Kostenloses eBook

LERNEN

d3.js

Free unaffiliated eBook created from
Stack Overflow contributors.

#d3.js

Inhaltsverzeichnis

Über

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: d3-js

It is an unofficial and free d3.js ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official d3.js.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Kapitel 1: Erste Schritte mit d3.js

Bemerkungen

D3.js ist eine JavaScript-Bibliothek zur Bearbeitung von Dokumenten, die auf Daten basieren. D3 hilft Ihnen, Daten mithilfe von HTML, SVG und CSS zum Leben zu erwecken. D3s Schwerpunkt auf Webstandards bietet Ihnen alle Möglichkeiten moderner Browser, ohne sich an ein proprietäres Framework zu binden, indem Sie leistungsstarke Visualisierungskomponenten und einen datengesteuerten Ansatz für die DOM-Manipulation kombinieren.

Die offizielle Website: https://d3js.org/ Viele Beispiele hier: http://bl.ocks.org/mbostock

Versionen

Ausführung	Veröffentlichungsdatum
v1.0.0	2011-02-11
2,0	2011-08-23
3.0 "Baja"	2012-12-21
v4.0.0	2016-06-28
v4.1.1	2016-07-11
v4.2.1	2016-08-03
v4.2.2	2016-08-16
v4.2.3	2016-09-13
v4.2.4	2016-09-19
v4.2.5	2016-09-20
v4.2.6	2016-09-22
v4.2.7	2016-10-11
v4.2.8	2016-10-20
v4.3.0	2016-10-27

Examples

Installation

Es gibt verschiedene Möglichkeiten, D3 herunterzuladen und zu verwenden.

Direkter Skript-Download

1. Laden Sie d3.zip herunter und entpacken Sie es
2. Kopieren Sie den Ordner, in den Sie die Abhängigkeiten Ihres Projekts beibehalten möchten
3. Verweisen Sie auf d3.js (für die Entwicklung) oder d3.min.js (für die Produktion) in Ihrem HTML-Code: `<script type="text/javascript" src="scripts/d3/d3.js"></script>`

NPM

1. Initialisieren Sie NPM in Ihrem Projekt, wenn Sie dies noch nicht getan haben: `npm init`
2. NPM install D3: `npm install --save d3`
3. Verweisen Sie auf d3.js (für Entwicklung) oder d3.min.js (für Produktion) in Ihrem HTML-Code: `<script type="text/javascript" src="node_modules/d3/build/d3.js"></script>`

CDN

1. Verweisen Sie auf d3.js (für die Entwicklung) oder d3.min.js (für die Produktion) in Ihrem HTML-Code: `<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/d3/4.1.1/d3.js"></script>`

GITHUB

1. Laden Sie eine beliebige Version von d3.js (für die Entwicklung) oder d3.min.js (für die Produktion) von Github herunter: `<script type="text/javascript" src="https://raw.githubusercontent.com/d3/d3/v3.5.16/d3.js"></script>`

Um direkt auf die neueste Version zu verlinken, kopieren Sie dieses Snippet:

```
<script src="https://d3js.org/d3.v4.min.js"></script>
```

Einfaches Balkendiagramm

index.html

```
<!doctype html>
<html>
  <head>
    <title>D3 Sample</title>
  </head>
  <body>
    <!-- This will serve as a container for our chart. This does not have to be a div, and can
in fact, just be the body if you want. -->
    <div id="my-chart"></div>

    <!-- Include d3.js from a CDN. -->
```

```
    <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/d3/4.1.1/d3.js"></script>

    <!-- Include our script that will make our bar chart. -->
    <script type="text/javascript" src="chart.js"></script>
  </body>
</html>
```

chart.js

```
// Sample dataset. In a real application, you will probably get this data from another source
such as AJAX.
var dataset = [5, 10, 15, 20, 25]

// Sizing variables for our chart. These are saved as variables as they will be used in
calculations.
var chartWidth = 300
var chartHeight = 100
var padding = 5

// We want our our bars to take up the full height of the chart, so, we will apply a scaling
factor to the height of every bar.
var heightScalingFactor = chartHeight / getMax(dataset)

// Here we are creating the SVG that will be our chart.
var svg = d3
  .select('#my-chart')           // I'm starting off by selecting the container.
    .append('svg')               // Appending an SVG element to that container.
    .attr('width', chartWidth)   // Setting the width of the SVG.
    .attr('height', chartHeight) // And setting the height of the SVG.

// The next step is to create the rectangles that will make up the bars in our bar chart.
svg
  .selectAll('rect')                                          // I'm selecting all of the
rectangles in the SVG (note that at this point, there actually aren't any, but we'll be
creating them in a couple of steps).
  .data(dataset)                                              // Then I'm mapping the dataset
to those rectangles.
  .enter()                                                    // This step is important in
that it allows us to dynamically create the rectangle elements that we selected previously.
    .append('rect')                                           // For each element in the
dataset, append a new rectangle.
      .attr('x', function (value, index) {                    // Set the X position of the
rectangle by taking the index of the current item we are creating, multiplying it by the
calculated width of each bar, and adding a padding value so we can see some space between
bars.
          return (index * (chartWidth / dataset.length)) + padding
        })
      .attr('y', function (value, index) {                    // Set the rectangle by
subtracting the scaled height from the height of the chart (this has to be done becuase SVG
coordinates start with 0,0 at their top left corner).
        return chartHeight - (value * heightScalingFactor)
      })
      .attr('width', (chartWidth / dataset.length) - padding) // The width is dynamically
calculated to have an even distribution of bars that take up the entire width of the chart.
      .attr('height', function (value, index) {               // The height is simply the
value of the item in the dataset multiplied by the height scaling factor.
        return value * heightScalingFactor
      })
```

```
      .attr('fill', 'pink')                                    // Sets the color of the bars.

/**
 *  Gets the maximum value in a collection of numbers.
 */
function getMax(collection) {
  var max = 0

  collection.forEach(function (element) {
    max = element > max ? element : max
  })

  return max
}
```

Beispielcode verfügbar unter https://github.com/dcsinnovationlabs/D3-Bar-Chart-Example

Demo verfügbar unter https://dcsinnovationlabs.github.io/D3-Bar-Chart-Example/

Hallo Welt!

Erstellen Sie eine `.html` Datei, die dieses Snippet enthält:

```
<!DOCTYPE html>
<meta charset="utf-8">
<body>
<script src="//d3js.org/d3.v4.min.js"></script>
<script>

d3.select("body").append("span")
    .text("Hello, world!");

</script>
```

Sehen Sie diesen Ausschnitt in Aktion bei diesem JSFiddle .

Was ist D3? Datengesteuerte Dokumente.

Wir sind so auf den Namen verwendet **D3.js** , dass es möglich ist, zu vergessen , dass D3 **DDD** ist eigentlich **(D** ATA- **D** riven **D** OKUMENTE). Und das ist, was D3 gut tut, ein datengesteuerter Ansatz für die DOM-Manipulation (Document Object Model): D3 bindet Daten an DOM-Elemente und manipuliert diese Elemente basierend auf den gebundenen Daten.

Lassen Sie uns in diesem Beispiel eine sehr grundlegende Funktion von D3 sehen. Hier werden keine SVG-Elemente angehängt. Stattdessen verwenden wir eine bereits auf der Seite vorhandene SVG-Datei, etwa wie folgt:

```
<svg width="400" height="400">
    <circle cx="50" cy="50" r="10"></circle>
    <circle cx="150" cy="50" r="10"></circle>
    <circle cx="210" cy="320" r="10"></circle>
    <circle cx="210" cy="30" r="10"></circle>
    <circle cx="180" cy="200" r="10"></circle>
</svg>
```

Dies ist eine ziemlich einfache SVG mit 5 Kreisen. Im Moment sind diese Kreise an keine Daten "gebunden". Lasst uns diese letzte Behauptung überprüfen:

In unserem Code schreiben wir:

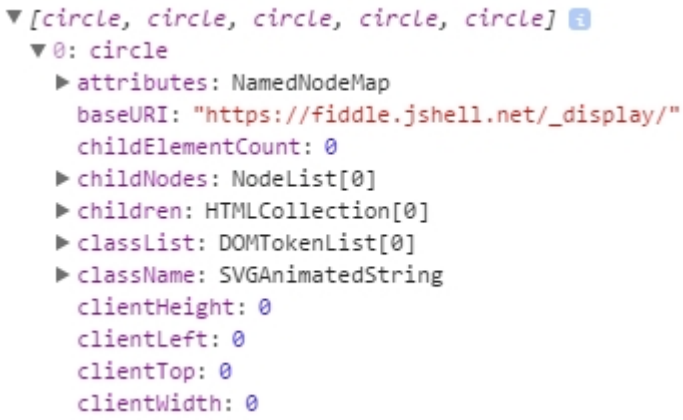
```
var svg = d3.select("svg");
var circles = svg.selectAll("circle");
console.log(circles.nodes());
```

Hier gibt `d3.select("svg")` ein d3-Objekt zurück, das das Tag `<svg width="400" height="400"></svg>` und alle untergeordneten Tags, die `<circle>` s, enthält. Wenn mehrere `svg` Tags auf der Seite vorhanden sind, wird nur das erste ausgewählt. Wenn Sie dies nicht möchten, können Sie auch die Tag-ID auswählen, wie zum Beispiel `d3.select("#my-svg")` . Das d3-Objekt verfügt über integrierte Eigenschaften und Methoden, die wir später verwenden werden.

`svg.selectAll("circle")` erstellt ein Objekt aus allen `<circle></circle>` -Elementen innerhalb des `<svg>` -Tags. Es kann mehrere Ebenen durchsuchen, es spielt also keine Rolle, ob die Tags direkte Kinder sind.

`circles.nodes()` gibt die Kreistags mit ihren Eigenschaften zurück.

Wenn wir uns die Konsole ansehen und den ersten Kreis auswählen, werden wir so etwas sehen:



Zuerst haben wir `attributes` , dann `childNodes` , dann `children` und so weiter ... aber keine Daten.

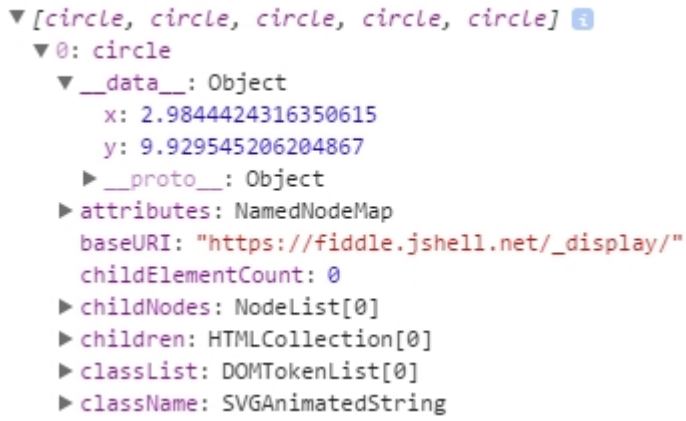
Lassen Sie uns einige Daten binden

Was passiert aber, wenn wir *Daten* an diese DOM-Elemente binden?

In unserem Code gibt es eine Funktion, die ein Objekt mit zwei Eigenschaften (`x` und `y`) mit numerischen Werten erstellt (dieses Objekt befindet sich innerhalb eines Arrays, überprüfen Sie die Geige unten). Wenn wir diese Daten an die Kreise binden ...

```
circles.data(data);
```

Das werden wir sehen, wenn wir die Konsole inspizieren:



Wir haben etwas Neues vor `attributes` ! Etwas namens `__data__` ... und schau: Die Werte von `x` und `y` sind da!

Wir können zum Beispiel die Position der Kreise basierend auf diesen Daten ändern. Schauen Sie sich diese Geige an .

Dies ist, was D3 am besten kann: Daten an DOM-Elemente binden und diese DOM-Elemente basierend auf den gebundenen Daten bearbeiten.

Einfaches D3-Diagramm: Hallo Welt!

Fügen Sie diesen Code in eine leere HTML-Datei ein und führen Sie ihn in Ihrem Browser aus.

```
<!DOCTYPE html>

<body>

<script src="https://d3js.org/d3.v4.js"></script>    <!-- This downloads d3 library -->

<script>
//This code will visualize a data set as a simple scatter chart using d3. I omit axes for
simplicity.
var data = [        //This is the data we want to visualize.
                    //In reality it usually comes from a file or database.
  {x: 10,    y: 10},
  {x: 10,    y: 20},
  {x: 10,    y: 30},
  {x: 10,    y: 40},
  {x: 10,    y: 50},
  {x: 10,    y: 80},
  {x: 10,    y: 90},
  {x: 10,    y: 100},
  {x: 10,    y: 110},
  {x: 20,    y: 30},
  {x: 20,    y: 120},
  {x: 30,    y: 10},
  {x: 30,    y: 20},
  {x: 30,    y: 30},
  {x: 30,    y: 40},
  {x: 30,    y: 50},
  {x: 30,    y: 80},
  {x: 30,    y: 90},
  {x: 30,    y: 100},
  {x: 30,    y: 110},
```

```
{x: 40,    y: 120},
{x: 50,    y: 10},
{x: 50,    y: 20},
{x: 50,    y: 30},
{x: 50,    y: 40},
{x: 50,    y: 50},
{x: 50,    y: 80},
{x: 50,    y: 90},
{x: 50,    y: 100},
{x: 50,    y: 110},
{x: 60,    y: 10},
{x: 60,    y: 30},
{x: 60,    y: 50},
{x: 70,    y: 10},
{x: 70,    y: 30},
{x: 70,    y: 50},
{x: 70,    y: 90},
{x: 70,    y: 100},
{x: 70,    y: 110},
{x: 80,    y: 80},
{x: 80,    y: 120},
{x: 90,    y: 10},
{x: 90,    y: 20},
{x: 90,    y: 30},
{x: 90,    y: 40},
{x: 90,    y: 50},
{x: 90,    y: 80},
{x: 90,    y: 120},
{x: 100,    y: 50},
{x: 100,    y: 90},
{x: 100,    y: 100},
{x: 100,    y: 110},
{x: 110,    y: 50},
{x: 120,    y: 80},
{x: 120,    y: 90},
{x: 120,    y: 100},
{x: 120,    y: 110},
{x: 120,    y: 120},
{x: 130,    y: 10},
{x: 130,    y: 20},
{x: 130,    y: 30},
{x: 130,    y: 40},
{x: 130,    y: 50},
{x: 130,    y: 80},
{x: 130,    y: 100},
{x: 140,    y: 50},
{x: 140,    y: 80},
{x: 140,    y: 100},
{x: 140,    y: 110},
{x: 150,    y: 50},
{x: 150,    y: 90},
{x: 150,    y: 120},
{x: 170,    y: 20},
{x: 170,    y: 30},
{x: 170,    y: 40},
{x: 170,    y: 80},
{x: 170,    y: 90},
{x: 170,    y: 100},
{x: 170,    y: 110},
{x: 170,    y: 120},
{x: 180,    y: 10},
```

```
  {x: 180,    y: 50},
  {x: 180,    y: 120},
  {x: 190,    y: 10},
  {x: 190,    y: 50},
  {x: 190,    y: 120},
  {x: 200,    y: 20},
  {x: 200,    y: 30},
  {x: 200,    y: 40},
  {x: 210,    y: 80},
  {x: 210,    y: 90},
  {x: 210,    y: 100},
  {x: 210,    y: 110},
  {x: 210,    y: 120},
  {x: 220,    y: 80},
  {x: 220,    y: 120},
  {x: 230,    y: 80},
  {x: 230,    y: 120},
  {x: 240,    y: 90},
  {x: 240,    y: 100},
  {x: 240,    y: 110},
  {x: 270,    y: 70},
  {x: 270,    y: 80},
  {x: 270,    y: 90},
  {x: 270,    y: 100},
  {x: 270,    y: 120}
];

//The following code chains a bunch of methods. Method chaining is what makes d3 very simple
and concise.
d3.select("body").append("svg").selectAll()  //'d3' calls the d3 library
                                             //'.select' selects the object (in this case the
body of HTML)
                                             //'.append' adds SVG element to the body
                                             //'.selectAll()' selects all SVG elements
    .data(data)                              //'.data' gets the data from the variable 'data'
  .enter().append("circle")                  //'.enter' enters the data into the SVG
                                             //the data enter as circles with
'.append("circle")'
    .attr("r", 3)                            //'.attr' adds/alters atributes of SVG,
                                             //such as radius ("r"), making it 3 pixels
    .attr("cx", function(d) { return d.x; }) //coordinates "cx" (circles' x coordinates)
    .attr("cy", function(d) { return d.y; }) //coordinates "cy" (circles' y coordinates)
    .style("fill", "darkblue");              //'.style' changes CSS of the SVG
                                             //in this case, fills circles with "darkblue"
color

</script>
```

Hier ist ein JSFiddle des Charts.

Sie können die bereits erstellte HTML-Datei auch von GitHub herunterladen.

Der nächste Schritt beim Lernen von d3 besteht darin, dem Lernprogramm von Mike Bostock (dem Entwickler des d3) zu folgen, um ein Balkendiagramm von Grund auf zu erstellen.

