Data Journalism

D3 - Part 2

Maps

Angelica Lo Duca
angelica.loduca@iit.cnr.it

Transitions

Transitions in D3

A transition is an animation from one form to another.

In D3 a transition starts with `.transition()` applied to a selection.

Example of Transition

Types of transitions

Duration

Ease

Delay

Duration

Specifies the animation duration in milliseconds for each element.

```
d3.select("#container")
    .transition()
    .duration(1000)
    .style("background-color", "red");
```

In this example, after 1 second the background color of the container is set to red.

Ease

Permits to specify and control the motion of the transition.

Types of ease

```
d3.select("#container")
    .transition()
    .ease(d3.easeLinear)
    .duration(2000)
    .attr("height",100)
```

Delay

Sets the delay parameter for each element in the selection on which the transition is applied. The transition will start after the specified delay value.

```
d3.select("#container")
    .transition()
     .ease(d3.easeLinear)
     .duration(2000)
     .delay(2000)
     .attr("height",100)
```

Delay VS Duration

Duration specifies how long the transition should run.

Delay is the time after the transition should start.

Example

Interactions

Interaction

Any activity triggered by the user:

Types of interactions

Mouse Events

Pan & Zoom

Drag and Drop

Brush

Mouse Events

mousedown

mouseup

click

dblclick

mouseover

mouseout

mouseenter

mouseleave

.on()

Adds an event listener for the specified type.

```
var listener = function(d){ // do some staff}

d3.selection.on("mousedown", listener);
```

.dispatch()

Dispatches a custom event of the specified type to each selected element.

For example, if you wanted to create an event dispatcher for "start" and "end" events, you can say:

```
var dispatch = d3.dispatch("start", "end");
```

Then, you can access the dispatchers for the different event types as dispatch.start and dispatch.end. For example, you might add an event listener:

```
dispatch.on("start", listener);
```

And then later dispatch an event to all registered listeners:

```
dispatch.call("start");
```

.event

Event object to access standard event fields such as timestamp.

The current position of the event:

`d3.event.pageX`

`d3.event.pageY`

.mouse(container)

Gets the x and y coordinates of the current mouse position in the specified DOM element.

```
var point = d3.mouse(this)

var p = {x: point[0], y: point[1] };
```

.touch(container)

Gets the touch coordinates to a container

Pan & Zoom, Drag & Drop and Brush

Composed of three parts:

Definition of the event with its properties

```
var zoom = d3.zoom(). [...] .on("zoom", my_event_listener);
```

Application to a specific selection

```
svg.call(zoom);
```

Definition of the event listener

```
function my_event_listener(){}
```

Pan & Zoom

Panning allows you to move around what you see.

Zooming allows you to expand or contract what you see.

Create a new zoom

```
var zoom = d3.zoom()
```

zoom.scaleExtent()

Permits to define zooming options

`zoom.scaleExtent([minimum, maximum])`

Specifies the zoom scale allowed range as a two-element array, **`[minimum, maximum]`**.

If not specified, returns the current scale extent, which defaults to **`[0, Infinity]`**.

zoom.translateExtent()

Permits to define panning options

```
zoom.translateExtent([[x0, y0], [x1, y1]])
```

[x0, y0] is the top-left corner of the world and

[x1, y1] is the bottom-right corner of the world

If the array of points is not specified, returns the current translate extent, which defaults to [[-∞, -∞], [+∞, +∞]].

zoom.extent()

Specifies the “area” involved in the zoom.

Default extent is `[[0, 0], [width, height]]`

Example

```
var zoom = d3.zoom()
      .scaleExtent([.5, 20])
      .extent([[5, 3], [width, height]])
      .on("zoom", my_event_listener);


svg.call(zoom);
```

Retrieve new scales

```
var x = d3.scaleLinear()
var xAxis = svg.append("g") . [...] .call(d3.axisBottom(x));
```

….

Within `my_function`

```
var newX = d3.event.transform.rescaleX(x);
xAxis.call(d3.axisBottom(newX))
```

`rescaleY` is used for the Y axis

Example (cont.)

```
function my_event_listener() {
    // retrieve the new scale
    var newX = d3.event.transform.rescaleX(x);
    var newY = d3.event.transform.rescaleY(y);

    // update axes with these new boundaries
    xAxis.call(d3.axisBottom(newX))
    yAxis.call(d3.axisLeft(newY))
```

Drag & Drop

Drag-and-drop is a popular and easy-to-learn pointing gesture: move the pointer to an object, press and hold to grab it, “drag” the object to a new location, and release to “drop”.

```
var drag = d3.drag();
```

.on(type, listener)

