begin with a 'moveto' command.
Line commands		
L / l	x, y	Draw a line from the current point to the point (x, y).
H / h	x	Draw a horizontal line from the current point to x.
V / v	y	Draw a vertical line from the current point to y.
Cubic Bezier curve commands		
C / c	x1, y1, x2, y2	Draw a cubic Bézier curve from the current point to the point (x,y) using (x1,y1) as the control point at the beginning of the curve and (x2,y2) as the control point at the end of the curve.
S / s	x2, y2, x, y	Draw a cubic Bézier curve from the current point to (x,y). The first control point is assumed to be the reflection of the last control point on the previous command relative to the current point. (x2,y2) is the second control point (i.e., the control point at the end of the curve).
Quadratic Bezier curve commands		
Q / q	x1, y1, x, y	Draw a quadratic Bézier curve from the current point to (x,y) using (x1,y1) as the control point.
T / t	x, y	Draw a quadratic Bézier curve from the current point to (x,y). The control point is assumed to be the reflection of the control point on the previous command relative to the current point.
Elliptical arc curve command		
A / a	rx ry x-axis-rotation large-arc-flag sweep-flag x y	Draws an elliptical arc from the current point to (x, y). The size and orientation of the ellipse are defined by two radii (rx, ry) and an x-axis-rotation, which indicate how the ellipse as a whole is rotated relative to the current SVG coordinate system. large-arc-flag and sweep-flag contribute to the automatic calculations and help determine how the arc is drawn.
End path command		
L / l	none	Closes the path. A line is drawn from the last point to the first point drawn.

SVG styling

- To style SVG elements one should use the names of SVG attributes (not CSS properties)
  - SVG attributes and CSS properties have many names in common but not all (`color` vs `fill`)

MDN SVG Attribute reference

SVG filters

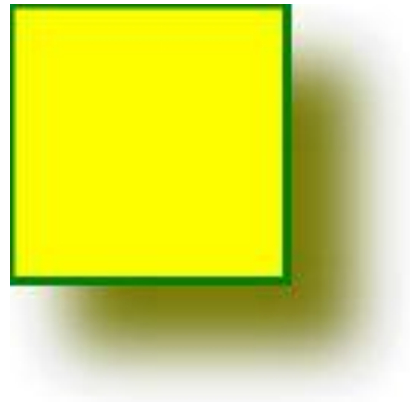
- http://www.w3.org/TR/SVG/filters.html

- **Series of graphics operations** that are applied over a given source graphic
  - Can be chained
- **Defined by** the **`filter`** element and **linked by** the **`filter`** property of given source graphics

  - `<feGaussianBlur>, <feBlend>, <feComposite>, <feMerge>, <feImage>, <feSpotLight>, <feOffset>, ,...`

```
<svg height="110" width="110">
    <defs>
        <filter id="f1" x="0" y="0">
            <feGaussianBlur in="SourceGraphic" stdDeviation="15" />
        </filter>
    </defs>
    <rect width="90" height="90" stroke="green" fill="yellow" filter=
"url(#f1)" />
</svg>
```

```
<svg height="140" width="140">
  <defs>
    <filter id="f4" x="0" y="0" width="200%" height="200%">
      <feOffset result="offOut" in="SourceGraphic" dx="20" dy="20" />
      <feColorMatrix result="matrixOut" in="offOut" type="matrix"
      values="0.2 0 0 0 0 0 0.2 0 0 0 0 0 0.2 0 0 0 0 0 1 0" />
      <feGaussianBlur result="blurOut" in="matrixOut" stdDeviation="10" />
      <feBlend in="SourceGraphic" in2="blurOut" mode="normal" />
    </filter>
  </defs>
  <rect width="90" height="90" stroke="green" stroke-width="3"
  fill="yellow" filter="url(#f4)" />
</svg>
```



SVG gradients

- **Smooth transition** from **one color to another**
- Several color transitions can be applied to the same element

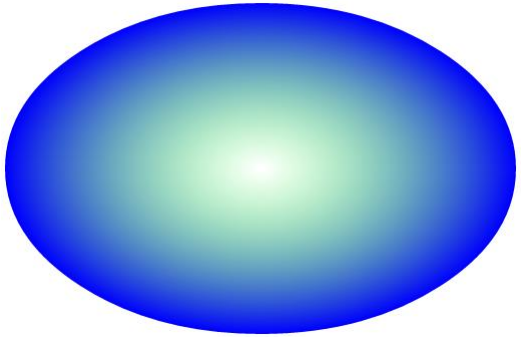
- Types
  - Linear
  - Radial

Linear gradient

```
<svg height="150" width="500">
  <defs>
    <linearGradient id="grad1" x1="0%" y1="0%"
x2="0%" y2="100%">
      <stop offset="0%" style="stop-
color:rgb(255,0,0);stop-opacity:1" />
      <stop offset="50%" style="stop-
color:rgb(255,255,0);stop-opacity:1" />
      <stop offset="100%" style="stop-
color:rgb(0,0,0);stop-opacity:1" />
    </linearGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55"
fill="url(#grad1)" />
</svg>
```