Erste Schritte mit d3.js online lesen: https://riptutorial.com/de/d3-js/topic/876/erste-schritte-mit-d3-js

Kapitel 2: Aktualisierungsmuster

Syntax

- selection.enter ()
- selection.exit ()
- selection.merge ()

Examples

Aktualisieren der Daten: Ein einfaches Beispiel für die Eingabe, Aktualisierung und Beendigung von Auswahlen

Das Erstellen eines Diagramms mit einem statischen Datensatz ist relativ einfach. Wenn wir zum Beispiel dieses Array von Objekten als Daten haben:

```
var data = [
    {title: "A", value: 53},
    {title: "B", value: 12},
    {title: "C", value: 91},
    {title: "D", value: 24},
    {title: "E", value: 59}
];
```

Wir können ein Balkendiagramm erstellen, in dem jeder Balken eine Kennzahl namens "Titel" darstellt und deren Breite den Wert dieser Kennzahl darstellt. Da sich dieser Datensatz nicht ändert, enthält unser Balkendiagramm nur eine Auswahl "Enter":

```
var bars = svg.selectAll(".bars")
    .data(data);

bars.enter()
    .append("rect")
    .attr("class", "bars")
    .attr("x", xScale(0))
    .attr("y", function(d){ return yScale(d.title)})
    .attr("width", 0)
    .attr("height", yScale.bandwidth())
    .transition()
    .duration(1000)
    .delay(function(d,i){ return i*200})
    .attr("width", function(d){ return xScale(d.value) - margin.left});
```

Hier setzen wir die Breite jedes Balkens auf 0 und nach dem Übergang auf den endgültigen Wert.

Diese Auswahl allein reicht aus, um unser Diagramm zu erstellen, das Sie in dieser Geige sehen können .

Was aber, wenn sich meine Daten ändern?

In diesem Fall müssen wir unseren Chart dynamisch ändern. Am besten erstellen Sie eine Auswahl für "Eingabe", "Aktualisierung" und "Beenden". Vorher müssen wir jedoch einige Änderungen am Code vornehmen.

Zuerst verschieben wir die sich ändernden Teile in eine Funktion namens `draw()` :

```
function draw(){
    //changing parts
};
```

Diese "wechselnden Teile" umfassen:

1. Die Eingabe-, Aktualisierungs- und Beendigungsauswahl;
2. Die Domäne jeder Skala;
3. Der Übergang der Achse;

Innerhalb dieser `draw()` Funktion rufen wir eine andere Funktion auf, die unsere Daten erstellt. Hier handelt es sich lediglich um eine Funktion, die ein Array von 5 Objekten zurückgibt. Dabei werden zufällig 5 von 10 Buchstaben (alphabetisch sortiert) und für jedes Objekt ein Wert zwischen 0 und 99 ausgewählt:

```
function getData(){
    var title = "ABCDEFGHIJ".split("");
    var data = [];
    for(var i = 0; i < 5; i++){
        var index = Math.floor(Math.random()*title.length);
        data.push({title: title[index],
            value: Math.floor(Math.random()*100)});
        title.splice(index,1);
    }
    data = data.sort(function(a,b){ return d3.ascending(a.title,b.title)});
    return data;
};
```

Und nun zu unserer Auswahl. Aber vorher noch ein Wort der Vorsicht: Um die so genannte *Objektkonstanz zu erhalten* , müssen wir eine Schlüsselfunktion als zweites Argument für selection.data angeben:

```
var bars = svg.selectAll(".bars")
    .data(data, function(d){ return d.title});
```

Ohne das wechseln unsere Balken nicht reibungslos und es wäre schwierig, den Änderungen in der Achse zu folgen (Sie können sehen, dass das zweite Argument unten in der Geige entfernt wird).

Wenn wir also die `var bars` richtig eingestellt haben, können wir unsere Auswahl treffen. Dies ist die Ausgangsauswahl:

```
bars.exit()
    .transition()
    .duration(1000)
```

```
    .attr("width", 0)
    .remove();
```

Dies sind die Eingaben und die Aktualisierungsauswahlen (in D3 v4.x wird die Aktualisierungsauswahl mit der Eingabeauswahl durch `merge`):

```
bars.enter()//this is the enter selection
    .append("rect")
    .attr("class", "bars")
    .attr("x", xScale(0) + 1)
    .attr("y", function(d){ return yScale(d.title)})
    .attr("width", 0)
    .attr("height", yScale.bandwidth())
    .attr("fill", function(d){ return color(letters.indexOf(d.title)+1)})
    .merge(bars)//and from now on, both the enter and the update selections
    .transition()
    .duration(1000)
    .delay(1000)
    .attr("y", function(d){ return yScale(d.title)})
    .attr("width", function(d){ return xScale(d.value) - margin.left});
```

Zum Schluss rufen wir die `draw()` -Funktion jedes Mal auf, wenn auf die Schaltfläche geklickt wird:

```
d3.select("#myButton").on("click", draw);
```

Und dies ist die Geige, die all diese drei Auswahlen in Aktion zeigt.

Auswahl zusammenführen

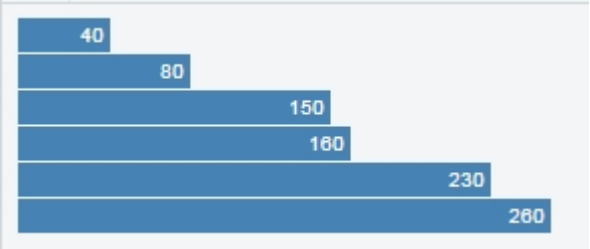
Das Aktualisierungsmuster in D3 Version 3

Ein korrektes Verständnis der Funktionsweise der Auswahlmöglichkeiten "Eintreten", "Aktualisieren" und "Beenden" ist für das korrekte Ändern des Dataviz mit D3 von grundlegender Bedeutung.

Seit D3 Version 3 (eigentlich seit Version 2) konnte dieses Snippet die Übergänge sowohl für die Eingabe- als auch für die Aktualisierungsauswahl festlegen (Live-Demo hier):

```
var divs = body.selectAll("div")
    .data(data);//binding the data

divs.enter()//enter selection
    .append("div")
    .style("width", "0px");

divs.transition()
    .duration(1000)
    .style("width", function(d) { return d + "px"; })
    .attr("class", "divchart")
    .text(function(d) { return d; });
```

Dieses Ergebnis nach dem Übergang geben:



Aber was passiert mit genau demselben Code, wenn wir D3 Version 4 verwenden? Sie können es in dieser Live-Demo sehen : *nichts* !

Warum?

Änderungen im Update-Pattern in D3 Version 4

Lassen Sie uns den Code überprüfen. Zuerst haben wir `divs` . Diese Auswahl bindet die Daten an das `<div>` .

```
var divs = body.selectAll("div")
    .data(data);
```

Dann haben wir `divs.enter()` , eine Auswahl, die alle Daten mit nicht übereinstimmenden Elementen enthält. Diese Auswahl enthält alle Divs beim ersten Aufruf der Funktion `draw` , und wir setzen deren Breite auf Null.

```
divs.enter()
    .append("div")
    .style("width", "0px");
```

Dann haben wir `divs.transition()` , und hier haben wir das interessante Verhalten: In D3 Version 3 `divs.transition()` dass alle `<div>` in der Auswahl "enter" auf ihre endgültige Breite `divs.transition()` . Das macht aber keinen Sinn! `divs` enthält nicht die Eingabe- `divs` und sollte kein DOM-Element ändern.

Es gibt einen Grund, warum dieses seltsame Verhalten in D3 Version 2 und 3 (Quelle hier) eingeführt wurde, und es wurde in D3 Version 4 "korrigiert". In der obigen Live-Demo passiert also nichts, und dies wird erwartet! Wenn Sie auf die Schaltfläche klicken, werden außerdem alle vorherigen sechs Balken angezeigt, da sie sich jetzt in der Auswahl „Aktualisieren“ befinden und nicht mehr in der Auswahl „Eingeben“.

Für den Übergang, der über die Auswahl "enter" wirkt, müssen wir separate Variablen erstellen oder die Auswahl einfacher zusammenführen:

```
divs.enter()//enter selection
    .append("div")
    .style("width", "0px")
    .merge(divs)//from now on, enter + update selections
    .transition().duration(1000).style("width", function(d) { return d + "px"; })
    .attr("class", "divchart")
    .text(function(d) { return d; });
```

Jetzt haben wir die Auswahlen "enter" und "update" zusammengeführt. Sehen Sie, wie es in dieser Live-Demo funktioniert.

Aktualisierungsmuster online lesen: https://riptutorial.com/de/d3-js/topic/5749/aktualisierungsmuster

Kapitel 3: Ansätze zum Erstellen von responsiven d3.js-Diagrammen

Syntax

- var width = document.getElementById ('chartArea'). clientWidth;
- Var Höhe = Breite / 3,236;
- window.onresize = resizeFunctionCall;

Examples

Bootstrap verwenden

Ein Ansatz, der häufig verwendet wird, ist die Verwendung des gerasterten Frameworks von bootstrap , um den Bereich zu definieren, in dem das Diagramm vorhanden ist. Wenn Sie dies in Verbindung mit der Variable `clientWidth` und dem `window.onresize` , ist es sehr einfach, responsive d3-SVGs zu erstellen .

Zuerst erstellen wir eine Zeile und eine Spalte, in die unser Diagramm eingebaut wird.

index.html

```
<!DOCTYPE html>
<html>
<head>
<link rel="stylesheet" type="text/css"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
</head>
<body>
  <div class="container">
    <div class="row">
      <div class="col-xs-12 col-lg-6" id="chartArea">
      </div>
    </div>
  </div>
<script src="https://cdnjs.cloudflare.com/ajax/libs/d3/4.1.1/d3.js"></script>
<script src="chart.js"></script>
</body>
</html>
```

Dadurch wird eine Kolumne erstellt, die auf dem mobilen Gerät als Vollbild und auf einem großen Bildschirm zur Hälfte angezeigt wird.

chart.js

```
var width = document.getElementById('chartArea').clientWidth;
```

```
//this allows us to collect the width of the div where the SVG will go.
var height = width / 3.236;
//I like to use the golden rectangle ratio if they work for my charts.

var svg = d3.select('#chartArea').append('svg');
//We add our svg to the div area


//We will build a basic function to handle window resizing.
function resize() {
    width = document.getElementById('chartArea').clientWidth;
    height = width / 3.236;
    d3.select('#chartArea svg')
      .attr('width', width)
      .attr('height', height);
}

window.onresize = resize;
//Call our resize function if the window size is changed.
```

Dies ist ein extrem vereinfachtes Beispiel, deckt jedoch die grundlegenden Konzepte für das Einrichten von Diagrammen ab. Die Größenänderungsfunktion muss Ihre Hauptaktualisierungsfunktion aufrufen, die alle Pfade, Achsen und Formen neu zeichnet, als ob die zugrunde liegenden Daten aktualisiert worden wären. Die meisten D3-Benutzer, die sich mit responsiven Visualisierungen beschäftigen, wissen bereits, wie sie ihre Update-Events in Funktionen umwandeln können, die einfach aufgerufen werden können, wie im Intro-Thema und in diesem Thema beschrieben .

Ansätze zum Erstellen von responsiven d3.js-Diagrammen online lesen: https://riptutorial.com/de/d3-js/topic/4312/ansatze-zum-erstellen-von-responsiven-d3-js-diagrammen

Kapitel 4: Auswahlmöglichkeiten

Syntax

- d3. **Auswahl** (**Auswahl**)
- d3. **selectAll** (**Auswahl**)
- *Auswahl* . **Auswahl** (**Auswahl**)
- *Auswahl* . **selectAll** (**Auswahl**)
- *Auswahl* . **filter** (filter)
- *Auswahl* . **zusammenführen** (andere)

Bemerkungen

Verwandte Lesungen:

- Wie Auswahlen funktionieren - Mike Bostock
- d3-selection README

Examples

Grundauswahl und Modifikationen

Wenn Sie mit der Syntax von jQuery und Sizzle vertraut sind, sollte die Auswahl von d3 nicht viel anders sein. d3 ahmt die W3C Selectors API nach, um die Interaktion mit Elementen zu vereinfachen.

Um ein einfaches Beispiel zu erhalten, wählen Sie alle `<p>` und fügen Sie jeweils eine Änderung hinzu:

```
d3.selectAll('p')
  .attr('class','textClass')
  .style('color', 'white');
```

Kurz gesagt ist dies relativ das Gleiche wie in jQuery

```
$('p')
  .attr('class','textClass')
  .css('color, 'white')
```

Im Allgemeinen beginnen Sie mit einer einfachen Auswahl in Ihrem Container div, um ein SVG-Element hinzuzufügen, das einer Variablen zugewiesen wird (am häufigsten als svg bezeichnet).

```
var svg = d3.select('#divID').append('svg');
```

Von hier aus können wir `svg` dazu auffordern, eine Unterauswahl mehrerer Objekte `svg` (auch

wenn sie noch nicht existieren).

```
svg.selectAll('path')
```

Verschiedene Selektoren

Sie können Elemente mit verschiedenen Selektoren auswählen:

- nach tag: `"div"`
- nach Klasse: `".class"`
- von id: `"#id"`
- nach Attribut: `"[color=blue]"`
- Mehrfachselektoren (ODER): `"div1, div2, class1"`
- Mehrfachselektoren (UND): `"div1 div2 class1"`

Einfache datenbeschränkte Auswahl

```
var myData = [
    { name: "test1", value: "ok" },
    { name: "test2", value: "nok" }
]

// We have to select elements (here div.samples)
// and assign data. The second parameter of data() is really important,
// it will determine the "key" to identify part of data (datum) attached to an
// element.
var mySelection = d3.select(document.body).selectAll("div.samples") // <- a selection
                    .data(myData, function(d){ return d.name; }); // <- data binding

// The "update" state is when a datum of data array has already
// an existing element associated.
mySelection.attr("class", "samples update")

// A datum is in the "enter" state when it's not assigned
// to an existing element (based on the key param of data())
// i.e. new elements in data array with a new key (here "name")
mySelection.enter().append("div")
    .attr("class", "samples enter")
    .text(function(d){ return d.name; });

// The "exit" state is when the bounded datum of an element
// is not in the data array
// i.e. removal of a row (or modifying "name" attribute)
// if we change "test1" to "test3", "test1" bounded
//          element will figure in exit() selection
// "test3" bounded element will be created in the enter() selection
mySelection.exit().attr("class", "samples remove");
```

Die Rolle von Platzhaltern bei der Auswahl von Einträgen

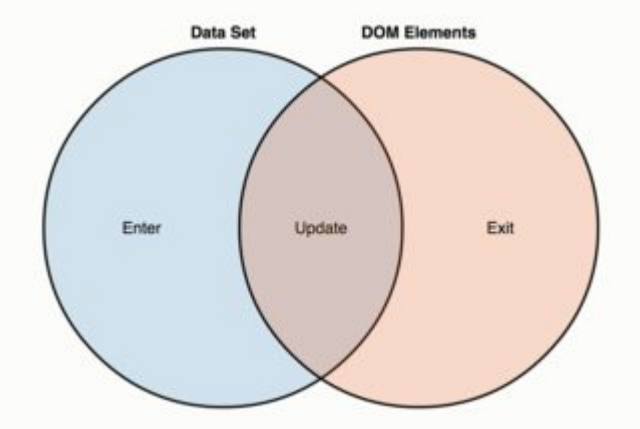
Was ist eine Enter-Auswahl?

Wenn in D3.js Daten an DOM-Elemente gebunden werden, sind drei Situationen möglich:

1. Die Anzahl der Elemente und die Anzahl der Datenpunkte sind gleich.
2. Es gibt mehr Elemente als Datenpunkte.
3. Es gibt mehr Datenpunkte als Elemente.

In der Situation , # 3, alle Datenpunkte ohne ein entsprechendes DOM - Element gehören zur Auswahl *gelangen.* Somit In D3.js *Geben* Auswahlen Auswahlen sind , dass nach Elementen mit dem Datenverbindungs enthält alle Daten , die nicht DOM - Element übereinstimmen. Wenn wir eine verwenden `append` Funktion in einer Auswahl *eingeben,* wird D3 neue Elemente schaffen für uns , dass die Datenbindung.

Dies ist ein Venn-Diagramm, das die möglichen Situationen in Bezug auf die Anzahl der Datenpunkte / Anzahl der DOM-Elemente erläutert:



Wie wir sehen können, ist die *Eingabeauswahl* der blaue Bereich links: Datenpunkte ohne entsprechende DOM-Elemente.

Die Struktur der Enter-Auswahl

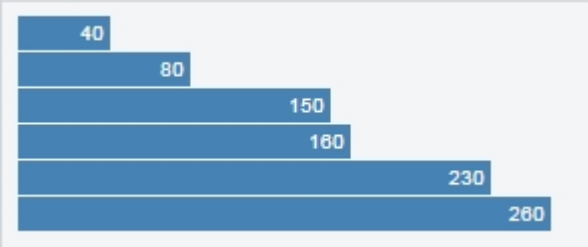
Eine *Eingabeauswahl* hat normalerweise die folgenden 4 Schritte:

1. `selectAll` : Elemente im DOM `selectAll` ;
2. `data` : Zählt und analysiert die Daten.
3. `enter` : Durch Vergleichen der Auswahl mit den Daten werden neue Elemente erstellt.
4. `append` : Die tatsächlichen Elemente im DOM `append` .

Dies ist ein sehr einfaches Beispiel (siehe die 4 Schritte in den `var divs`):

```
var data = [40, 80, 150, 160, 230, 260];

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

var divs = body.selectAll("div")
    .data(data)
    .enter()
    .append("div");

divs.style("width", function(d) { return d + "px"; })
    .attr("class", "divchart")
    .text(function(d) { return d; });
```

Und das ist das Ergebnis (jsfiddle hier):



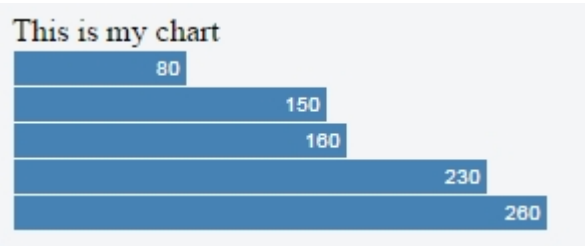
Beachten Sie, dass in diesem Fall `selectAll("div")` als erste Zeile in unserer Auswahlvariable "enter" verwendet wurde. Wir haben einen Datensatz mit 6 Werten und D3 hat 6 Divs für uns erstellt.