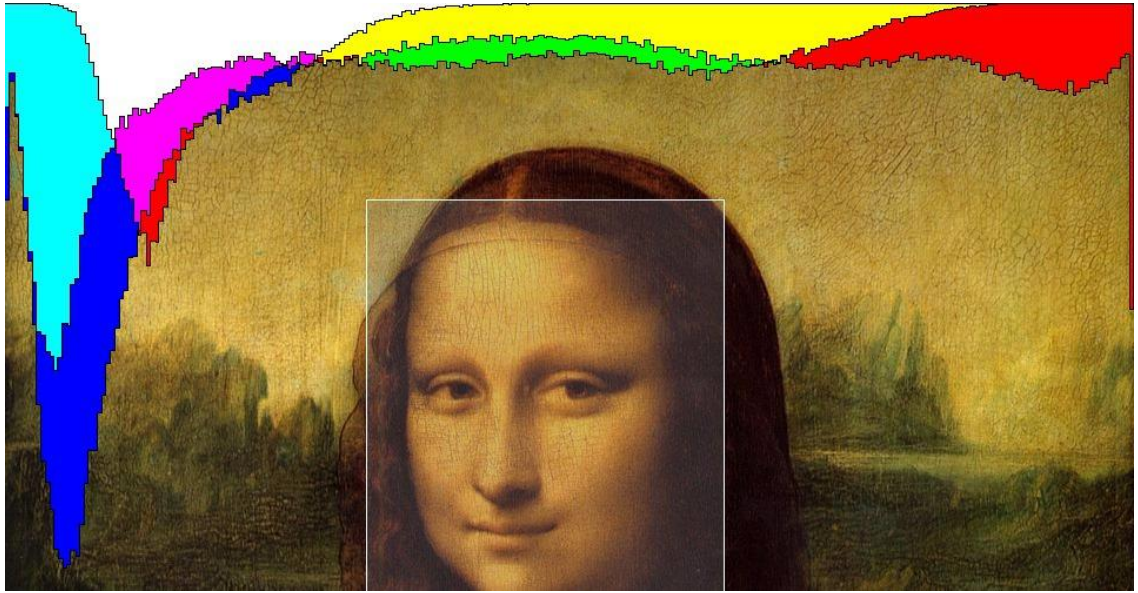
The type must be one of the following:

- **start** - after a new pointer becomes active (on mousedown or touchstart).
- **drag** - after an active pointer moves (on mousemove or touchmove).
- **end** - after an active pointer becomes inactive (on mouseup, touchend or touchcancel).

Drag Events

Brush

Brushing is the interactive specification a one- or two-dimensional selected region using a pointing gesture, such as by clicking and dragging the mouse.



Brush Event

```
var brush = d3.brush();
```

Definition of the listener

```
brush.on(type, listener)
```

The type must be one of the following:

- start - at the start of a brush gesture, such as on mousedown.
- brush - when the brush moves, such as on mousemove.
- end - at the end of a brush gesture, such as on mouseup.

Promises

Promises simplify the structure of asynchronous code,

A promise represents a value that is not yet known, but that will be known in the future. For example, when you load a file from a web server into a browser, the file's contents aren't available right away: the file must first be transferred over the network.

Rather than locking up while the file is downloading, browsers download asynchronously.

```
var promises = []
promises.push(d3.json("file1.json"))
promises.push(d3.csv("file2.csv"))
myDataPromises =
Promise.all(promises).then(function(data) {
    var json_data = data[0]
    var csv_data = data[1]

    // do some stuff
}
```

Maps

D3 mapping concepts

D3 requests vector geographic information in the form of GeoJSON and renders this to SVG or Canvas in the browser.

The 3 concepts that are key to understanding map creation using D3 are:

- **GeoJSON** - a JSON-based format for specifying geographic data
- **Projections** - functions that convert from latitude/longitude co-ordinates to x & y co-ordinates
- **Geographic path generators** - functions that convert GeoJSON shapes into SVG or Canvas paths

* source

Geographic Data

GeoJSON is a standard for representing geographic data using the JSON format and the full specification is at geojson.org.

TopoJSON is an extension of GeoJSON, which eliminates redundancy by storing relational information between geographic features.

GeoJSON

Each feature consists of

- **geometry** - simple polygons in the case of the countries.
- **properties** can contain any information about the feature such as name, id, and other data such as population, GDP etc.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

Maps download

https://geojson-maps.ash.ms/

https://observablehq.com/collection/@nitaku/official-italy-data

...

Projections

A projection function takes a longitude and latitude coordinate (in the form of an array [lon, lat]) and transforms it into an x and y coordinate.

Common projections:

- Azimuthal
- Composite
- Conic
- Cylindrical

Geographic Path Generator

A geographic path generator is a function that takes a GeoJSON object and converts it into an SVG path string. (In fact, it’s just another type of shape generator.)