Radial gradient

```
<svg height="150" width="500">
  <defs>
    <radialGradient id="grad2" cx="50%"
cy="50%" r="50%" fx="50%" fy="50%">
      <stop offset="0%" style="stop-
color:rgb(0,255,0);
      stop-opacity:0" />
      <stop offset="100%" style="stop-
color:rgb(0,0,255);stop-opacity:1" />
    </radialGradient>
  </defs>
  <ellipse cx="200" cy="70" rx="85" ry="55"
fill="url(#grad2)" />
</svg>
```



SVG Transformation

$$\begin{vmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{vmatrix}$$

- Enables transformation of all SVG shapes and groups
- Available transformations
  - Move – `translate(x, y)`
  - Rotate – `rotate(angle, x, y)` … x and y defined the point around which the shape will be rotated
  - Scale – `scale(factor)`, `scale(factor_x, factor_y)` … scales size, position coordinates and stroke width (setting -1 as value of factor_x does mirroring)
  - Skew – `skewX(angle)`, `skewY(angle)`
  - Free transformation – `matrix(a, b, c, d, e, f)`

Translate

$$\begin{matrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{matrix}$$

Rotate

$$\begin{matrix} cos(a) & -sin(a) & 0 \\ sin(a) & cos(a) & 0 \\ 0 & 0 & 1 \end{matrix}$$

Scale

$$\begin{matrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{matrix}$$

Skew

$$\begin{matrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{matrix}$$

http://jsfiddle.net/DavidHoksza/xvrc6ojz/

SVG animation

- Manipulation of shape elements over time
- Animation elements

`<set>`	sets an attribute to a certain value after a specific time interval has passed
`<animate>`	animates an attribute of an SVG shape
`<animateColor>`	works on the color of a shape instead of its position or dimensions (`<animate>` does not work on colors)
`<animateTransform>`	animates the transform attribute of a shape (`<animate>` does not work on transformations)
`<animateMotion>`	animates the movement of a shape along a path

Often does not work (animate on color does)

- It is also possible to coordinate or combine animations

http://jsfiddle.net/DavidHoksza/swdzm5dz/

SVG grouping

- **g** element allows to group objects
- Transformation performed over all its child elements
- Attributes applied over all its child elements

```
<svg width="100%" height="100%">
  <g stroke="green" fill="white" stroke-width="5">
    <circle cx="25" cy="25" r="15" />
    <circle cx="40" cy="25" r="15" />
    <circle cx="55" cy="25" r="15" />
    <circle cx="70" cy="25" r="15" />
  </g>
</svg>
```

Back to D3

Including D3 into a webpage

```
<!doctype html>
<html>
<head>
    <title>Test</title>
    <script type="text/javascript" src="d3.min.js"></script>
    <!--<script src="https://d3js.org/d3.v6.min.js"></script>-->
</head>
<body>
    <p>Text</p>
    <script type="text/javascript">
        d3.select("p").text("Hello world");
    </script>
</body>
</html>
```

Data in D3

- Limited to **text-based** data
  - Commonly plain text files (.txt), comma-separated value files (.csv), xml files (.xml) or JSON documents (.json)
  - Arrays generated in the page
- **Data** need to be attached (**bound**) to something to be visualized → elements in the page (existing or newly formed based on the data)
  - Existing elements can be selected using **selectors** → **`d3.select()`, `d3.selectAll()`, …**
  - New elements created using **appending** to existing ones → **`d3.append()`**
  - Existing elements can be modified using **operators** → **`selection.attr()`, `selection.style()`, `selection.text()`, …**

Selections (1)

- Identify a set of elements using simple predicates (selectors), then apply a series of operators that mutate the selected elements

- D3 selectors **based on W3C Selectors API**

  - Patterns that match against elements in a tree → DOM tree nodes selection

Selections (2)

Type selectors	`h1 { background-color: rgb(255,0,255) }`
Class selectors	`.pastoral { color: green }` `h1.pastoral { color: green }`
ID selectors	`#chapter1 { color: red }, h1#chapter1 { color: red }`

Attribute selectors	`h1[title], span[class="example"],`
	`span[hello="Cleveland"][goodbye="Columbus"]`
	`a[rel~="copyright"],`
	`a[href="http://www.w3.org/"], a[hreflang*="en"]`
	`a[hreflang^="en"], a[hreflang$="en"]`

Pseudo-classes	`a.external:visited, a:hover, *:target { color : red }, tr:nth-child(2n+1) { color: navy; },` `tr:nth-child(2n) { color: steelblue; }`

Selection based on information that lies outside of the document tree

Selections (3)

- The Document Web API way of retrieving nodes corresponding to a selection

```
var matches = document.getElementById("footer")
var matches = document.getElementsByName("animal")
document.getElementsByTagName("P")[0].innerHTML = "Hello World!";
document.getElementsByClassName("dog cat")[0]
    .setProperty("color", "gray", null)


var matches = document.querySelector(".viewer-main")
var matches = document.querySelectorAll("h1")
var matches = document.querySelectorAll("div.note, div.alert");
```