Die Rolle der Platzhalter

Angenommen, wir haben bereits ein div in unserem Dokument, etwa `<div>This is my chart</div>` oben. In diesem Fall, wenn wir schreiben:

```
body.selectAll("div")
```

Wir wählen das vorhandene div aus. Unsere Eingabe-Auswahl hat also nur 5 Datum ohne passende Elemente. Zum Beispiel in diesem jsfiddle , wo es bereits ein div im HTML- Code gibt ("This is my chart"), wird dies das Ergebnis sein:



Wir sehen den Wert "40" nicht mehr: Unser erster "Balken" ist verschwunden, und der Grund dafür ist, dass unsere Auswahl "Enter" jetzt nur noch 5 Elemente hat.

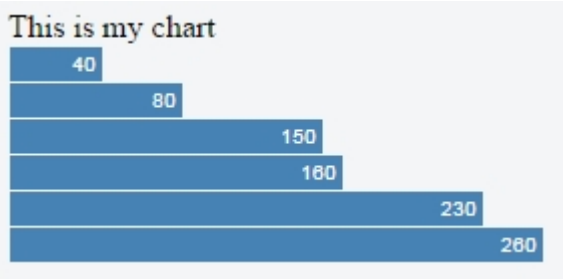
Was wir hier verstehen müssen ist, dass in der ersten Zeile unserer Enter-Auswahlvariable `selectAll("div")` diese Divs nur *Platzhalter sind* . Wir müssen nicht alle `divs` auswählen, wenn wir `divs` anhängen, oder den gesamten `circle` wenn wir einen `circle` anhängen. Wir können verschiedene Dinge auswählen. Und wenn wir nicht vorhaben, ein "Update" oder ein "Exit" auszuwählen, können wir *alles* auswählen:

```
var divs = body.selectAll(".foo")//this class doesn't exist, and never will!
    .data(data)
    .enter()
    .append("div");
```

Auf diese Weise wählen wir alle ".foo" aus. Hier ist "foo" eine Klasse, die nicht nur nicht existiert, sondern auch nirgendwo anders im Code erstellt wurde! Das ist aber egal, dies ist nur ein Platzhalter. Die Logik ist folgende:

Wenn Sie in Ihrer Auswahl "Eingabe" etwas auswählen, das nicht existiert, enthält Ihre Auswahl "Eingabe" *immer* alle Ihre Daten.

`.foo` Sie nun `.foo` auswählen, enthält unsere Auswahl "enter" 6 Elemente, selbst wenn im Dokument bereits ein div vorhanden ist:



Und hier ist das entsprechende jsfiddle .

`null` auswählen

Die beste Möglichkeit, um sicherzustellen, dass Sie nichts auswählen, ist die Auswahl von `null` . Nicht nur das, diese Alternative ist viel schneller als jede andere.

Führen Sie für eine Auswahl die Eingabe einfach aus:

```
selection.selectAll(null)
    .data(data)
    .enter()
    .append(element);
```

Hier ist eine Demo-Geige: https://jsfiddle.net/gerardofurtado/th6s160p/

Fazit

Achten Sie bei der Auswahl von "Eingaben" darauf, dass Sie nicht auswählen, was bereits vorhanden ist. Sie können alles in Ihrer `selectAll` , auch Dinge, die nicht existieren und niemals existieren werden (wenn Sie nicht `selectAll` , ein "Update" oder "Exit" auszuwählen).

Der Code in den Beispielen basiert auf diesem Code von Mike Bostock: https://bl.ocks.org/mbostock/7322386

Verwenden Sie "this" mit einer Pfeilfunktion

Die meisten Funktionen in D3.js akzeptieren eine anonyme Funktion als Argument. Die häufigsten Beispiele sind `.attr` , `.style` , `.text` , `.on` und `.data` , aber die Liste ist viel größer.

In solchen Fällen wird die anonyme Funktion für jedes ausgewählte Element ausgewertet und der Reihe nach übergeben:

1. Das aktuelle Datum (`d`)
2. Der aktuelle Index (`i`)
3. Die aktuelle Gruppe (`nodes`)
4. `this`

als aktuelles DOM-Element.

Das Datum, der Index und die aktuelle Gruppe werden als Argumente übergeben, das berühmte erste, zweite und dritte Argument in D3.js (dessen Parameter traditionell `d` , `i` und `p` in D3 v3.x heißen). Um `this` , muss jedoch kein Argument verwendet werden:

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

Der obige Code wählt `this` , wenn die Maus über das Element ist. Überprüfen Sie, ob es in dieser Geige funktioniert: https://jsfiddle.net/y5fwgopx/

Die Pfeilfunktion

Als neue ES6-Syntax hat eine Pfeilfunktion im Vergleich zum Funktionsausdruck eine kürzere Syntax. Für einen D3-Programmierer, der `this` ständig verwendet, gibt es jedoch eine Gefahr: Eine Pfeilfunktion erzeugt `this` Kontext nicht. Das bedeutet , dass in einer Pfeil Funktion, `this` seine ursprüngliche Bedeutung aus dem einschließenden Kontext hat.

Dies kann unter verschiedenen Umständen nützlich sein, aber es ist ein Problem für einen Codierer, der es gewohnt ist, `this` in D3 zu verwenden. Wenn Sie beispielsweise dasselbe Beispiel in der oben genannten Geige verwenden, funktioniert dies nicht:

```
.on("mouseover", ()=>{
    d3.select(this);
});
```

Wenn Sie daran zweifeln, ist hier die Geige: https://jsfiddle.net/tfxLsv9u/

Nun, das ist kein großes Problem: Man kann bei Bedarf einfach einen regulären, altmodischen Funktionsausdruck verwenden. Was aber, wenn Sie Ihren gesamten Code mithilfe von Pfeilfunktionen schreiben möchten? Ist es möglich , einen Code mit Pfeil Funktionen zu haben **und** nach wie vor richtig nutzt `this` in D3?

Das zweite und dritte Argument kombiniert

Die Antwort lautet **ja** , weil `nodes[i] this` ist. Der Hinweis ist in der gesamten D3-API vorhanden, wenn er Folgendes beschreibt:

> ... mit `this` als aktuellem DOM-Element (`nodes[i]`)

Die Erklärung ist einfach: Da `nodes` die aktuelle Gruppe von Elementen im DOM und `i` der Index jedes Elements ist, beziehen sich `nodes[i]` auf das aktuelle DOM-Element selbst. Das ist `this` .

Daher kann man verwenden:

```
.on("mouseover", (d, i, nodes) => {
    d3.select(nodes[i]);
});
```

Und hier ist die entsprechende Geige: https://jsfiddle.net/2p2ux38s/

Auswahlmöglichkeiten online lesen: https://riptutorial.com/de/d3-js/topic/2135/auswahlmoglichkeiten

Kapitel 5: D3-Projektionen

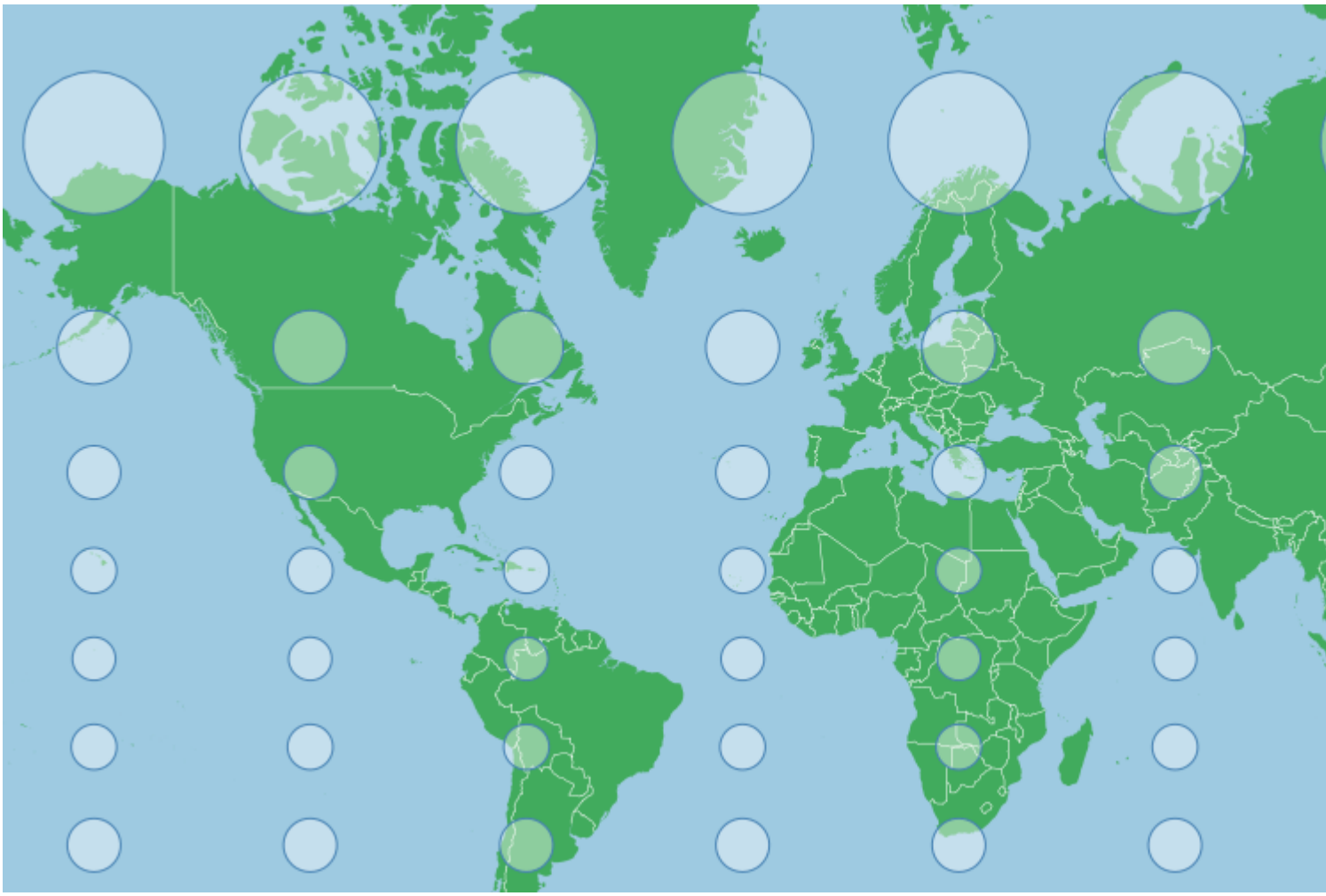
Examples

Mercator-Projektionen

Eine Mercator-Projektion ist eine der am häufigsten erkennbaren Projektionen, die in Karten verwendet werden. Wie alle Kartenprojektionen hat es eine Verzerrung, und für einen Mercator ist die Projektion in den Polarregionen am deutlichsten. Es ist eine zylindrische Projektion, Meridiane verlaufen vertikal und Breiten verlaufen horizontal.

Die Skalierung hängt von der Größe Ihrer svg-Datei ab. In diesem Beispiel werden alle Skalen mit einer Breite von 960 Pixel und einer Breite von 450 Pixel verwendet.

Die folgende Karte zeigt eine Tissotsche Indikatrix für eine Mercator-Projektion. Jeder Kreis ist in Wirklichkeit gleich groß, aber die Projektion zeigt offensichtlich einige größer als andere:



Diese Verzerrung ist darauf zurückzuführen, dass die Projektion versucht, eine eindimensionale Dehnung der Karte zu vermeiden. Wenn Meridiane an den Nord- und Südpolen zu verschmelzen beginnen, nähert sich der Abstand zwischen ihnen null, aber die Oberfläche der Projektion ist rechteckig (nicht die Karte, obwohl sie auch rechteckig ist) und erlaubt keine Änderung der Entfernung zwischen Meridianen in der Projektion. Dies würde Merkmale entlang der x-Achse in

der Nähe der Pole strecken und ihre Form verzerren. Um dem entgegenzuwirken, streckt ein Mercator sowohl die y-Achse als auch eine Annäherung an die Pole, wodurch die Indikatoren kreisförmig werden.

Die Projektion für die Karte oben ist im Wesentlichen die Standardprojektion von Mercator, die etwas nach oben verschoben wurde:

```
var projection = d3.geoMercator()
    .scale(155)
    .center([0,40]) // Pan north 40 degrees
    .translate([width/2,height/2]);
```

Um die Projektion auf einen bestimmten Punkt mit einem bekannten Breitengrad und einem bekannten Längengrad zu zentrieren, können Sie diesen Punkt leicht verschieben, indem Sie den Mittelpunkt angeben:

```
var projection = d3.geoMercator()
    .center([longitude,latitude])
```

Dadurch wird auf der projizierten Fläche (die der obigen Karte ähnelt) zu dieser Funktion geschwenkt (aber nicht gezoomt).

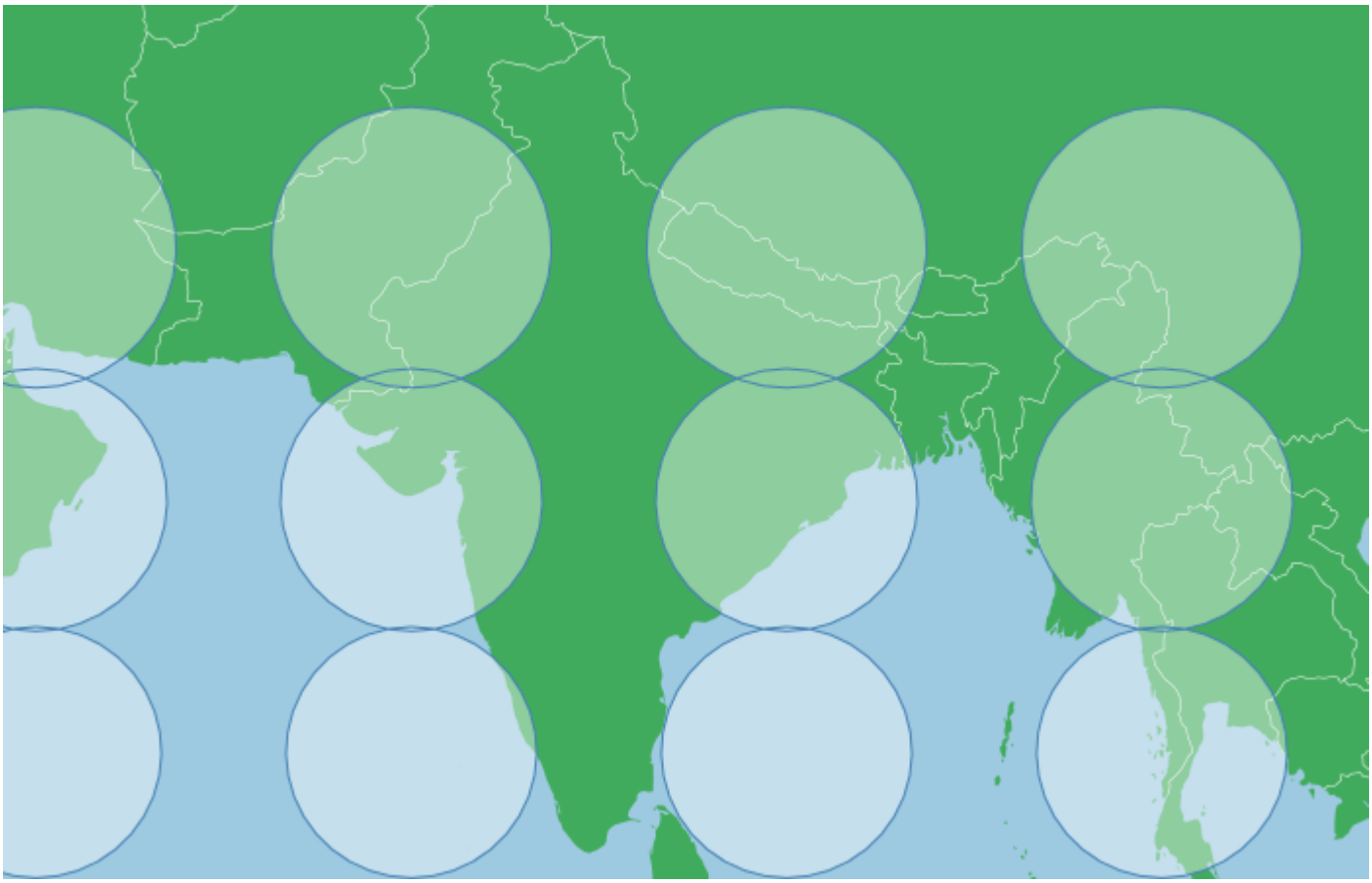
Maßstäbe müssen auf das interessierende Gebiet zugeschnitten sein, größere Zahlen entsprechen größeren Merkmalen (größeres Maß an Vergrößerung), kleinere Zahlen das Gegenteil. Das Herauszoomen kann eine gute Möglichkeit sein, um sich zu orientieren, wo sich Ihre Features relativ zu dem Punkt befinden, auf den Sie zentriert sind - falls Sie Probleme haben, sie zu finden.

Aufgrund der Natur einer Mercator-Projektion eignen sich Bereiche in Äquatornähe oder in niedrigen Breiten mit dieser Art von Projektion am besten, während polare Bereiche stark verzerrt sein können. Die Verzerrung verläuft entlang jeder horizontalen Linie, sodass Bereiche, die breit, aber nicht groß sind, ebenfalls gut sein können, während Bereiche, die einen großen Unterschied zwischen den nördlichen und südlichen Extremen aufweisen, eine stärkere visuelle Verzerrung aufweisen.