We create a generator using the method **.geoPath()** and configure it with a projection function

Example

```
var projection = d3.geoEquirectangular();
var geoGenerator = d3.geoPath()
  .projection(projection);

var geoJson = {
  "type": "Feature",
  "properties": {
    "name": "Africa"
  },
  "geometry": {
    "type": "Polygon",
    "coordinates": [[[-6, 36], [33, 30], ... , [-6, 36]]]
  }
}

geoGenerator(geoJson);
```

Example (cont.)

```
// Join the features array to path elements
var u = d3.select('#content g.map')
  .selectAll('path')
  .data(geojson.features);

// Create path elements and update the d attribute using the geo generator
u.enter()
  .append('path')
  .attr('d', geoGenerator);
```

Shapes

Lines

Circles

Grid

Lines

If we need to add lines to a map we can achieve it by adding features to our GeoJSON.

Lines can be added as a **LineString** feature and will be projected into great-arcs (i.e. the shortest distance across the surface of the globe). Here's an example where we add a line between London and New York:

```
var line = {
  type: 'Feature',
  geometry: {
    type: 'LineString',
    coordinates: [[0.1278,
51.5074], [-74.0059, 40.7128]]
  }
}

geoGenerator(line)
```

Circles

Circle features can be generated using `d3.geoCircle()`.

Typically the center ([lon, lat]) and the angle (degrees) between the points are set:

```
var circle = d3.geoCircle()
  .center([0.1278, 51.5074])
  .radius(5);

// returns a GeoJSON object
// representing a circle
circle();

// returns a path string
//representing the projected circle
geoGenerator(circle());
```

Grid

A GeoJSON grid of longitude and latitude lines (known as a graticule) can be generated using **`d3.graticule()`**

```
var graticule = d3.geoGraticule();

// returns a GeoJSON object
//representing the graticule
graticule();

// returns a path string
//representing the projected
// graticule
geoGenerator(graticule());
```

Types of maps

Choropleth maps

Heat maps

Proportional symbol maps

Dot density maps

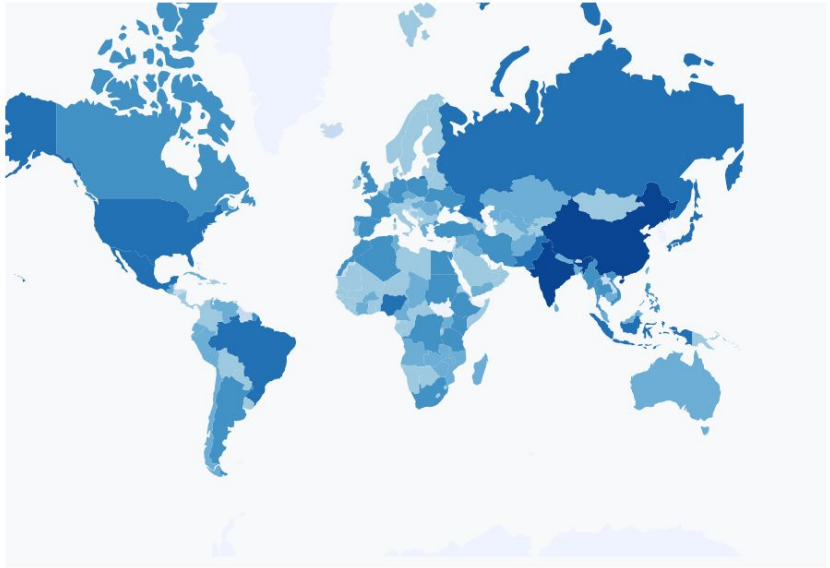
Animated time-series maps

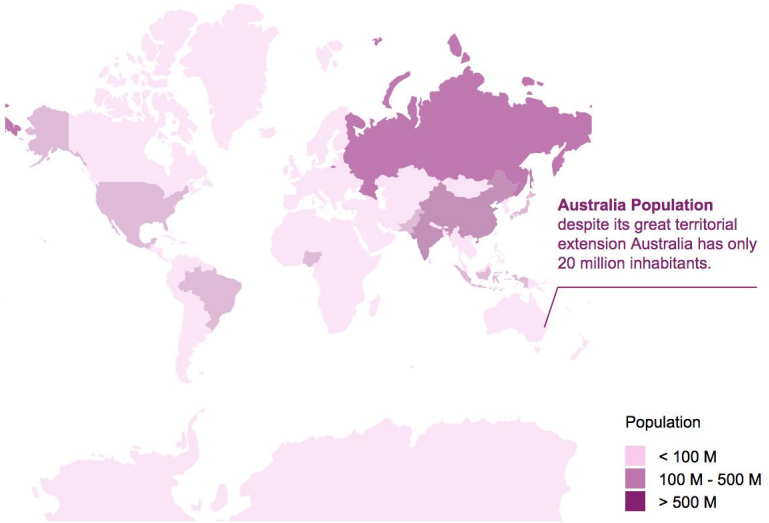
...

* source

Choropleth maps - Example

[What to consider when creating choropleth maps]





References

https://www.tutorialsteacher.com/d3js

https://datawanderings.com/2018/10/28/making-a-map-in-d3-js-v-5/

https://coolors.co/gradient-palette/f8caee-852170?number=3

https://d3-legend.susielu.com/