Selections (4)

- **D3 selections**
  - Native DOM selectors return DOM objects, not D3 objects → **d3 namespace prefix** → **`d3.select("body"), d3.selectAll("div")`**
  - Filtering by **tag** ("tag"), **class** (".class"), **unique identifier** ("#id"), **attribute** ("[name=value] "), **containment** ("parent child")
  - Predicates can be **intersected** (".a.b") or **unioned** (".a, .b")

https://github.com/d3/d3/blob/master/API.md#selections-d3-selection

https://github.com/d3/d3-selection

Selections (5)

- To obtain a selection matching a predicate use **`select`** (first element) and **`selectAll`** (all elements) methods
  - Exported using the global `d3` : **`d3.select("body")`**
- **`append`** and **`insert`** operators add a new element for each element in the current selection, returning the added nodes
- **`remove`** operator discards selected elements
- Due to methods chaining **subselections** are possible : `d3.selectAll("p").select("b")`

https://github.com/d3/d3/blob/master/API.md#modifying-elements

https://github.com/d3/d3-selection/blob/v1.4.1/README.md#modifying-elements

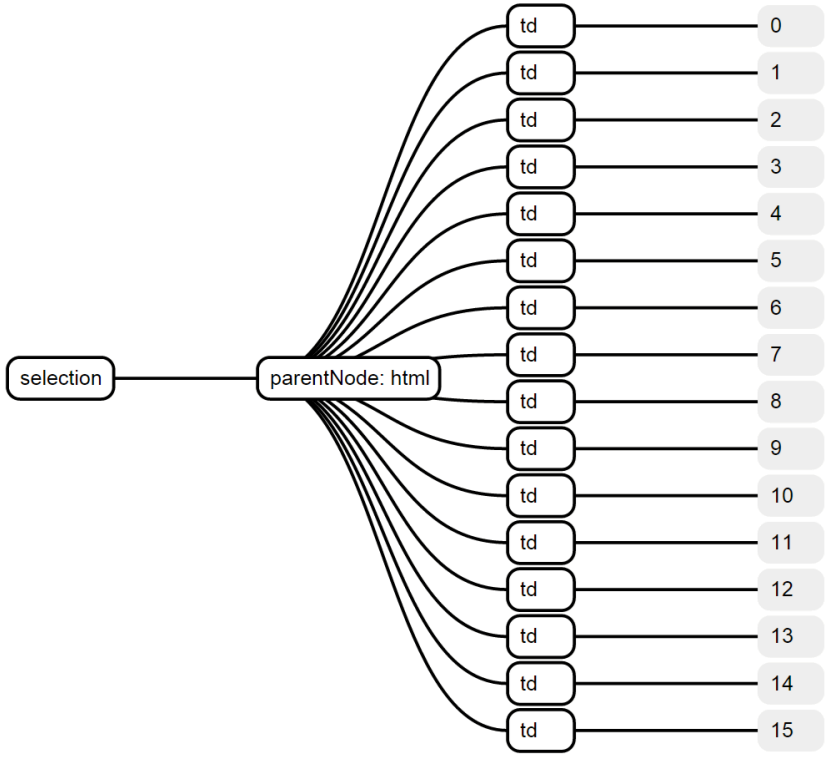
`var ps = document.getElementsByTagName("p");`
`for (var i = 0; i < ps.length; i++) {`
`  var p = ps.item(i);`
`  p.style.setProperty("color", "red", null);`
`}`
	W3C DOM

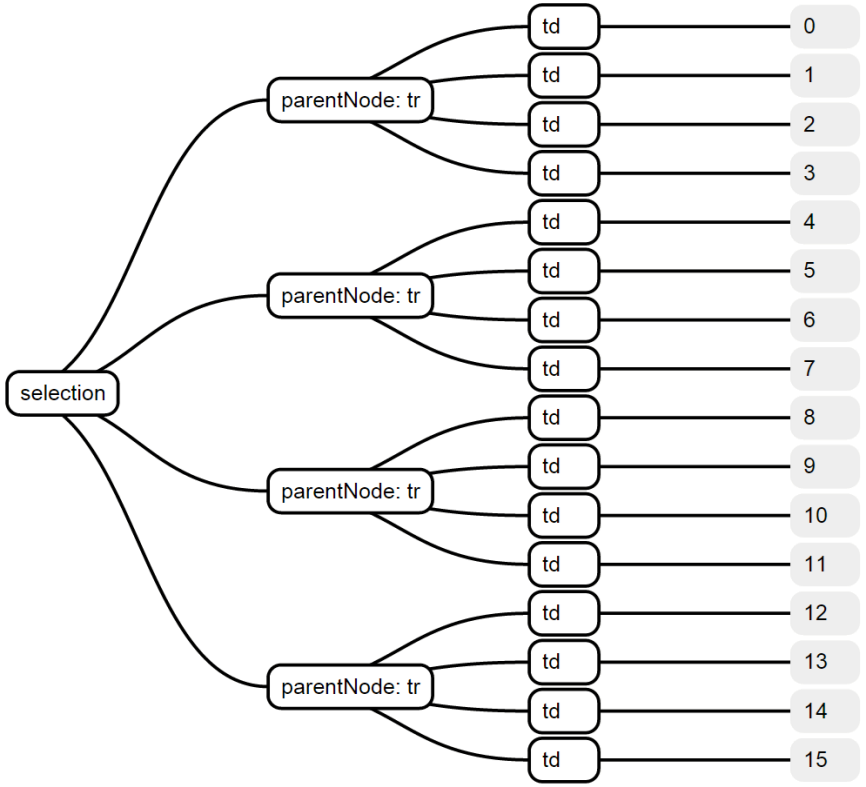
`p { color: red; }`	CSS
`$("p").css("color", "red");`	jQuery
`d3.selectAll("p").style("color", "red");`	D3