Für Indien könnten wir beispielsweise Folgendes verwenden:

```
var projection = d3.geoMercator()
    .scale(800)
    .center([77,21])
    .translate([width/2,height/2]);
```

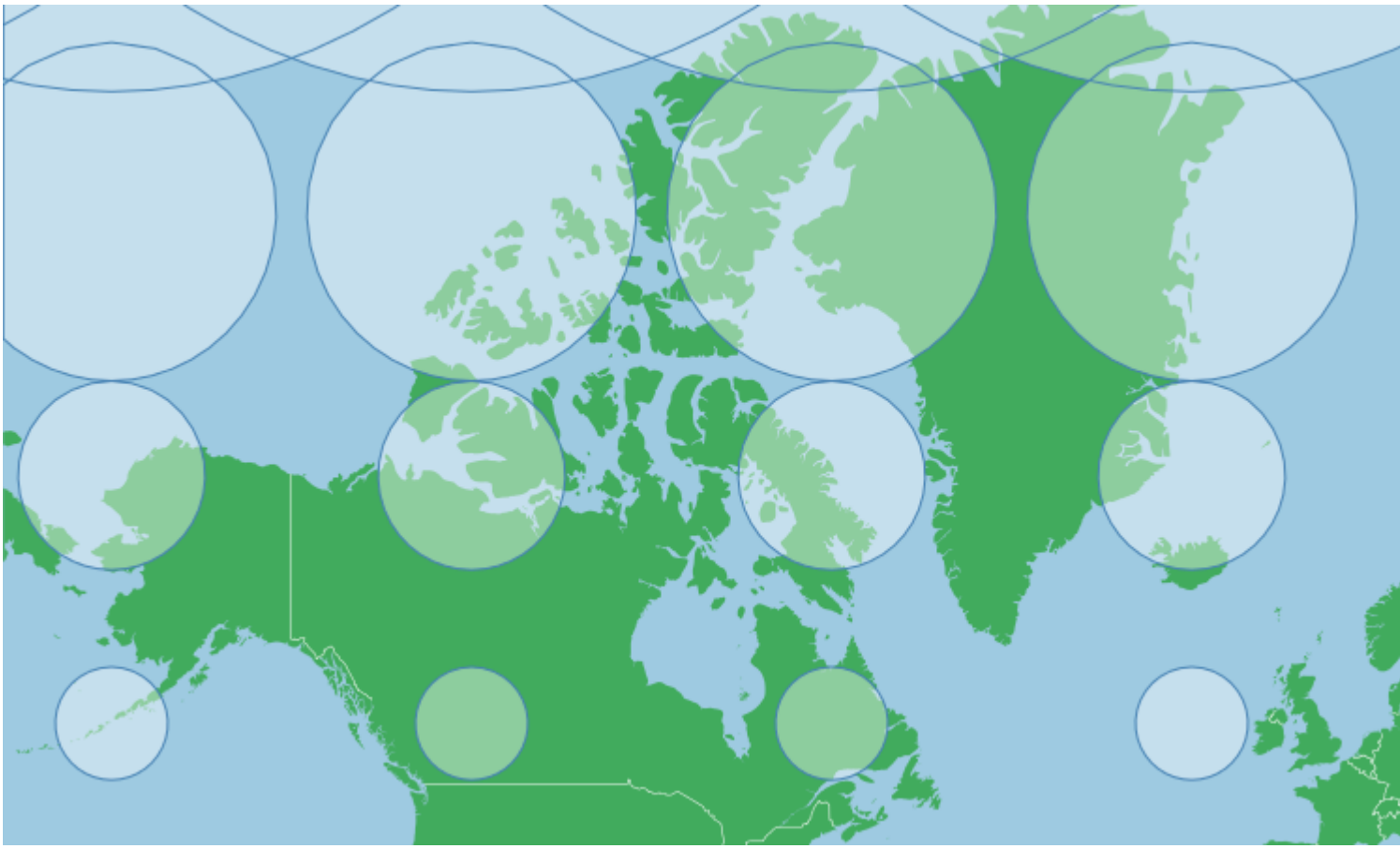
Das gibt uns (wieder mit einer Tissot-Indikatrix, um Verzerrung zu zeigen):



Dies hat eine geringe Verzerrung, aber die Kreise haben im Wesentlichen die gleiche Größe (Sie können eine größere Überlappung zwischen den beiden oberen Reihen als den beiden unteren Reihen sehen, sodass Verzerrungen sichtbar sind). Insgesamt zeigt die Karte eine für Indien bekannte Form.

Die Verzerrung im Bereich ist nicht linear, sie ist in Richtung der Pole stark übertrieben, so dass Kanada mit ziemlich weit auseinander liegenden nördlichen und südlichen Extremen und einer Position nahe an den Polen bedeutet, dass Verzerrung unhaltbar sein kann:

```
var projection = d3.geoMercator()
    .scale(225)
    .center([-95,69.75])
    .translate([width/2,height/2]);
```

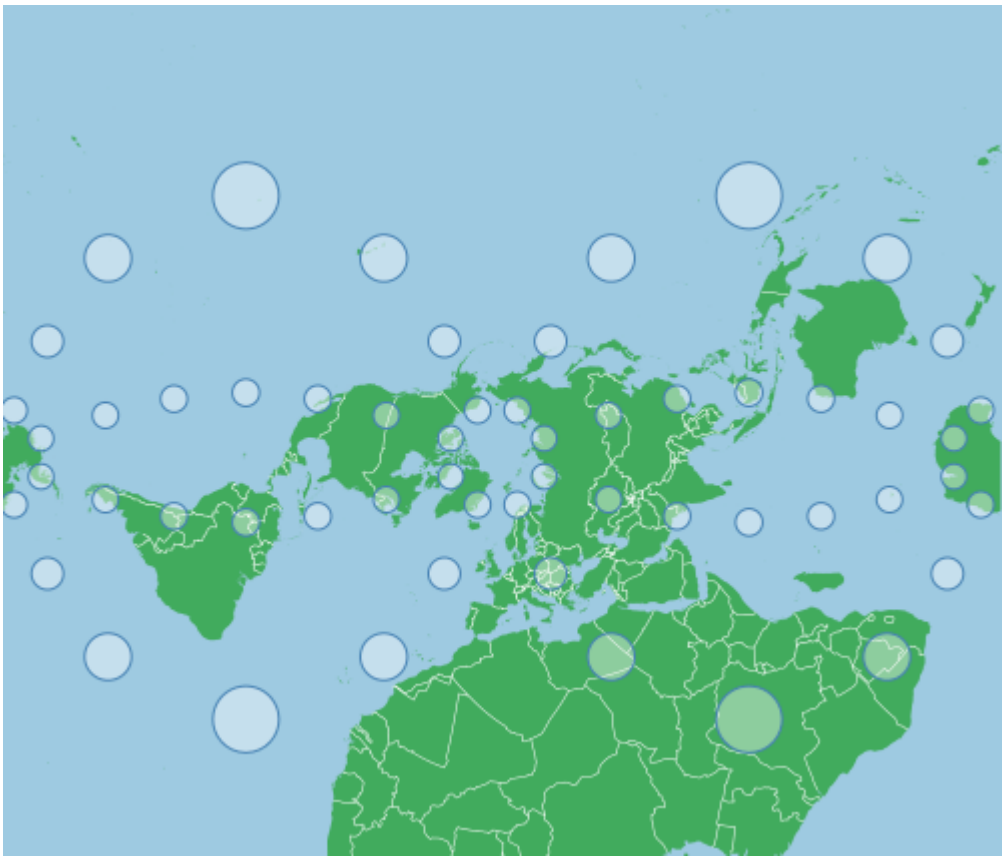


Diese Projektion lässt Grönland so groß aussehen wie Kanada, obwohl Kanada in Wirklichkeit fast fünfmal so groß ist wie Grönland. Dies liegt einfach daran, dass Grönland weiter nördlich als der Großteil Kanadas liegt (Sorry Ontario, ich scheine einige Ihrer südlichen Extremitäten abgeschnitten zu haben).

Da die y-Achse in einem Mercator in der Nähe der Pole stark gestreckt ist, wird für diese Projektion ein Punkt nördlich des geographischen Zentrums von Kanada verwendet. Wenn Sie mit großen Breiten arbeiten, müssen Sie möglicherweise Ihren Zentrierpunkt anpassen, um diese Dehnung zu berücksichtigen.

Wenn Sie eine Mercator-Projektion für polare Bereiche benötigen, können Sie die Verzerrung minimieren und trotzdem eine Mercator-Projektion verwenden. Sie können dies erreichen, indem Sie den Globus drehen. Wenn Sie die x-Achse am Standard-Mercator drehen, scheinen Sie nach links oder rechts zu schwenken (Sie drehen einfach den Globus in dem Zylinder, auf den Sie projizieren). Wenn Sie jedoch die y-Achse des Standard-Mercators ändern, können Sie dies drehen Sie die Erde seitwärts oder in einen anderen Winkel. Hier ist ein Mercator mit einer Rotation von -90 Grad:

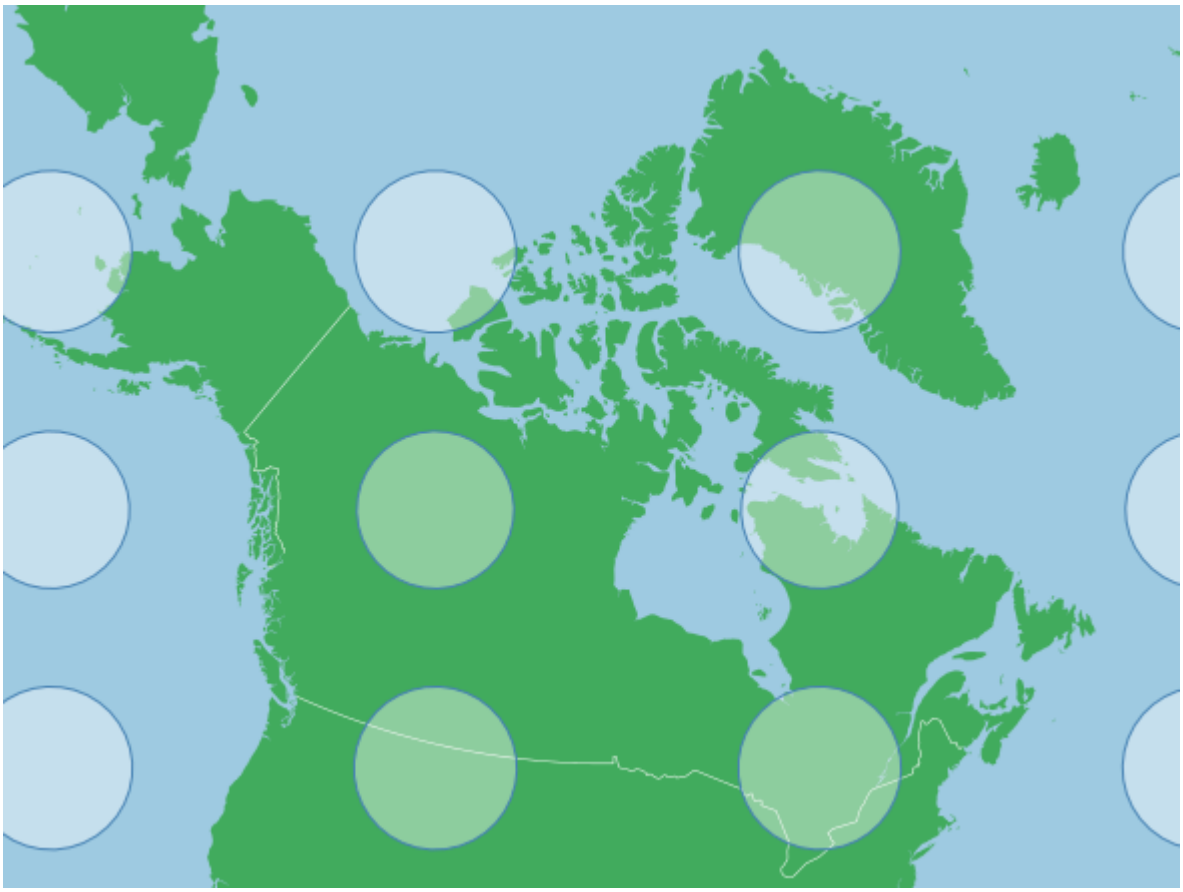
```
var projection = d3.geoMercator()
.scale(155)
.rotate([0,-90]);
.translate([width/2,height/2]);
```



Die Indikatrixpunkte befinden sich an den gleichen Stellen wie die erste Karte oben. Die Verzerrung nimmt immer mehr zu, je höher der obere oder untere Rand der Karte ist. Auf diese Weise würde eine Mercator-Standardkarte erscheinen, wenn sich die Erde bei [0,0] um einen Nordpol und bei [180,0] um einen Südpol drehen würde. Durch die Rotation wurde der Zylinder, auf den wir projizieren, um 90 Grad gegenüber gedreht die Pole. Beachten Sie, dass die Pole keine unhaltbaren Verzerrungen mehr aufweisen. Dies ist eine Alternative für das Projizieren von Bereichen in Polnähe ohne zu starke Verzerrung der Fläche.

Wenn wir wieder Kanada als Beispiel verwenden, können wir die Karte auf eine Mittelpunktskoordinate drehen, wodurch Verzerrungen im Bereich minimiert werden. Dazu können wir wieder zu einem Zentrierpunkt drehen, dies erfordert jedoch einen zusätzlichen Schritt. Mit der Zentrierung schwenken wir zu einem Merkmal, mit der Drehung bewegen wir die Erde unter uns, also brauchen wir das Negativ unserer Zentrierkoordinate:

```
var projection = d3.geoMercator()
    .scale(500)
    .rotate([96,-64.15])
    .translate([width/2,height/2]);
```



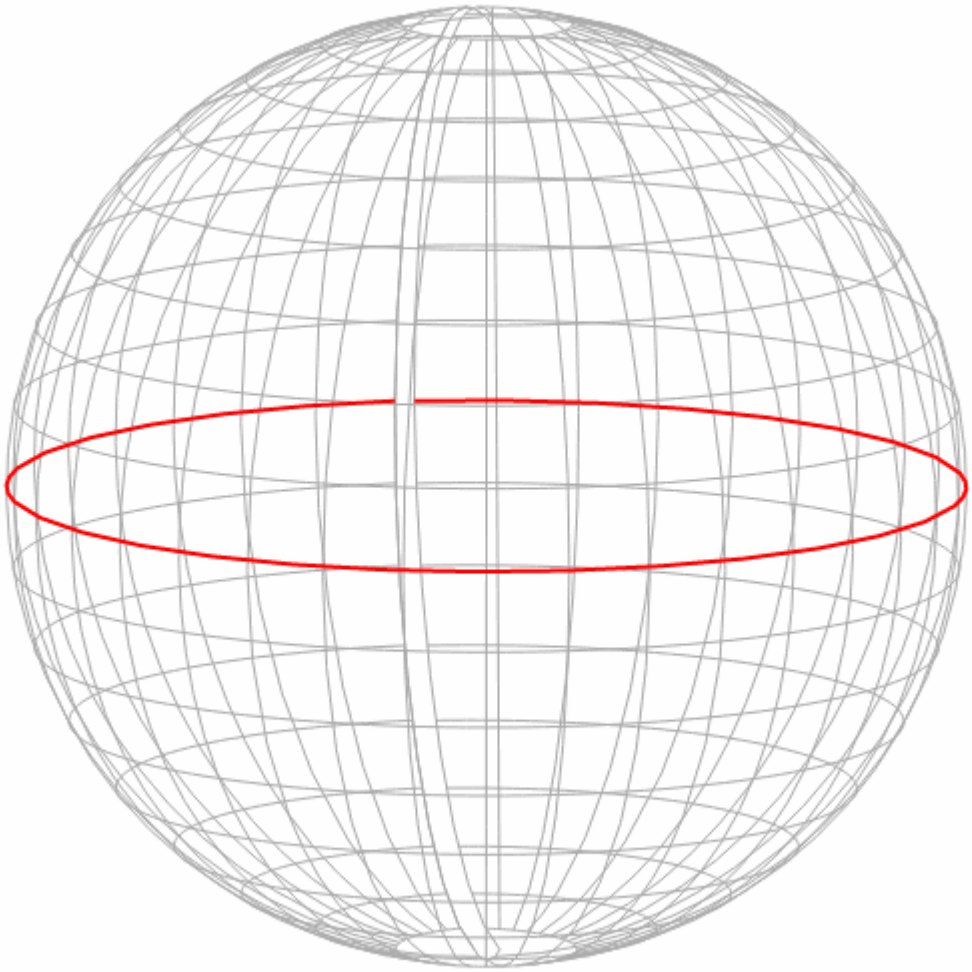
Beachten Sie, dass die Tissot-Indikatrix jetzt eine geringe Verzerrung im Bereich aufweist. Der Skalierungsfaktor ist auch viel größer als zuvor, da sich dieses Kanada jetzt am Ursprung der Projektion befindet und entlang der Kartenmitte kleiner sind als oben oder unten (siehe erste Indikatrix oben). Wir müssen nicht zentrieren, da der Mittelpunkt oder Ursprung dieser Projektion bei `[-96,64.15]` liegt. Die Zentrierung würde uns von diesem Punkt entfernen.

Albers-Projektionen

Eine Albers-Projektion oder besser eine konische Albers-Fläche mit gleichem Flächeninhalt ist eine übliche konische Projektion und eine offizielle Projektion einer Reihe von Ländern und Organisationen wie dem US-amerikanischen Volkszählungsbüro und der Provinz British Columbia in Kanada. Die Fläche bleibt auf Kosten anderer Aspekte der Karte wie Form, Winkel und Entfernung erhalten.

Allgemeine Eigenschaften

Die allgemeine Transformation wird in folgendem GIF erfasst:



(Basierend auf Mike Bostocks Block)

Die Albers-Projektion minimiert Verzerrungen um zwei Standardparallelen. Diese Parallelen stellen dar, wo die konische Projektion die Erdoberfläche schneidet.

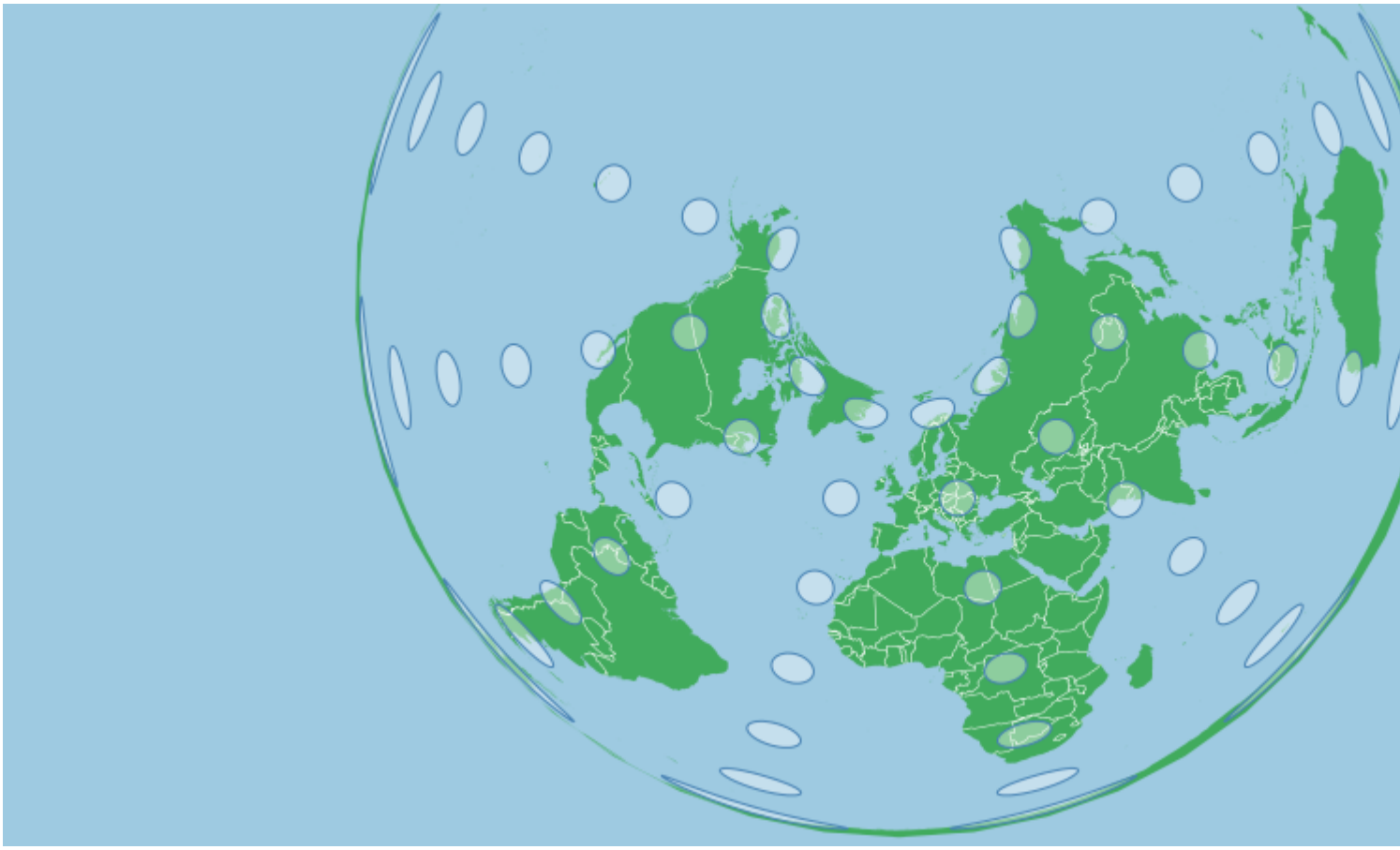
In diesem Beispiel werden alle Maßstäbe mit einer Breite von 960 Pixeln und einer Breite von 450 Pixeln verwendet, wobei sich der Maßstab mit diesen Abmessungen ändert

Die folgende Karte zeigt eine Tissotsche Indikatrix für eine Albers-Projektion mit Standardparallelen von 10 und 20 Grad Nord. Jeder Kreis hat in Wirklichkeit die gleiche Größe und Form, aber die Kartenprojektion verzerrt diese in der Form (nicht in der Fläche). Beachten Sie, dass bei ungefähr 10 bis 20 Grad Nord die Indikatoren runder sind als anderswo:

Dies wurde mit der folgenden Projektion erstellt:

```
var projection = d3.geoAlbers()
    .scale(120)
    .center([0,0])
    .rotate([0,0])
    .parallels([10,20])
    .translate([width/2,height/2]);
```

Wenn wir Parallelen verwenden, die in höheren Lagen zunehmen, nimmt der Bogengrad in der Projektion zu. Die folgenden Bilder verwenden die Parallelen von 50 und 60 Grad Nord:



```
var projection = d3.geoAlbers()
    .scale(120)
    .center([0,70]) // shifted up so that the projection is more visible
    .rotate([0,0])
    .parallels([40,50])
    .translate([width/2,height/2]);
```

Wenn wir negative (südliche) Parallelen hatten, sollte die Karte nach unten konkav sein anstatt nach oben. Wenn eine Parallele nördlich und eine südlich ist, ist die Karte in Richtung der höheren / extremeren Parallele konkav. Wenn sie vom Äquator gleich weit entfernt sind, ist die Karte in keiner Richtung konkav.

Auswahl von Parallelen

Da Parallelen die Bereiche mit der geringsten Verzerrung markieren, sollten sie auf der Grundlage Ihres Interessengebiets ausgewählt werden. Wenn sich der Interessenbereich von 10 Grad nach Norden bis 20 Grad nach Norden erstreckt, werden durch die Auswahl von Parallelen von 13 und 17 Verzerrungen auf der gesamten Karte minimiert (da die Verzerrung auf beiden Seiten dieser Parallelen minimiert wird).

Parallelen sollten nicht die extremen nördlichen und südlichen Grenzen Ihres Interessengebiets sein. *Parallelen können beide den gleichen Wert haben, wenn die Projektion nur einmal die Erdoberfläche schneiden soll.*

Projektionsreferenzen und Definitionen enthalten parallele Daten, mit denen Sie standardisierte

Projektionen erstellen können.

Zentrieren und Drehen

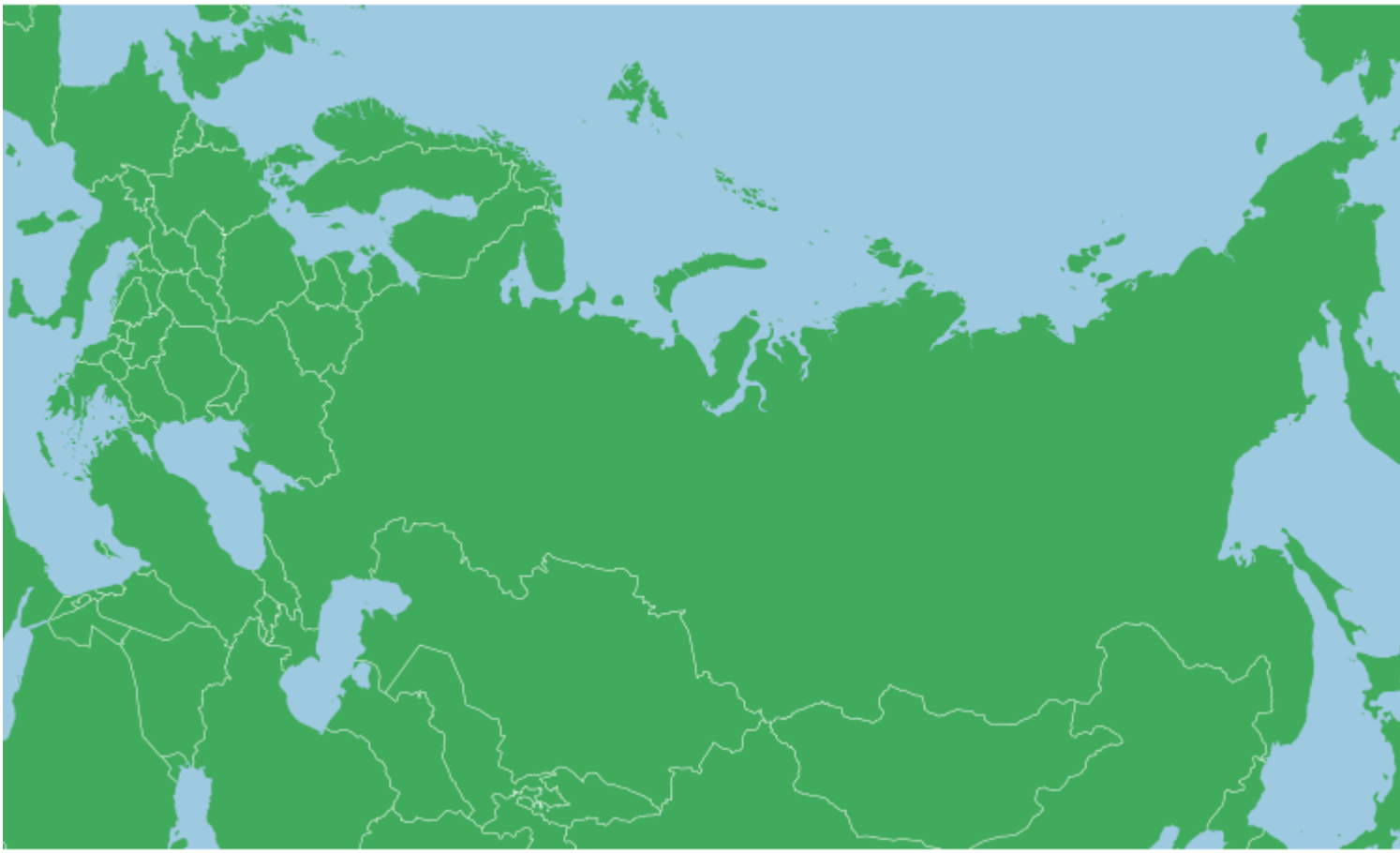
Nach der Auswahl von Parallelen muss die Karte so positioniert werden, dass der interessierende Bereich korrekt ausgerichtet ist. Wenn Sie nur `projection.center([x,y])` , wird die Karte einfach zum ausgewählten Punkt verschoben, und es erfolgt keine weitere Transformation. Wenn das Zielgebiet Russland ist, ist das Verschieben möglicherweise nicht ideal:

```
var projection = d3.geoAlbers()
    .scale(120)
    .center([0,50]) // Shifted up so the projection is more visible
    .rotate([0,0])
    .parallels([50,60])
    .translate([width/2,height/2]);
```

Der zentrale Meridian einer Albers-Projektion ist vertikal und wir müssen die Erde unter der Projektion drehen, um den zentralen Meridian zu ändern. Rotation für eine Alber-Projektion ist die Methode zum Zentrieren einer Projektion auf der x-Achse (oder nach Längengrad). Und da sich die Erde unter der Projektion dreht, verwenden wir das Negativ der Länge, die wir zentrieren wollen. Für Russland könnte dies etwa 100 Grad Ost sein, also drehen wir den Globus um 100 Grad in die andere Richtung.

```
var projection = d3.geoAlbers()
    .scale(120)
    .center([0,60])
    .rotate([-100,0])
    .parallels([50,60])
```

Jetzt können wir nach oben und unten schwenken, und die Merkmale entlang und in der Nähe des Mittelmeridians werden gerade sein. *Wenn Sie* `.center()` *auf der x-Achse liegen, ist Ihre Zentrierung relativ zum mittleren Meridian, der durch die Rotation festgelegt wird* . Für Russland möchten wir vielleicht ein gutes Stück nach Norden schwenken und ein wenig vergrößern:



```
var projection = d3.geoAlbers()
    .scale(500)
    .center([0,65])
    .rotate([-100,0])
    .parallels([50,60])
```

Bei einem Feature wie Russland bedeutet der Bogen der Karte, dass sich die weiten Ränder des Landes um den Pol herum erstrecken, was bedeutet, dass der Zentrierpunkt möglicherweise nicht der Schwerpunkt Ihres Features ist, da Sie möglicherweise mehr zum Panorama schwenken müssen Norden oder Süden als üblich.

Mit der Tissots Indicatrix können wir eine Abflachung in der Nähe der Stange selbst feststellen, aber diese Form ist im gesamten Interessengebiet ziemlich zutreffend.

Standardparameter

Im Gegensatz zu den meisten anderen Projektionen verfügt die Projektion d3.geoAlbers über Standardparameter, die nicht .rotate ([0,0]) und .center ([0,0]) sind. Die Standardprojektion wird für die Vereinigten Staaten zentriert und gedreht. Dies gilt auch für `.parallels()` . Wenn einer dieser Werte nicht gesetzt ist, werden die Werte standardmäßig nicht Null.

Zusammenfassung

Eine Albers-Projektion wird im Allgemeinen mit den folgenden Parametern festgelegt:

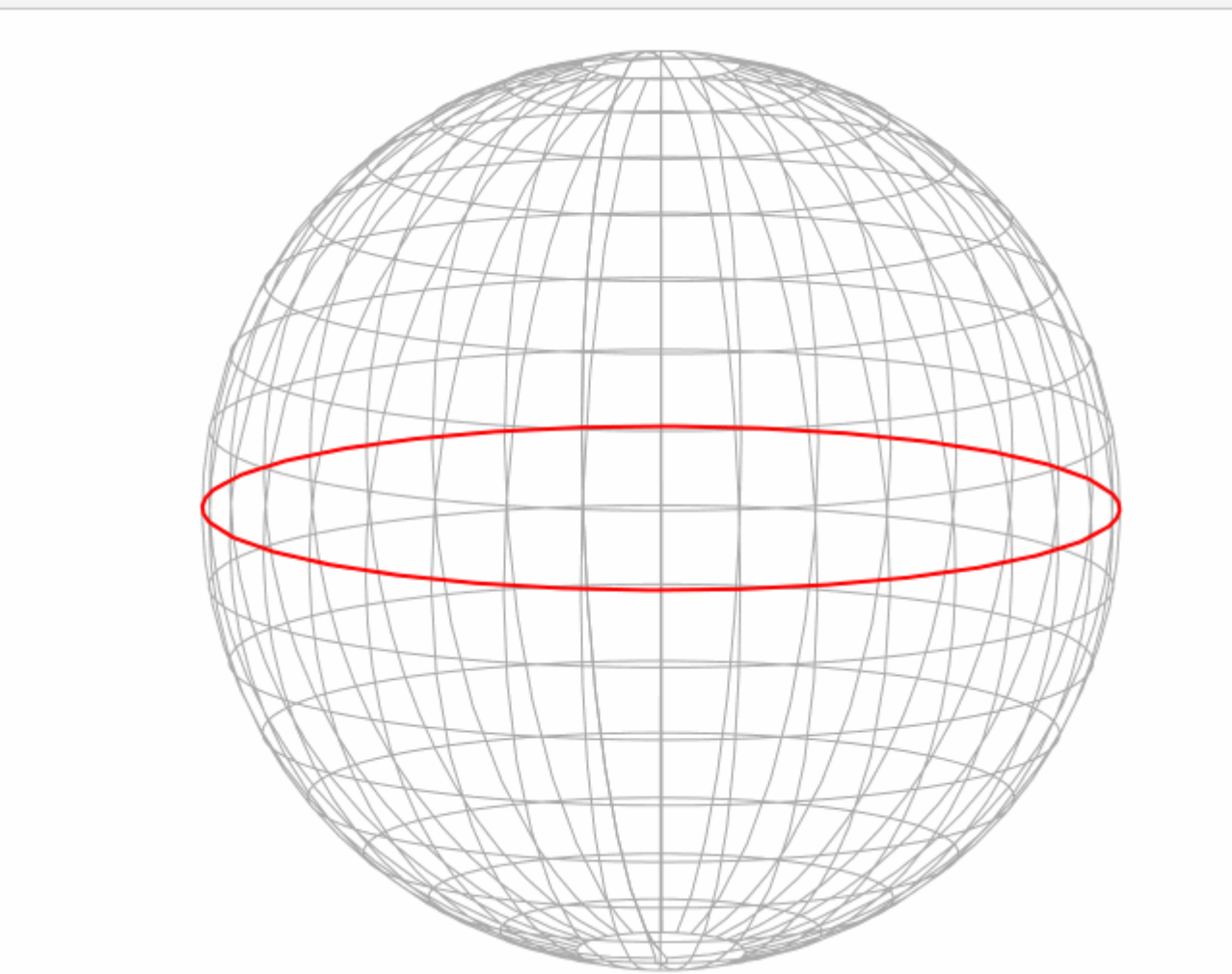
```
var projection = d3.geoAlbers()
    .rotate([-x,0])
    .center([0,y])
    .parallels([a,b]);
```

Wo a und b die beiden Parallelen sind.