Nested selections

- Selection can be **hierarchical**

```
<table>
  <thead>
    <tr><td>  A</td><td>  B</td><td>  C</td><td>  D</td></tr>
  </thead>
  <tbody>
    <tr><td>  0</td><td>  1</td><td>  2</td><td>  3</td></tr>
    <tr><td>  4</td><td>  5</td><td>  6</td><td>  7</td></tr>
    <tr><td>  8</td><td>  9</td><td> 10</td><td> 11</td></tr>
    <tr><td> 12</td><td> 13</td><td> 14</td><td> 15</td></tr>
  </tbody>
</table>
```






```
var td = d3.selectAll("tbody td");
```

```
var td = d3.selectAll("tbody tr").selectAll("td");
```

Operators

- Having a **selection** allows one to **apply operators** to all selected elements
- Operators wrap the W3C DOM API
  - Setting **attributes** (`attr`), **styles** (`style`), **properties** (`property`), **HTML** (`html`) and **text** (`text`) content or **remove** (remove) elements
  - Operators can be **applied to the whole selections or**, since a selection is an array, over **given elements** (e.g., [0])
  - For most of the functions, D3 returns the modified object → operators **can be chained** since the next operator works on a changed object from the previous operator

https://github.com/d3/d3/blob/master/API.md#modifying-elements

https://github.com/d3/d3-selection/blob/v1.4.1/README.md#modifying-elements

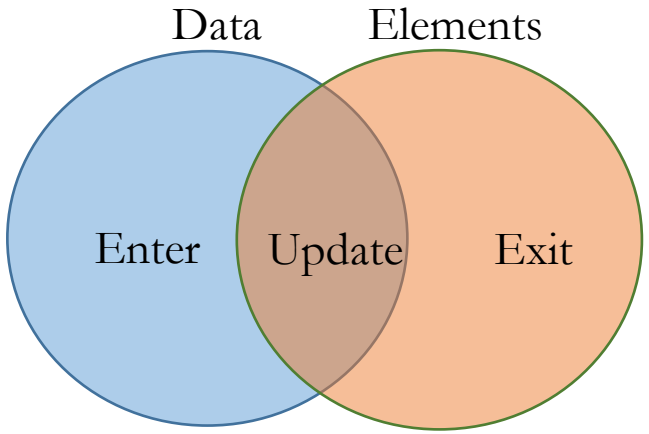
```
var svg = d3.select("body").append("svg:svg");
svg.append("svg:path")
   .attr("class", "line")
   .attr("d", d3.line().x(x).y(y));
```

```
d3.selectAll("p").style("color", "white");
```

```
d3.select("body").style("background-color");
```

```
d3.select("body").style("background-color", "black");
```

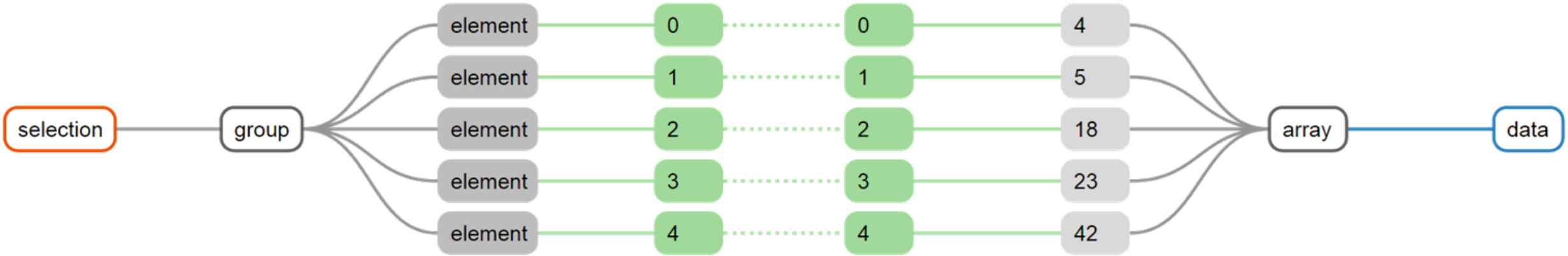
Data binding (1)