Azimutale äquidistante Projektionen

Allgemeine Eigenschaften:

Eine azimutale äquidistante Projektion wird am besten bei Verwendung in Polargebieten erkannt. Es wird im Emblem der UNO verwendet . Vom Mittelpunkt aus bleiben Winkel und Abstand erhalten. Um dies zu erreichen, verzerrt die Projektion die Form und den Bereich, insbesondere wenn man sich weiter vom Zentrum entfernt. Ebenso sind Abstand und Winkel an anderen Stellen als der Mitte nicht wahr. Die Projektion fällt in die azimutale Kategorie (anstatt zylindrisch (Mercator) oder konisch (Albers). Diese Projektion projiziert die Erde als Scheibe:



(Basierend auf Mike Bostocks Block . *Ignoriere das dreieckige Artefakt oben auf dem Bild, sobald es ausgeklappt ist, zentriert am Nordpol.*)

Die Skalierung hängt von der Größe Ihrer SVG-Datei ab. In diesem Beispiel sind alle verwendeten Skalen 960 Pixel breit und 450 Pixel hoch (und der Bildschirm wird bei Bedarf für ein Quadrat abgeschnitten), sofern nichts anderes angegeben ist.

Die folgende Karte zeigt eine Tissotsche Indikatrix für eine azimutale äquidistante Projektion:

Dies wurde mit der folgenden Projektion erstellt:

```
var projection = d3.geoAzimuthalEquidistant()
   .scale(70)
   .center([0,0])
   .rotate([0,0])
   .translate([width/2,height/2]);
```

Zentrieren und Drehen:

Durch das Zentrieren wird eine Karte verschoben, die Gesamtzusammensetzung jedoch nicht geändert. Wenn ein azimutaler Äquidistant auf dem Nordpol zentriert wird, während andere Parameter unverändert bleiben oder auf Null stehen, wird der Nordpol in die Mitte des Bildschirms verschoben. Andernfalls wird die Karte oben nicht geändert.

Um einen Bereich richtig zu zentrieren, müssen Sie ihn drehen. Wie bei jeder Drehung in d3 ist es am besten, die Erde unter der Projektion zu bewegen. Wenn Sie also die Erde um -90 Grad unter der Karte auf der y-Achse drehen, wird der Nordpol tatsächlich in der Mitte platziert:

```
var projection = d3.geoAzimuthalEquidistant()
   .scale(70)
   .center([0,0])
   .rotate([0,-90])
   .translate([width/2,height/2]);
```

Ebenso verhält sich die Drehung auf der x-Achse ähnlich. Wenn Sie zum Beispiel die Karte so drehen möchten, dass die kanadische Arktis aufrecht steht und sich auf den Nordpol zentriert, können Sie eine Projektion wie die folgende verwenden:

```
var projection = d3.geoAzimuthalEquidistant()
    .scale(400)
    .center([0,0])
    .rotate([100,-90])
    .translate([width/2,height/2]);
```

Diese Karte verwendete ein 600x600-SVG

Insgesamt macht diese Einfachheit einen Azimut-Äquidistanten zu einer einfacheren Projektion, verwenden Sie einfach die Rotation. Eine typische Projektion sieht folgendermaßen aus:

```
var projection = d3.geoProjection()
   .center([0,0])
   .rotate([-x,-y])
   .scale(k)
   .translate([width/2,height/2]);
```

D3-Projektionen online lesen: https://riptutorial.com/de/d3-js/topic/9001/d3-projektionen

Kapitel 6: Die wichtigsten SVG-Konzepte, die in der D3.js-Visualisierung verwendet werden

Examples

Koordinatensystem

In einem normalen mathematischen Koordinatensystem befindet sich der Punkt x = 0, y = 0 in der linken unteren Ecke des Diagramms. Im SVG-Koordinatensystem befindet sich dieser (0,0) Punkt jedoch in der oberen linken Ecke der 'Zeichenfläche'. Er ähnelt CSS, wenn Sie die Position als absolut / fix angeben und die Position mithilfe von oben und links steuern genauer Punkt des Elements.

Es ist wichtig zu wissen, dass sich die Formen mit zunehmendem SVG-Wert nach unten bewegen.

Nehmen wir an, wir werden ein Streudiagramm erstellen, wobei jeder Punkt dem Achsenwert und dem y-Wert entspricht. Um den Wert zu skalieren, müssen wir die Domäne und den Bereich folgendermaßen einstellen:

```
d3.svg.scale()
  .range([0, height])
  .domain([0,max])
```

Wenn Sie jedoch nur die Einstellungen beibehalten, basieren die Punkte auf der oberen horizontalen Kante und nicht auf der unteren horizontalen Linie wie erwartet.

Das Schöne an d3 ist, dass Sie dies leicht durch eine einfache Anpassung der Domain-Einstellungen ändern können:

```
d3.scale.linear()
  .range([height, 0])
  .domain([0, max])
```

Mit obigem Code entspricht der Nullpunkt der Domäne der Höhe des SVG, der in den Augen des Betrachters die unterste Linie des Diagramms ist, während der Maximalwert der Quelldaten dem Nullpunkt des SVG entspricht Koordinatensystem, das den maximalen Wert für Zuschauer.

Das Element

`<rect>` für ein Rechteck. Abgesehen von den ästhetischen Eigenschaften wie Strich und Füllung muss das Rechteck nach Ort und Größe definiert werden.

Der Ort wird durch die Attribute x und y bestimmt. Die Position ist relativ zum übergeordneten Rechteck des Rechtecks. Wenn Sie das x- oder y-Attribut nicht angeben, ist der Standardwert 0

relativ zum übergeordneten Element.

Nachdem Sie den Ort bzw. den "Startpunkt" des Rect festgelegt haben, müssen Sie als Nächstes die Größe angeben. Dies ist wichtig, wenn Sie etw auf der Leinwand zeichnen möchten, d. H., Wenn Sie dies nicht tun Geben Sie die Größenattribute an, oder der Wert ist auf 0 gesetzt. Auf der Leinwand wird nichts angezeigt.

Fall: Balkendiagramm

Fahren Sie mit dem ersten Szenario fort, den Y-Achsen. Versuchen Sie diesmal jedoch, ein Balkendiagramm zu zeichnen.

Wenn man davon ausgeht, dass die y-Skaleneinstellung gleich ist, ist auch die y-Achse richtig gesetzt. Der einzige Unterschied zwischen dem Streudiagramm und diesem Balkendiagramm besteht darin, dass wir die Breite und Höhe angeben müssen, insbesondere die Höhe. Genauer gesagt, wir haben bereits den "Ausgangspunkt", der Rest besteht darin, Dinge wie für die Höhe zu verwenden:

```
.attr("height", function(d){
  return (height - yScale(d.value))
})
```

Das Element

`<svg>` -Element ist das Stammelement oder die Leinwand, wenn wir Diagramme darauf zeichnen.

SVG-Elemente können ineinander geschachtelt werden. Auf diese Weise können SVG-Formen in Gruppen zusammengefasst werden. Gleichzeitig werden alle in einem Element verschachtelten Formen relativ zu dem umgebenden Element positioniert.

Eine Sache, die wir vielleicht erwähnen müssen, ist, dass wir nicht in einem anderen nisten können, es funktioniert nicht.

Fall: Mehrere Diagramme

Beispielsweise besteht dieses Mehrfach-Donuts- Diagramm aus mehreren Elementen, die jeweils ein Donut-Diagramm enthalten. Dies kann durch die Verwendung des Elements erreicht werden. In diesem Fall ist es jedoch zweckmäßiger, Donut-Diagramme nacheinander nebeneinander zu platzieren.

Dabei ist zu beachten, dass wir das Attribut transform nicht verwenden können, aber x, y als Position verwenden können.

Das Element

SVG unterstützt derzeit keine automatischen Zeilenumbrüche oder Zeilenumbrüche, das ist die Rettung. Das Element positioniert neue Textzeilen in Bezug auf die vorherige Textzeile. Durch die Verwendung von dx oder dy in jedem dieser Bereiche können wir das Wort in Bezug auf das Wort davor positionieren.

Fall: Anmerkung zu den Achsen

Wenn wir zum Beispiel eine Anmerkung zu y Ax hinzufügen wollen:

```
svg.append("g")
  .attr("class", "y axis")
  .call(yAxis)
.append("text")
  .attr("transform", "rotate(-90)")
  .attr("y", 6)
  .attr("dy", ".71em")
  .style("text-anchor", "end")
  .text("Temperature (°F)");
```

Ein SVG-Element korrekt anhängen

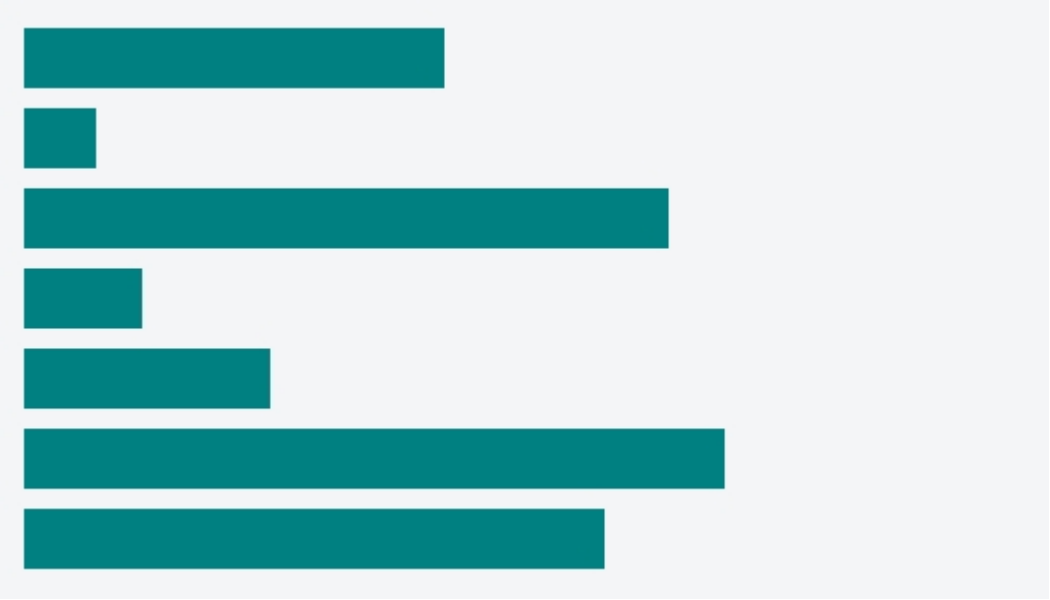
Dies ist ein relativ häufiger Fehler: Sie haben beispielsweise ein `rect` Element in einem Balkendiagramm erstellt und möchten eine `rect` hinzufügen (beispielsweise den Wert dieses Balkens). Also, die gleiche Variable , die Sie verwendet , um die anhängen `rect` und definieren seine `x` und `y` Position, fügen Sie Ihren `text` - Element. Sehr logisch, denkst du vielleicht. Das wird aber nicht funktionieren.

Wie tritt dieser Fehler auf?

Sehen wir uns ein konkretes Beispiel an, einen sehr einfachen Code zum Erstellen eines Balkendiagramms (hier klicken):

```
var data = [210, 36, 322, 59, 123, 350, 290];

var width = 400, height = 300;

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

var bars = svg.selectAll(".myBars")
    .data(data)
    .enter()
    .append("rect");

bars.attr("x", 10)
    .attr("y", function(d,i){ return 10 + i*40})
    .attr("width", function(d){ return d})
    .attr("height", 30);
```

Was uns zu diesem Ergebnis führt:



Sie möchten jedoch einige Textelemente hinzufügen, vielleicht einen einfachen Wert für jeden Balken. Also machst du das:

```
bars.append("text")
   .attr("x", 10)
   .attr("y", function(d,i){ return 10 + i*40})
   .text(function(d){ return d});
```

Und voilà: nichts passiert! Wenn Sie daran zweifeln, ist hier die Geige .

"Aber ich sehe den Tag!"

Wenn Sie sich die SVG ansehen, die mit diesem letzten Code erstellt wurde, sehen Sie Folgendes:

```
▼<rect x="10" y="250" width="290" height="30">
    <text x="10" y="250">290</text>
  </rect>
```

Und an diesem Punkt sagen viele Leute: "Aber ich sehe den Text-Tag, der angehängt ist!". Ja, aber das heißt nicht, dass es funktionieren wird. Sie können *alles* anhängen! Sehen Sie dies zum Beispiel:

```
svg.append("crazyTag");
```

Es wird Ihnen dieses Ergebnis geben:

```
<svg>
    <crazyTag></crazyTag>
</svg>
```

Aber Sie erwarten kein Ergebnis, nur weil der Tag da ist, oder?

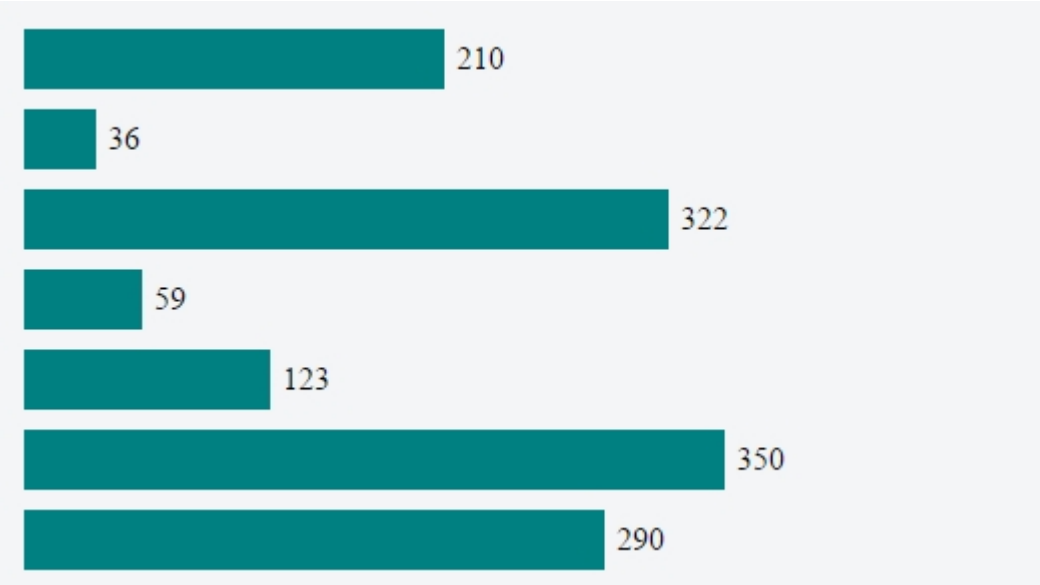
SVG-Elemente richtig anhängen

Erfahren Sie, welche SVG-Elemente Kinder aufnehmen können, indem Sie die Spezifikationen lesen. In unserem letzten Beispiel funktioniert der Code nicht, da `rect` Elemente keine `text` enthalten können. Wie können wir unsere Texte anzeigen?

Erstellen Sie eine weitere Variable und hängen Sie den `text` an die SVG-Datei an:

```
var texts = svg.selectAll(".myTexts")
    .data(data)
    .enter()
    .append("text");

texts.attr("x", function(d){ return d + 16})
    .attr("y", function(d,i){ return 30 + i*40})
    .text(function(d){ return d});
```

Und das ist das Ergebnis:



Und hier ist die Geige .

SVG: die Zeichnungsreihenfolge

Dies kann etwas frustrierend sein: Sie machen eine Visualisierung mit D3.js, aber das Rechteck, das Sie oben möchten, ist hinter einem anderen Rechteck verborgen, oder die Linie, die Sie hinter einem Kreis geplant haben, ist darüber. Sie versuchen dies mit dem *Z-Index* in Ihrem CSS zu lösen, aber es funktioniert nicht (in SVG 1.1).

Die Erklärung ist einfach: In einem SVG legt die Reihenfolge der Elemente die Reihenfolge des "Gemäldes" fest, und die Reihenfolge des Gemäldes bestimmt, wer oben steht.

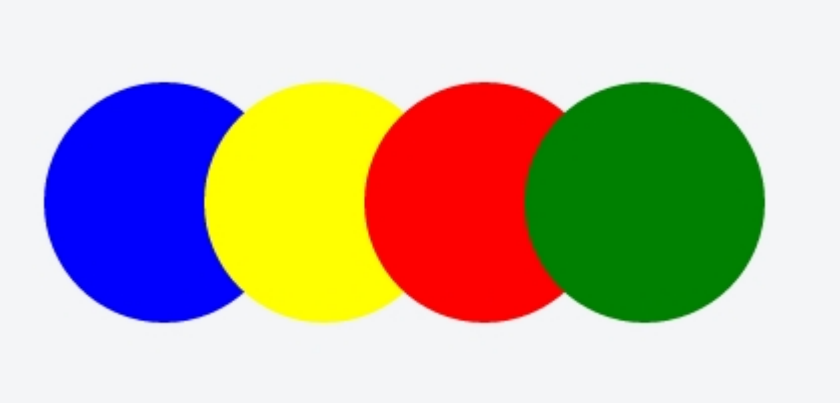
> Elemente in einem SVG-Dokumentfragment haben eine implizite Zeichenreihenfolge, wobei die ersten Elemente im SVG-Dokumentfragment zuerst "gemalt" werden. Nachfolgende Elemente werden auf zuvor gemalte Elemente gemalt.

Nehmen wir an, wir haben diese SVG:

```
<svg width="400" height=200>
    <circle cy="100" cx="80" r="60" fill="blue"></circle>
    <circle cy="100" cx="160" r="60" fill="yellow"></circle>
    <circle cy="100" cx="240" r="60" fill="red"></circle>
    <circle cy="100" cx="320" r="60" fill="green" z-index="-1"></circle>
</svg>
```

Er hat vier Kreise. Der blaue Kreis ist der erste, der "gemalt" ist, daher wird er unter allen anderen angezeigt. Dann haben wir die Gelbe, dann die Rote und schließlich die Grüne. Der grüne ist der letzte und wird oben sein.

So sieht es aus:



Ändern der Reihenfolge der SVG-Elemente mit D3

Kann man also die Reihenfolge der Elemente ändern? Kann ich den roten Kreis vor dem grünen Kreis machen?

Ja. Der erste Ansatz, den Sie berücksichtigen müssen, ist die Reihenfolge der Zeilen in Ihrem Code: Zeichnen Sie zuerst die Elemente des Hintergrunds und später im Code die Elemente des Vordergrunds.

Aber wir können die Reihenfolge der Elemente auch nach dem Bemalen dynamisch ändern. Es gibt mehrere einfache JavaScript-Funktionen, die Sie schreiben können, um dies zu tun, aber D3 hat bereits zwei nette Features, `selection.raise()` und `selection.lower()` .

Gemäß der API:

> selection.raise (): Fügt jedes ausgewählte Element der Reihe nach als letztes untergeordnetes Element des übergeordneten Elements ein. selection.lower (): Fügt jedes ausgewählte Element der Reihe nach als erstes untergeordnetes Element des übergeordneten Elements ein.

Um zu zeigen, wie Sie die Reihenfolge der Elemente in unserer vorherigen SVG-Datei ändern können, haben wir hier einen sehr kleinen Code:

```
d3.selectAll("circle").on("mouseover", function(){
    d3.select(this).raise();
});
```

Was tut es? Es wählt alle Kreise aus, und wenn der Benutzer über einen Kreis fährt, wählt er diesen Kreis aus und bringt ihn nach vorne. Sehr einfach!

Und hier ist das JSFiddle mit dem Live-Code.

Die wichtigsten SVG-Konzepte, die in der D3.js-Visualisierung verwendet werden online lesen: https://riptutorial.com/de/d3-js/topic/2537/die-wichtigsten-svg-konzepte--die-in-der-d3-js-visualisierung-verwendet-werden

Kapitel 7: Ereignisse mit d3.dispatch absenden

Syntax

- d3. **Versand** - Erstellen Sie einen benutzerdefinierten Ereignisverteiler.
- Versand. **on** - Einen Event Listener registrieren oder die Registrierung aufheben.
- Versand. **Kopie** - Erstellen Sie eine Kopie eines Dispatchers.
- Versand. **call** - sendet eine Veranstaltung an registrierte Listener.
- Versand. **bewerben** - Versenden Sie eine Veranstaltung an registrierte Hörer.

Bemerkungen

Dispatching ist ein praktischer Mechanismus zum Trennen von Problemen mit locker gekoppeltem Code: Registrieren Sie benannte Callbacks und rufen Sie sie dann mit beliebigen Argumenten auf. Eine Vielzahl von D3-Komponenten, wie z. B. d3-request, verwenden diesen Mechanismus, um Ereignisse an Listener zu senden. Stellen Sie sich dies wie den EventEmitter von Node vor, mit der Ausnahme, dass jeder Listener einen genau definierten Namen hat, sodass er leicht entfernt oder ersetzt werden kann.

Verwandte Lesungen

- Dispatching Events von Mike Bostock
- d3.dispatch Dokumentation
- Versandereignis im NPM

Examples

einfache Benutzung

```
var dispatch = d3.dispatch("statechange");

dispatch.on('statechange', function(e){ console.log(e) })

setTimeout(function(){dispatch.statechange('Hello, world!')}, 3000)
```

Ereignisse mit d3.dispatch absenden online lesen: https://riptutorial.com/de/d3-js/topic/3399/ereignisse-mit-d3-dispatch-absenden

Kapitel 8: Robust, ansprechend und wieder verwendbar (r3) für d3

Einführung

d3 ist eine leistungsstarke Bibliothek zum Erstellen interaktiver Diagramme. Diese Leistung ergibt sich jedoch aus Benutzern, die auf einer niedrigeren Ebene arbeiten müssen als andere interaktive Bibliotheken. Viele der Beispiele für d3-Diagramme zeigen daher, wie ein bestimmtes Objekt erzeugt wird - z. B. Whisker für eine Box und ein Whisker-Plot -, während die Parameter häufig hart codiert werden, wodurch der Code unflexibel wird. In dieser Dokumentation wird gezeigt, wie Sie wiederverwendbaren Code erstellen können, um in der Zukunft Zeit zu sparen.

Examples

Streudiagramm

Dieses Beispiel enthält insgesamt mehr als 1000 Codezeilen (zu viel, um hier eingebettet zu werden). Aus diesem Grund ist der gesamte Code unter http://blockbuilder.org/SumNeuron/956772481d4e625eec9a59fdb9fbe5b2 verfügbar (alternativ unter https://bl.ocks.org/SumNeuron/956772481d4e625eec9a59fdb9fbe5b2) . Beachten Sie, dass bl.ocks.org iframe verwendet. Um die Größenänderung zu sehen, klicken Sie auf die Schaltfläche Öffnen (untere rechte Ecke des iframe). Da es viel Code gibt, wurde er in mehrere Dateien aufgeteilt, und das entsprechende Codesegment wird sowohl nach Dateiname als auch nach Zeilennummer referenziert. Bitte öffnen Sie dieses Beispiel, während wir es durchgehen.

Was macht ein Diagramm aus?

Es gibt mehrere Kernkomponenten, die in ein vollständiges Diagramm einfließen. Dazu gehören nämlich:

- Titel
- Achsen
- Achsenbeschriftungen
- die Daten

Je nach Diagramm können andere Aspekte enthalten sein, z. B. eine Diagrammlegende. Viele dieser Elemente können jedoch mit dem Tooltip umgangen werden. Aus diesem Grund gibt es interaktive Diagramm-spezifische Elemente, z. B. Schaltflächen zum Wechseln zwischen Daten.

Da der Inhalt unseres Diagramms interaktiv ist, ist es angebracht, dass das Diagramm selbst dynamisch ist - z. B. Größe ändern, wenn sich die Fenstergröße ändert. SVG ist skalierbar, Sie können also die Skalierung Ihres Diagramms unter Beibehaltung der aktuellen Perspektive

zulassen. Abhängig von der eingestellten Perspektive kann das Diagramm jedoch zu klein werden, um lesbar zu sein, selbst wenn noch genügend Platz für das Diagramm vorhanden ist (z. B. wenn die Breite größer als die Höhe ist). Daher kann es wünschenswert sein, das Diagramm nur in der verbleibenden Größe neu zu zeichnen.

In diesem Beispiel wird beschrieben, wie Sie die Platzierung der Schaltflächen, Titel, Achsen, Achsenbeschriftungen dynamisch berechnen und Datensätze mit unterschiedlichen Datenmengen behandeln

---

Konfiguration

Aufbau

Da wir die Wiederverwendung von Code anstreben, sollten wir eine Konfigurationsdatei erstellen, die globale Optionen für Aspekte unseres Diagramms enthält. Ein Beispiel für eine solche Konfigurationsdatei ist `charts_configuration.json` .

Wenn wir uns diese Datei ansehen, können wir feststellen, dass ich einige Elemente eingefügt habe, die bereits klar sein sollten, wenn wir unser Diagramm erstellen:

- Dateien (speichert die Zeichenfolge, in der sich unsere Diagrammdaten befinden)
- document_state (welche Schaltfläche ist aktuell für unser Diagramm ausgewählt)
- chart_ids (HTML-IDs für die Diagramme, die wir erstellen werden)
- svg (Optionen für das svg, zB Größe)
- plot_attributes
    - Titel (verschiedene Schriftattribute setzen)
    - Tooltip (verschiedene Tooltip-Stileigenschaften festlegen)
    - Achsen (verschiedene Schriftattribute setzen)
    - Schaltflächen (verschiedene Schrift- und Stilattribute festlegen)
- Handlungen
    - Streuung (verschiedene Aspekte unseres Streudiagramms einstellen, z. B. Punktradius)
- Farben (eine bestimmte Farbpalette, die verwendet werden soll)

Hilfsfunktionen

Neben dem Einrichten dieser globalen Aspekte müssen einige Hilfsfunktionen definiert werden. Diese finden Sie unter `helpers.js`

- `ajax_json` (`ajax_json` Dateien entweder synchron oder asynchron laden)
- `keys` (gibt Schlüssel des gegebenen Objekts zurück - entspricht d3.keys ())
- `parseNumber` (eine allgemeine `parseNumber` falls wir nicht wissen, welcher Typ oder welche Nummer ist)
- `typeofNumber` (gibt den Nummerntyp zurück)

index.html

Zuletzt sollten wir unsere HTML-Datei einrichten. In diesem Beispiel werden wir unser Diagramm in ein `section` Tag einfügen, wobei die `id` mit der in der Konfigurationsdatei angegebenen `id` übereinstimmt (Zeile 37). Da Prozentsätze nur funktionieren, wenn sie von ihrem übergeordneten Mitglied berechnet werden können, enthalten wir auch einige Grundstile (Zeilen 19-35).

Unser Streudiagramm erstellen

Lassen Sie uns `make_scatter_chart.js` öffnen. Lassen Sie uns nun auf Zeile 2 achten, in der viele der wichtigsten Variablen vordefiniert sind:

- svg - d3 Auswahl der svg des Diagramms
- chart_group - d3 Auswahl der Gruppe innerhalb der SVG, in der die Daten platziert werden
- Canvas - Kernaspekte des SVG-Extrakts für mehr Komfort
- Margen - die Margen müssen wir berücksichtigen
- maxi_draw_space die größten x- und y-Werte, in denen wir unsere Daten zeichnen können
- doc_state - der aktuelle Status des Dokuments, wenn wir Schaltflächen verwenden (in diesem Beispiel sind wir dies)

Möglicherweise haben Sie bemerkt, dass wir die SVG-Datei nicht in die HTML-Datei aufgenommen haben. Deshalb müssen wir, bevor wir etwas mit unserem Diagramm tun können, die svg zu `index.html` hinzufügen, falls sie noch nicht existiert. Dies wird in der Datei `make_svg.js` durch die Funktion `make_chart_svg` . Wenn Sie `make_svg.js` , sehen wir, dass wir in der Diagrammkonfiguration die `parseNumber` für die Breite und Höhe der `parseNumber` . Wenn es sich bei der Zahl um einen Float handelt, wird die Breite und Höhe des SVG proportional zur Breite und Höhe des Abschnitts. Wenn es sich bei der Zahl um eine Ganzzahl handelt, wird sie nur auf diese Ganzzahlen gesetzt.

Zeile 6 - 11 prüft, ob es wirksam ist - ob dies der erste Aufruf ist oder nicht, und legt die `chart_group` (und den Dokumentstatus, wenn es sich um den ersten Aufruf handelt) fest.

Zeile 14 - 15 extrahiert die aktuell ausgewählten Daten durch Klicken auf die Schaltfläche; Zeile 16 legt `data_extent` . Während d3 eine Funktion zum Extrahieren des Datenbereichs hat, ziehe *ich* es vor, den Datenbereich in dieser Variablen zu speichern.

Die Zeilen 27 - 38 enthalten die Magie, durch die die Ränder, die Schaltflächen, der Titel und die Achsen erstellt werden. Diese sind alle dynamisch bestimmt und wirken ein wenig komplex, so dass wir uns nacheinander anschauen.

make_margins (in make_margins.js)

Wir können sehen, dass das Rands-Objekt etwas Platz auf der linken, rechten, oberen und unteren Seite des Diagramms (x.left, x.right, y.top, y.bottom), den Titel, die Schaltflächen und die Äxte

Wir sehen auch, dass die Achsenränder in Zeile 21 aktualisiert werden.

Warum machen wir das? Wenn wir nicht die Anzahl der Ticks angeben, der Tick die Tickgröße und die Schriftgröße des Ticklabels, können wir die Größe nicht berechnen, die die Achsen benötigen. Selbst dann müssten wir noch den Abstand zwischen den Tick-Labels und den Ticks schätzen. Daher ist es einfacher, einige Dummy-Achsen mit unseren Daten zu erstellen, zu sehen, wie groß die entsprechenden svg-Elemente sind, und dann die Größe zurückzugeben.

Wir brauchen eigentlich nur die Breite der Y-Achse und die Höhe der X-Achse, die in Achsen.y und Achsen.x gespeichert ist.

Mit unseren Standardrändern berechnen wir dann den `max_drawing_space` (Zeilen 29-34 in make_margins.js).

make_buttons (in make_buttons.js)

Die Funktion erstellt eine Gruppe für alle Schaltflächen und dann eine Gruppe für jede Schaltfläche, die wiederum einen Kreis und ein Textelement speichert. Zeile 37 - 85 berechnet die Position der Tasten. Dies geschieht, indem Sie sehen, ob der Text rechts von jeder Schaltfläche länger ist als der Platz, den wir einzeichnen können (Zeile 75). Wenn dies der Fall ist, wird die Schaltfläche um eine Zeile nach unten verschoben und die Ränder werden aktualisiert.

make_title (in make_title.js)

`make_title` ähnelt make_buttons dahingehend, dass der Titel Ihres Diagramms automatisch in mehrere Zeilen aufgeteilt wird und bei Bedarf ein Bindestrich ist. Es ist ein bisschen hackig, da es nicht die Komplexität des TeX-Trennungsschemas besitzt, aber es funktioniert ausreichend gut. Wenn wir mehr Zeilen als eine benötigen, werden die Ränder aktualisiert.

Wenn die Schaltflächen, Titel und Ränder festgelegt sind, können wir unsere Achsen erstellen.

make_axes (in make_axes.js)

Die Logik von make_axes spiegelt die zur Berechnung des Platzbedarfs der Dummy-Achsen wider. Hier werden jedoch Übergänge hinzugefügt, um zwischen den Achsen zu wechseln.

Zum Schluss unser Streudiagramm

Wenn alles fertig ist, können wir endlich unser Streudiagramm erstellen. Da unsere Datensätze eine unterschiedliche Anzahl von Punkten aufweisen können, müssen wir dies berücksichtigen und die Eintritts- und Beendigungsereignisse von d3 entsprechend nutzen. Die Anzahl der bereits vorhandenen Punkte wird in Zeile 40 ermittelt. Die if-Anweisung in Zeile 45 - 59 fügt mehr Kreiselemente hinzu, wenn mehr Daten vorhanden sind, oder übergibt die zusätzlichen Elemente in eine Ecke und entfernt sie, wenn zu viele vorhanden sind.

Sobald wir wissen, dass wir die richtige Anzahl von Elementen haben, können wir alle verbleibenden Elemente an ihre korrekte Position bringen (Zeile 64).

Zuletzt fügen wir Tooltip in Zeile 67 und 68 hinzu. Die Tooltip-Funktion befindet sich in `make_tooltip.js`

Box- und Whisker-Chart

Um den Wert der Erstellung verallgemeinerter Funktionen wie im vorherigen Beispiel (make_title, make_axes, make_buttons usw.) darzustellen, beachten Sie dieses Kästchen und dieses Whisker-Diagramm: https://bl.ocks.org/SumNeuron/262e37e2f932cf4b693f241c52a410ff

Während der Code zum Erstellen der Boxen und Whiskers intensiver ist als das einfache Platzieren der Punkte, sehen wir, dass dieselben Funktionen perfekt funktionieren.

Balkendiagramm

https://bl.ocks.org/SumNeuron/7989abb1749fc70b39f7b1e8dd192248

Robust, ansprechend und wieder verwendbar (r3) für d3 online lesen: https://riptutorial.com/de/d3-js/topic/9849/robust--ansprechend-und-wieder-verwendbar--r3--fur-d3

Kapitel 9: SVG-Diagramme mit D3 js

Examples

Verwenden von D3-js zum Erstellen von SVG-Elementen

Obwohl D3 nicht spezifisch für die Handhabung von SVG-Elementen ist, wird es häufig zum Erstellen und Bearbeiten komplexer SVG-basierter Datenvisualisierungen verwendet. D3 bietet viele leistungsstarke Methoden, mit deren Hilfe Sie problemlos verschiedene geometrische SVG-Strukturen erstellen können.

Es wird empfohlen, zunächst die grundlegenden Konzepte der SVG-Spezifikationen zu verstehen und dann umfangreiche D3-Beispiele zu verwenden, um Visualisierungen zu erstellen.

D3 js Beispiele

Grundlagen von SVG

SVG-Diagramme mit D3 js online lesen: https://riptutorial.com/de/d3-js/topic/1538/svg-diagramme-mit-d3-js

Kapitel 10: Verwenden von D3 mit JSON und CSV

Syntax

- d3.csv (URL [[, Zeile], Rückruf])
- d3.tsv (URL [[, Zeile], Rückruf])
- d3.html (URL [, Rückruf])
- d3.json (URL [, Rückruf])
- d3.text (URL [, Rückruf])
- d3.xml (URL [, Rückruf])

Examples

Laden von Daten aus CSV-Dateien

Es gibt verschiedene Möglichkeiten, die Daten abzurufen, die Sie an die DOM-Elemente binden möchten. Der einfachere ist, Ihre Daten als Array in Ihrem Skript zu haben ...

```
var data = [ ... ];
```

Mit **D3.js** können wir jedoch Daten aus einer externen Datei laden. In diesem Beispiel wird gezeigt, wie Daten aus einer CSV-Datei ordnungsgemäß geladen und verarbeitet werden.

CSV-Dateien sind durch *Kommas getrennte Werte* . In dieser Art von Datei ist jede Zeile ein Datensatz, wobei jeder Datensatz aus einem oder mehreren Feldern besteht, die durch Kommas getrennt sind. Es ist wichtig zu wissen, dass die Funktion, die wir verwenden werden, `d3.csv` , die erste Zeile der CSV als **Header verwendet** , d. `d3.csv` Die Zeile, die die Namen der Felder enthält.