- Binding data to selected elements is called **data binding (data join)** → operator **`data`**
  - Data is joined to current selection by index (the first element to the first datum, …) or a user-defined key function
  - **Data** (blue) are **joined** with **DOM elements** (orange) forming three virtual subselections - **enter**, **update** and **exit**
    - `update` selection contains placeholders for existing elements, bound to data
    - `enter` selection contains placeholders for any missing elements
    - `exit` selection contains placeholders for elements not bound to data
    - The selections contain reference over which one can apply `append`, `insert` and `select` operators

https://github.com/d3/d3/blob/master/API.md#joining-data

https://github.com/d3/d3-selection/blob/v1.4.1/README.md#joining-data



How selections work

```
var width = 500;
var height = 500;

var data = [
  {pos: 250, color:'red' },
  {pos: 100, color:'green' },
  {pos: 150, color:'blue' }
];

var svgCanvas = d3.select('body')
.append('svg')
.attr('width', width)
.attr('height', height);

var circle1 = svgCanvas.append('circle')
.attr('cx', 25)
.attr('cy', 25)
.attr('r', 20)
.style('fill', 'black');
```

```
svgCanvas.selectAll('circle')
 .data(data)
 .attr('cx', function (d) { return d.pos; } )
 .enter().append('circle')
     .attr('cx', function (d) { return d.pos; } )
     .attr('cy', 25)
     .attr('r', 20 )
     .style('fill', function (d) { return
d.color; })
     .style('stroke', function (d) { return
d.color; })
```


http://jsfiddle.net/DavidHoksza/jzmm1o5g/

Data binding (2)

- Data is “sticky”
  - **Data** are **stored** in the `__data__` property of the bound element
  - Data are evaluated one by one – variable `i` denotes the index index of the data entry while `d` denotes the data element `(__data__)`
  - Once bound to nodes, it is available on subsequent re-selection without again requiring the `data` operator

Loading external resources

- Existing **routines** to load data from **CSV** (`d3.csv`), **TSV** (`d3.tsv`), **JSON** (`d3.json`), **XML** (`d3.xml`), **HTML** (`d3.html`) and **text** (`d3.text`)
  - Common interface
    - `d3.csv(pathToCsv, callback)`
    - `d3.csv.parse(csvString)`
    - `d3.csv.parseRows(csvString)`

```
// 1. Code here runs first, before the download starts.

d3.tsv("data.tsv").then(function(error, data) {
  // 3. Code here runs last, after the download finishes.
});

// 2. Code here runs second, while the file is downloading.
```

https://github.com/d3/d3-dsv

```
d3.csv(http://siret.ms.mff.cuni.cz/hoksza/teaching/vis/ds/gdp.csv).then(function (csvData) {

    var table = d3.select("body").append("table");

    var tr = table.selectAll("tr")
        .data(csvData)
        .enter()
            .append("tr");

    var td = tr.selectAll("td")
        .data(function (d) { return d3.values(d).splice(0,5); } )
        .enter()
            .append("td")
            .text(function (d) { return d; } );

});
```


http://jsfiddle.net/DavidHoksza/yzds7cqj/5/

SVG in D3.js

```
circleRadii = [40, 20, 10]

var svgContainer =
    d3.select("body").append("svg")
    .attr("width", 100)
    .attr("height", 100);

var circles =
svgContainer.selectAll("circle")
    .data(circleRadii)
    .enter()
    .append("circle")
```

```
var circleAttributes = circles
    .attr("cx", 50)
    .attr("cy", 50)
    .attr("r", function (d) { return d; })
    .style("fill", function(d, i) {
        var returnColor;
        if (i == 0) {
            returnColor = "green";
        } else if (i == 1) {
            returnColor = "purple";}
        else {
            returnColor = "red"; }
        return returnColor;
    });
```


http://jsfiddle.net/DavidHoksza/85Lj7jp2/

SVG shapes in D3.js

- Helper functions simplifying the construction of the SVG's *d* attributes from data → **path generators**

- Line → `d3.line, d3.lineRadial`
- Area → `d3.area, d3.areaRadial`
- Arc → `d3.arc`
- Symbol → `d3.symbol`
- …

https://github.com/d3/d3-shape

http://jsfiddle.net/DavidHoksza/ays78sjL/

Color

- Representations for various color spaces
  - RGB `d3.rgb(r, g, b), d3.rgb(color), rgb.brighter([k]), rgb.darker([k]), rgb.hsl()`
  - HSL `... hsl.rgb()`
  - HCL `... hcl.rgb()`
  - LAB `... lab.rgb()`

https://github.com/d3/d3-color

Scales (1)

- Data values need to be converted to pixel values → `d3-scale` module

- Axis can have different types, ranges and scales → function transforming input values to output values

  - $f: domain \rightarrow range$

    - *Domain* contains the data range of the variables (age,…), while *range* is the range in the target space(e.g. offset from the root SVG element or color)

Scales (2)

- Scale types
  - Quantitative – continuous input domains
    - scaleLinear, scaleIdentity, scalePow, scaleLog, scaleQuantize, scaleQuantile, scaleThreshold
  - Ordinal – discrete input domains
    - Basically an explicit mappings
    - Allows mapping into color range (`d3.scaleOrdinal(d3.schemeBlues[9]), …`)
  - Time – time domains
    - Extension of the linear time scale (`scaleTime`) using JavaScript `Date` object

https://github.com/d3/d3-scale

https://github.com/d3/d3-scale-chromatic

http://jsfiddle.net/DavidHoksza/6t8baonp/

Interpolators

- **Quantitative scales** support **interpolators**
  - Automatic detection of colors for RGB interpolation
  - String interpolation matches embedded numbers
  - Can be explicitly defined

```
var x = d3.scaleLinear()
    .domain([12, 24])
    .range(["steelblue", "brown"]);

x(16); // #666586
```

```
var x = d3.scaleLinear()
    .domain([12, 24])
    .range(["0px", "720px"]);

x(16); // 240px
```


http://jsfiddle.net/DavidHoksza/9ocy9dtc/

Arrays (1)

- D3's canonical data representation is an array
- Build on top of the JavaScript array manipulation functions
  - Mutator methods
    - array.pop, array.push, array.reverse, array.shift, array.sort, array.splice, array.unshift
  - Accessor methods
    - array.concat, array.join, array.slice, array.indexOf, array.lastIndexOf
  - Iteration methods
    - array.filter, array.forEach, array.every, array.some , array.map, array.reduce, array.reduceRight

Arrays (2)

- D3's helper functions
  - Ordering
    - `d3.ascending(a, b), d3. descending(a, b), d3.min(array), d3.max(array), d3.extent(array), d3.sum(array), d3.mean(array), d3.median(array), …`
  - Associative arrays (objects)
    - `d3.keys(object), d3.values(object), d3.entries(object)`

      `d3.entries({foo: 42, bar: true});`
  - Maps
  - Sets
  - Operators
    - `d3.merge(arrays), d3.range([start, ]stop[, step]), d3.zip(arrays…), …`

Arrays (3)

- **Nesting**
  - Allows grouping of an array into a hierarchy similar to **GROUP BY**, which is single-level
  - **Allows** hierarchical (**multi-level**) nesting
  - Leaf nodes of the tree can be sorted by value, while the internal nodes can be sorted by key

http://jsfiddle.net/DavidHoksza/bcaLcv8d/

Axis

- Axis labeling component → d3-axis → d3.axisLeft(), d3.axisTop(), d3.axisRight(), d3.axisBottom()
  - `scale()` - definition of scale on which the axis operates
  - `ticks()` - sets the ticks
  - `svg.append("g").call(xAxis)`

https://jsfiddle.net/DavidHoksza/fdrokgb8/

Animation and interaction

- `selection.on(type, listener)`
  - adds or removes an event listener (*click*, *mouseover*, *submit*, …) to each element in the current selection
  - `d`, `i` and `this` are passed to the listener
- `d3.event`
  - stores the current event, if any and implements the standard event fields like `timeStamp` and `keyCode`
- `d3.mouse`
  - x and y coordinates of the current d3.event
- `selection.transition`
  - registers a transition with given selection

http://jsfiddle.net/DavidHoksza/Lt99gjax/

Transitions

- Support for visualization of transformation from one state to another → transitions **interpolate values over time** (numbers, colors, geometric transforms, strings with embedded numbers)
- Transition is supported by the **transition interface** which is almost a mirror of the selection interface

```
d3.select("body").style("color", "red");
d3.select("body").transition().style("color", "red");
```

- See the example for details

http://jsfiddle.net/DavidHoksza/052hmbkw/

Behavior

- Support for dragging and zooming
  - Simplifies creation of event listeners to handle drag and zoom gestures

```
d3.drag()
        .on("start", dragstarted)
        .on("drag", dragged)
        .on("end", dragended);
```

```
d3.zoom()
        .scaleExtent([1, 10])
        .on("zoom", zoomed);
```


http://jsfiddle.net/DavidHoksza/rc543cjo/

Tooltips

- **Browser** tooltips
  - Title element
- **SVG** tooltips
  - Interactive creation (mouseover) and removal (mouseout) of an SVG text element
- **DIV** tooltips
  - CSS styled DIV element which can be positioned and hidden or showed based on the mouse position

http://jsfiddle.net/DavidHoksza/8r0odn9k/

Layouts

- Layouts remap provided data into new data suitable for given visualization

  - Chord
  - Cluster
  - Force
  - Histogram
  - Pack
  - Partition
  - Pie
  - Stack
  - Tree
  - Treemap

Pie layout

- `d3.pie(dataset)` – converts an array of numbers into array of objects containing *startAngle*, *endAngle* → `d3.arc()`

- The pie layout converts data into a suitable form to be bound (using data join) to SVG elements and used as `.attr('d', arc)` (see example)

http://jsfiddle.net/DavidHoksza/ghoLy9r5/

Treemap layout

- **Hierarchical** layout (all hierarchical layouts work similarly)
  - **Input**
    - Root node of the hierarchy
    - Either a JSON object or any hierarchical structure converted using the `children` and `value` accessor functions
  - **Output**
    - Array representing the computed positions (left upper corners and extents of the rectangles) of all nodes