Betrachten Sie diese CSV mit dem Namen "data.csv":

```
city,population,area
New York,3400,210
Melbourne,1200,350
Tokyo,5200,125
Paris,800,70
```

Um "data.csv" zu laden, verwenden wir die Funktion `d3.csv` . Angenommen, "data.csv" befindet sich im selben Verzeichnis unseres Skripts und der relative Pfad ist einfach "data.csv". Also schreiben wir:

```
d3.csv("data.csv", function(data){
    //code dealing with data here
});
```

Beachten Sie, dass wir im Callback `data` als Argument verwendet haben. Dies ist eine übliche Praxis in D3, aber Sie können jeden anderen Namen verwenden.

Was macht `d3.csv` mit unserer CSV? Es konvertiert die CSV in ein Array von Objekten. Wenn wir zum `console.log` unsere Daten in einer `console.log` :

```
d3.csv("data.csv", function(data){
    console.log(data)
});
```

Das werden wir sehen:

```
[
    {
        "city": "New York",
        "population": "3400",
        "area": "210"
    },{
        "city": "Melbourne",
        "population": "1200",
        "area": "350"
    },{
        "city": "Tokyo",
        "population": "5200",
        "area": "125"
    },{
        "city": "Paris",
        "population": "800",
        "area": "70"
    }
]
```

Jetzt können wir diese Daten an unsere DOM-Elemente binden.

Beachten Sie, dass in diesem Beispiel `population` und `area` Zeichenfolgen sind. Aber wahrscheinlich möchten Sie sie als Zahlen behandeln. Sie können sie in einer Funktion innerhalb des Callbacks (als `forEach`) `d3.csv` , aber in `d3.csv` Sie eine "Accessor" -Funktion verwenden:

```
d3.csv("data.csv", conversor, function(data){
    //code here
});

function conversor(d){
    d.population = +d.population;
    d.area = +d.area;
    return d;
}
```

Sie können Accessoren auch in `d3.tsv` , jedoch nicht in `d3.json` .

Hinweis: `d3.csv` ist eine asynchrone Funktion. Das bedeutet, dass der Code unmittelbar danach ausgeführt wird, sogar bevor die CSV-Datei geladen wird. Daher ist die Verwendung Ihrer `data` im Callback besonders zu beachten.

Ein oder zwei Parameter im Rückruf - Fehlerbehandlung in d3.request ()

Wenn Sie `d3.request()` oder einen der Convenience-Konstruktoren (d3.json , d3.csv , d3.tsv , d3.html und d3.xml) verwenden, gibt es viele Fehlerquellen. Möglicherweise gibt es Probleme mit der ausgegebenen Anforderung oder deren Antwort aufgrund von Netzwerkfehlern oder die Analyse kann fehlschlagen, wenn der Inhalt nicht ordnungsgemäß formatiert ist.

Innerhalb der Rückrufe, die an eine der oben genannten Methoden übergeben werden, ist es daher wünschenswert, eine Fehlerbehandlung zu implementieren. Zu diesem Zweck können die Rückrufe zwei Argumente akzeptieren, wobei das erste der Fehler ist, falls vorhanden, das zweite die Daten. Wenn beim Laden oder Parsen ein Fehler aufgetreten ist, werden die Informationen zum Fehler als erster `error` mit den `data` als `null` .

```
d3.json{"some_file.json", function(error, data) {
  if (error) throw error;   // Exceptions handling
  // Further processing if successful
});
```

Sie sind auf keinen Fall verpflichtet, zwei Parameter anzugeben. Es ist vollkommen in Ordnung, die Anforderungsmethoden mit einem Rückruf zu verwenden, der nur einen Parameter enthält. Für diese Art von Rückrufen gibt es eine private Funktion `fixCallback()` in request.js, die die Art und Weise angibt, in der Informationen an das einzelne Argument der Methode übergeben werden.

```
function fixCallback(callback) {
  return function(error, xhr) {
    callback(error == null ? xhr : null);
  };
}
```

Dies wird von D3 für alle Rückrufe aufgerufen, die nur einen Parameter haben, der definitionsgemäß die Daten sind.

Egal , wie viele Parameter an die Callback - Anforderung Verfahren zugeführt werden, die Regel für den `data` ist:

- Wenn die Anforderung fehlschlägt, sind die `data` `null`
- Wenn die Anforderung erfolgreich ist, enthalten die `data` den geladenen (und analysierten) Inhalt

Der einzige Unterschied zwischen der Ein-Parameter- und der Zwei-Parameter-Version besteht in der Art und Weise, in der Informationen über den Fehler bereitgestellt werden, der möglicherweise auftritt. Wenn der `error` angegeben wird, schlägt die Methode automatisch fehl, und die `data` als `null` . Wenn der Rückruf dagegen mit zwei Argumenten definiert ist, werden Informationen über einen Fehler beim Laden oder Parsen an den ersten Parameter übergeben, damit Sie ihn entsprechend behandeln können.

Die folgenden vier Aufrufe von `d3.json` die möglichen Szenarien für vorhandene / nicht vorhandene Dateien im Vergleich zu einem Parameter / zwei Parameter-Rückrufen:

```
// FAIL: resource not available or content not parsable
// error contains information about the error
// data will be null because of the error
d3.json("non_existing_file.json", function(error, data) {
  console.log("Fail, 2 parameters: ", error, data);
});

// FAIL: resource not available or content not parsable
// no information about the error
// data will be null because of the error
d3.csv("non_existing_file.json", function(data) {
  console.log("Fail, 1 parameter: ", data);
});

// OK: resource loaded successfully
// error is null
// data contains the JSON loaded from the resource
d3.json("existing_file.json", function(error, data) {
  console.log("OK, 2 parameters: ", error, data);
});

// OK: resource loaded successfully
// no information about the error; this fails silently on error
// data contains the JSON loaded from the resource
d3.json("existing_file.json", function(data) {
  console.log("OK, 1 parameter: ", data);
});
```

Verwenden von D3 mit JSON und CSV online lesen: https://riptutorial.com/de/d3-js/topic/5201/verwenden-von-d3-mit-json-und-csv

Kapitel 11: Verwendung von D3 mit anderen Frameworks

Examples

D3.js-Komponente mit ReactJS

Dieses Beispiel basiert auf einem Blogeintrag von Nicolas Hery . Es verwendet ES6-Klassen und die Lebenszyklusmethoden von ReactJS, um die D3-Komponente auf dem neuesten Stand zu halten

---

d3_react.html

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <title>Hello, d3React!</title>
  <style>
    .d3Component {
      width: 720px;
      height: 120px;
    }
  </style>
</head>
<script src="https://fb.me/react-15.2.1.min.js"></script>
<script src="https://fb.me/react-dom-15.2.1.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/babel-
core/5.8.34/browser.min.js"></script>
<script src="https://d3js.org/d3.v4.min.js"></script>

<body>
  <div id="app" />
  <script type="text/babel" src="d3_react.js"></script>
</body>

</html>
```

d3_react.js

```
class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      d3React: new d3React()
    };
    this.getd3ReactState = this.getd3ReactState.bind(this);
  }
```

```
  getd3ReactState() {
    // Using props and state, calculate the d3React state
    return ({
      data: {
        x: 0,
        y: 0,
        width: 42,
        height: 17,
        fill: 'red'
      }
    });
  }

  componentDidMount() {
    var props = {
      width: this._d3Div.clientWidth,
      height: this._d3Div.clientHeight
    };
    var state = this.getd3ReactState();
    this.state.d3React.create(this._d3Div, props, state);
  }

  componentDidUpdate(prevProps, prevState) {
    var state = this.getd3ReactState();
    this.state.d3React.update(this._d3Div, state);
  }

  componentWillUnmount() {
    this.state.d3React.destroy(this._d3Div);
  }

  render() {
    return (
      <div>
        <h1>{this.props.message}</h1>
        <div className="d3Component" ref={(component) => { this._d3Div = component; } } />
      </div>
    );
  }
}

class d3React {
  constructor() {
    this.create = this.create.bind(this);
    this.update = this.update.bind(this);
    this.destroy  = this.destroy.bind(this);
    this._drawComponent = this._drawComponent.bind(this);
  }

  create(element, props, state) {
    console.log('d3React create');
    var svg = d3.select(element).append('svg')
      .attr('width', props.width)
      .attr('height', props.height);

    this.update(element, state);
  }

  update(element, state) {
    console.log('d3React update');
```

```
    this._drawComponent(element, state.data);
  }

  destroy(element) {
    console.log('d3React destroy');
  }

  _drawComponent(element, data) {
    // perform all drawing on the element here
    var svg = d3.select(element).select('svg');

    svg.append('rect')
      .attr('x', data.x)
      .attr('y', data.y)
      .attr('width', data.width)
      .attr('height', data.height)
      .attr('fill', data.fill);
  }
}

ReactDOM.render(<App message="Hello, D3.js and React!"/>, document.getElementById('app'));
```

`d3_react.html` `d3_react.js` den Inhalt von `d3_react.html` und `d3_react.js` im selben Verzeichnis ab und navigieren Sie mit einem Webbrowser zur Datei d3React.html. Wenn alles gut geht, sehen Sie eine Kopfzeile mit `Hello, D3.js and React!` gerendert aus der React-Komponente und einem roten Rechteck darunter aus der benutzerdefinierten D3-Komponente.

React verwendet refs , um die Komponenteninstanz zu erreichen. Die Lebenszyklusmethoden der `d3React` Klasse erfordern, dass dieser `d3React` DOM-Elemente `d3React` , ändert und entfernt. Die `d3React` Klasse kann erweitert werden, um weitere benutzerdefinierte Komponenten zu erstellen, und an einer `div.d3Component` Stelle eingefügt, an der eine `div.d3Component` von React erstellt wird.

D3js mit Winkel

Die Verwendung von D3js mit Angular kann neue Möglichkeiten eröffnen, z. B. die Live-Aktualisierung von Diagrammen, sobald die Daten aktualisiert werden. Wir können die vollständige Diagrammfunktionalität in einer Angular-Direktive kapseln, sodass sie leicht wiederverwendbar ist.

index.html >>

```
<!DOCTYPE html>
<html ng-app="myApp">
<head>
  <script src="https://d3js.org/d3.v4.min.js"></script>
  <script data-require="angular.js@1.4.1" data-semver="1.4.1"
src="https://code.angularjs.org/1.4.1/angular.js"></script>
  <script src="app.js"></script>
  <script src="bar-chart.js"></script>
</head>

<body>

  <div ng-controller="MyCtrl">
```

```
    <!-- reusable d3js bar-chart directive, data is sent using isolated scope -->
    <bar-chart data="data"></bar-chart>
  </div>

</body>
</html>
```

Wir können die Daten mithilfe eines Controllers an das Diagramm übergeben und auf Änderungen in den Daten achten, um eine Live-Aktualisierung des Diagramms in der Anweisung zu ermöglichen:

app.js >>

```
angular.module('myApp', [])
  .controller('MyCtrl', function($scope) {
    $scope.data = [50, 40, 30];
    $scope.$watch('data', function(newVal, oldVal) {
      $scope.data = newVal;
    }, true);
  });
```

Schließlich die Definition der Richtlinie. Der Code, den wir zum Erstellen und Bearbeiten des Diagramms schreiben, wird in der Link-Funktion der Direktive gespeichert.

Beachten Sie, dass wir auch einen Geltungsbereich. $ Watch in die Direktive eingefügt haben, der aktualisiert wird, sobald der Controller neue Daten übergibt. Wir ordnen unserer Datenvariablen neue Daten zu, wenn sich Daten ändern, und rufen dann die Funktion repaintChart () auf, die das erneute Rendern des Diagramms durchführt.

bar-chart.js >>

```
angular.module('myApp').directive('barChart', function($window) {
  return {
    restrict: 'E',
    replace: true,
    scope: {
      data: '='
    },
    template: '<div id="bar-chart"></div>',
    link: function(scope, element, attrs, fn) {

      var data = scope.data;
      var d3 = $window.d3;
      var rawSvg = element;

      var colors = d3.scale.category10();

      var canvas = d3.select(rawSvg[0])
        .append('svg')
        .attr("width", 300)
        .attr("height", 150);

      // watching for any changes in the data
      // if new data is detected, the chart repaint code is run
      scope.$watch('data', function(newVal, oldVal) {
        data = newVal;
```

```
        repaintChart();
      }, true);

      var xscale = d3.scale.linear()
        .domain([0, 100])
        .range([0, 240]);

      var yscale = d3.scale.linear()
        .domain([0, data.length])
        .range([0, 120]);

      var bar = canvas.append('g')
        .attr("id", "bar-group")
        .attr("transform", "translate(10,20)")
        .selectAll('rect')
        .data(data)
        .enter()
        .append('rect')
        .attr("class", "bar")
        .attr("height", 15)
        .attr("x", 0)
        .attr("y", function(d, i) {
          return yscale(i);
        })
        .style("fill", function(d, i) {
          return colors(i);
        })
        .attr("width", function(d) {
          return xscale(d);
        });

      // changing the bar widths according to the changes in data
      function repaintChart() {
        canvas.selectAll('rect')
          .data(data)
          .transition()
          .duration(800)
          .attr("width", function(d) {
            return xscale(d);
          })
      }
    }
  }
});
```

Hier ist das funktionierende JSFiddle.

D3.js-Diagramm mit Angular v1

HTML:

```
<div ng-app="myApp" ng-controller="Controller">
    <some-chart data="data"></some-chart>
</div>
```

Javascript:

```
angular.module('myApp', [])
.directive('someChart', function() {
    return {
        restrict: 'E',
        scope: {data: '=data'},
        link: function (scope, element, attrs) {
        var chartElement = d3.select(element[0]);
            // here you have scope.data and chartElement
            // so you may do what you want
        }
    };
});

function Controller($scope) {
    $scope.data = [1,2,3,4,5]; // useful data
}
```

Verwendung von D3 mit anderen Frameworks online lesen: https://riptutorial.com/de/d3-js/topic/3733/verwendung-von-d3-mit-anderen-frameworks

Kapitel 12: Zu Veranstaltungen

Syntax

- .on ('mouseover', Funktion)
- .on ('mouseout', Funktion)
- .on ('klick', funktion)
- .on ('mouseenter', Funktion)
- .on ('mouseleave', Funktion)

Bemerkungen

Ein ausführlicheres Beispiel, in dem benutzerdefinierte Ereignisse definiert werden, finden Sie hier .

Examples

Anhängen grundlegender Ereignisse an Auswahlen

Oft möchten Sie Ereignisse für Ihre Objekte haben.

```
function spanOver(d,i){
  var span = d3.select(this);
  span.classed("spanOver",true);
}

function spanOut(d,i){
  var span = d3.select(this);
  span.classed("spanOver", false);
}

var div = d3.select('#divID');

div.selectAll('span')
  .on('mouseover' spanOver)
  .on('mouseout' spanOut)
```

In diesem Beispiel wird die Klasse `spanOver` wenn Sie mit der id `divID` über einen Bereich innerhalb des div- `spanOver` schweben, und entfernen, wenn die Maus den Bereich verlässt.

Standardmäßig übergibt d3 das Datum des aktuellen Bereichs und den Index. Es ist wirklich praktisch, dass der Kontext `this` Objekts auch das aktuelle Objekt ist, sodass wir darauf Operationen ausführen können, beispielsweise Klassen hinzufügen oder entfernen.

Sie können auch eine anonyme Funktion für das Ereignis verwenden.

```
div.selectAll('span')
  .on('click', function(d,i){ console.log(d); });
```

Datenelemente können auch zum aktuell ausgewählten Objekt hinzugefügt werden.

```
div.selectAll('path')
  .on('click', clickPath);

function clickPath(d,i) {
  if(!d.active) {
    d.active = true;
    d3.select(this).classed("active", true);
  }
  else {
    d.active = false;
    d3.select(this).classed("active", false);
  }
}
```

In diesem Beispiel ist aktiv nicht für die Auswahl definiert, bevor das Klickereignis ausgelöst wird. Wenn Sie die Pfadauswahl erneut durchführen würden, würden alle angeklickten Objekte den `active` Schlüssel enthalten.

Event-Listener entfernen

d3.js verfügt nicht über eine `.off()` -Methode, um vorhandene Ereignislistener abzugleichen. Um einen Event-Handler zu entfernen, müssen Sie ihn auf `null` :

```
d3.select('span').on('click', null)
```

Zu Veranstaltungen online lesen: https://riptutorial.com/de/d3-js/topic/2722/zu-veranstaltungen

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit d3.js	4444, Adam, altocumulus, Arthur Tarasov, Community, davinov, Fawzan, Gerardo Furtado, Ian H, jmdarling, kashesandr, Marek Skiba, Maximilian Kohl, Rachel Gallen, Ram Visagan, RamenChef, Ruben Helsloot, TheMcMurder, Tim B, winseybash
2	Aktualisierungsmuster	Gerardo Furtado
3	Ansätze zum Erstellen von responsiven d3.js-Diagrammen	Ian H
4	Auswahlmöglichkeiten	Ashitaka, aug, Carl Manaster, Gerardo Furtado, Ian, Ian H, JulCh, Tim B
5	D3-Projektionen	Andrew Reid
6	Die wichtigsten SVG-Konzepte, die in der D3.js-Visualisierung verwendet werden	fengshuo, Gerardo Furtado
7	Ereignisse mit d3.dispatch absenden	aug, kashesandr, Ram Visagan
8	Robust, ansprechend und wieder verwendbar (r3) für d3	SumNeuron
9	SVG-Diagramme mit D3 js	rajarshig
10	Verwenden von D3 mit JSON und CSV	altocumulus, Gerardo Furtado
11	Verwendung von D3 mit anderen Frameworks	Adam, kashesandr, Rishabh
12	Zu Veranstaltungen	aug, Ian H, Kemi, Tim B