Geography

- Based on GeoJSON format specification

- Consist of three submodules
  - **Path**
    - Given a geometry or feature object, it generates the path data string suitable for the "d" attribute of an SVG path element
  - **Projections**
    - Similarly as scales, projections convert domain objects (latitude and longitude) to pixel coordinates
  - **Streams**
    - Fast transformations of geometry without temporary copies of geometry objects

Big datasets

Using SVG is not possible with larger datasets → HTML canvas

SVG	Canvas
• 1,000 objects (more, when animation is not needed)	• More than 1,000 objects (but less then 100,000)
• CSS styles, animations	• DOM sees canvas as one object → visual objects can't be styled or capture events
• DOM events	

High level charting libraries

- https://nivo.rocks/
  - General-purpose charting library
- D3plus
  - Simple to use library
- crossfilter.js
  - Exploration of multivariate datasets
- dc.js
  - Library built on top of crossfilter.js for building interactive dashboards

Time series libraries

- Cubism.js
  - Real-time dashboards, data are incrementally fetched and rendered

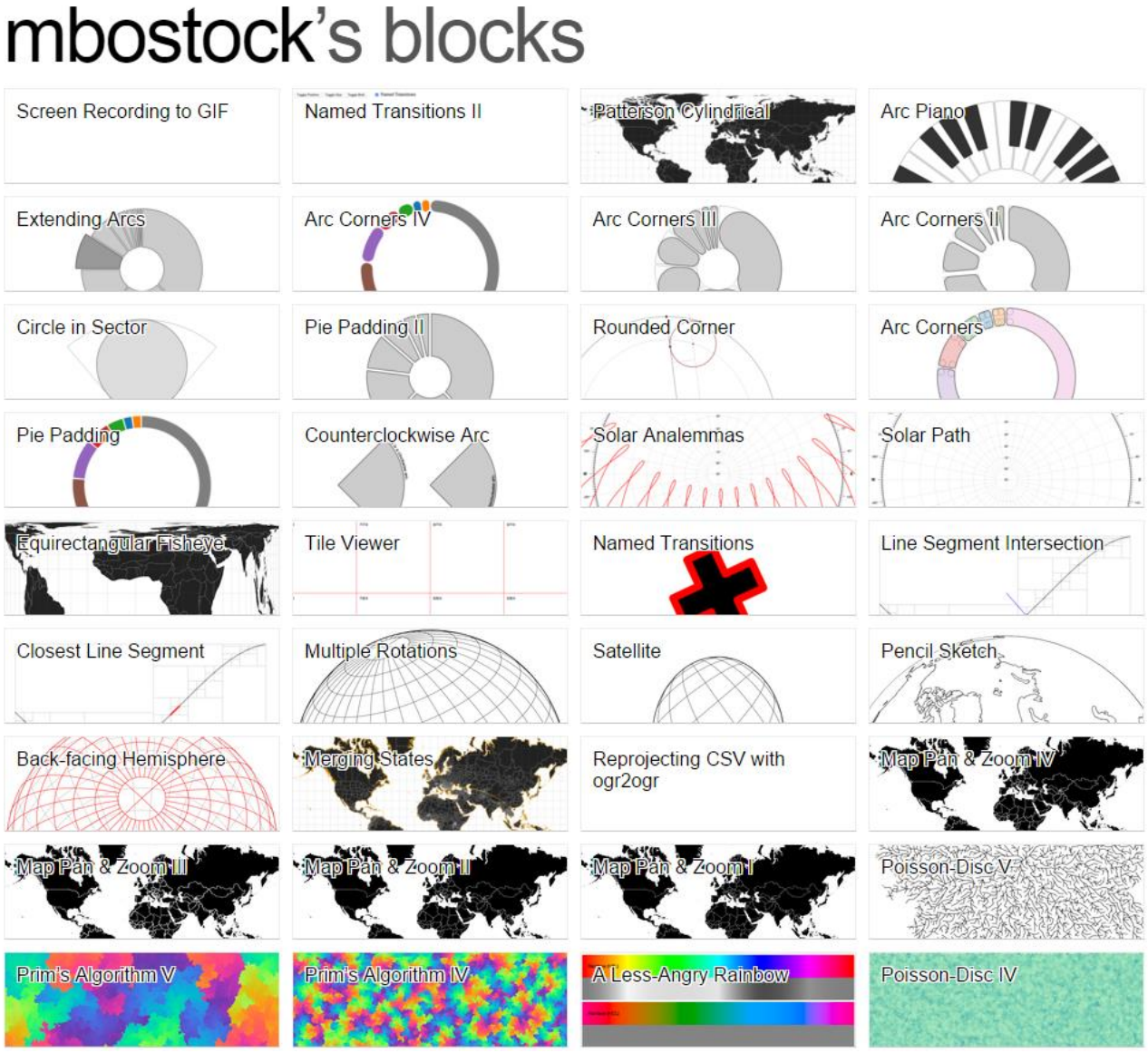
- Rickshaw
  - Unlike cubism keeps local copy of data
  - Interactive time series

Graph libraries

- sigma.js
  - Interactive visualization of both static and dynamic graphs
- arbor.js
  - Force-directed layout algorithms

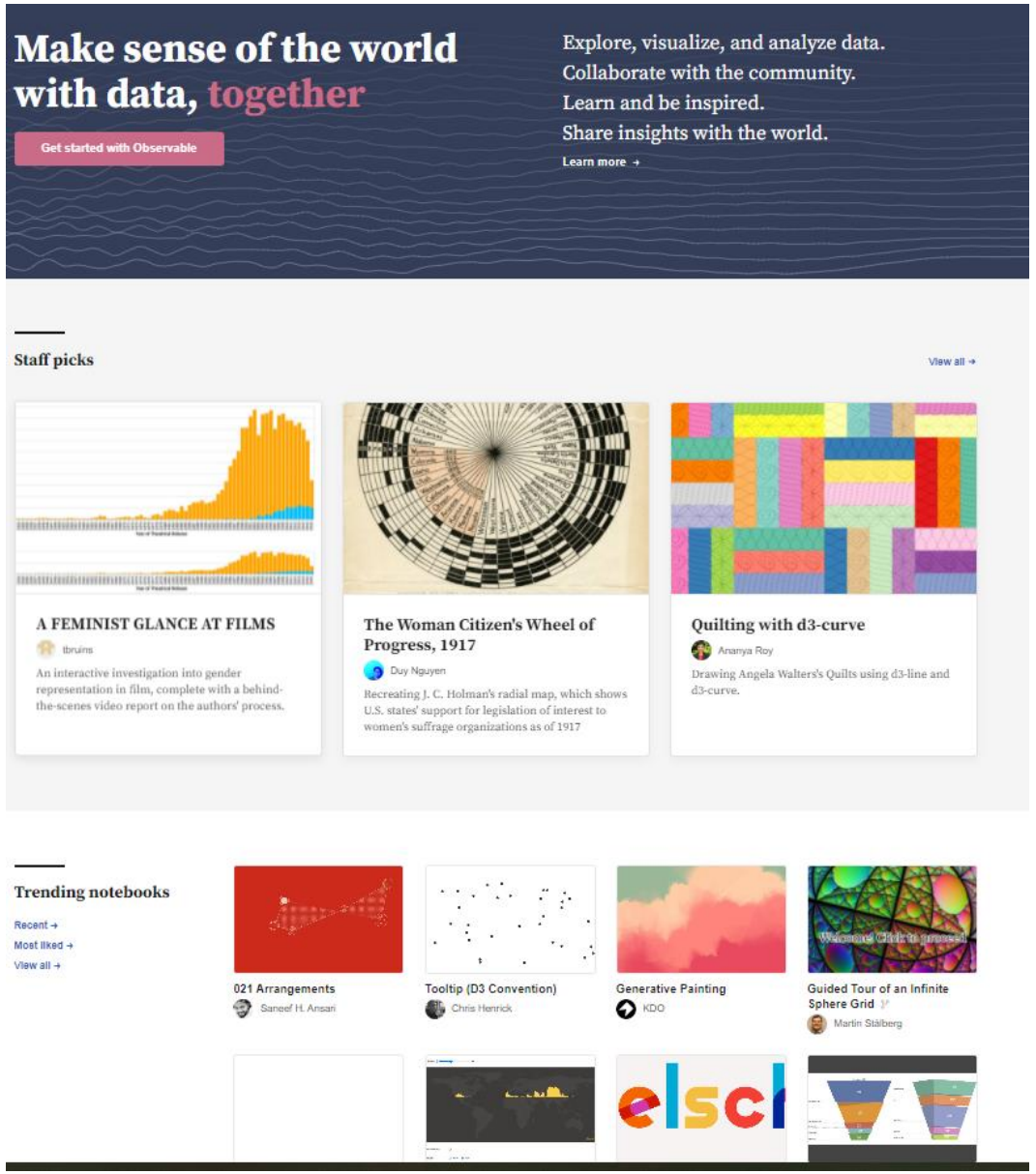
bl.ocks.org

- http://bl.ocks.org/mbostock
- Plenty of code examples



Playgrounds

- [CodePen]
- [JSFiddle]
- [Observable]
  - Notebooks



Sources

- Scott Murray (2013) **Interactive Data Visualization for the Web**. O'Reilly Media
- Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer (2011) **D^3 Data-Driven Documents**. IEEE Transactions on Visualization and Computer Graphics 17, 12 (December 2011), 2301-2309

- http://tutorials.jenkov.com/svg/index.html
- https://www.w3schools.com/graphics/svg_intro.asp

- https://github.com/d3/d3/wiki
- http://bost.ocks.org/mike/d3/workshop
- http://bost.ocks.org/mike/ → https://observablehq.com/@mbostock