THE DEVELOPER'S CONFERENCE

Julho/2016

#dataviz

visualização de big data com d3.js

helder da rocha

summa
Technology + Business

Quem sou eu? Who am I? Кто я?

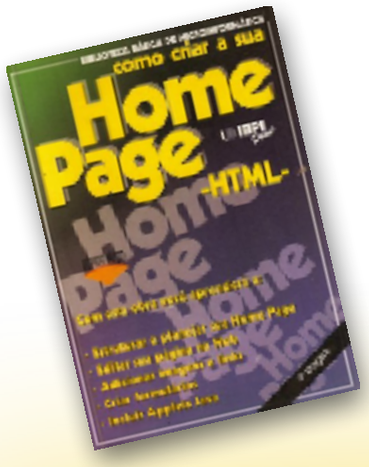
Helder da **Rocha**

Tecnologia * Ciência * Arte

HTML & tecnologias Web desde 1995

Autor de cursos e livros sobre

Java, XML e tecnologias Web



argonavis.com.br

helderdarocha.com.br

Objetivos

- Apresentar **D3.js** e mostrar como pode ser usada para visualizações de **Big Data**
  - Por que usar uma biblioteca de visualização?
  - Como o D3 representa dados graficamente
  - Exemplos de visualizações usando D3
- Problemas ao lidar com milhões de objetos
  - **Estratégias** para usar D3 com big data: navegação/ brushing, canvas, dados sob demanda, algoritmos

#dataviz

- Gerar **representações visuais** dos dados permite
  - **revelar** informações
  - **explorar** dados e encontrar **padrões**
  - identificar **tendências**
- aplicações: business intelligence & analytics, geographical information systems

Como gerar visualizações?

- Ferramentas
  - Ferramentas GIS, ferramentas de BI & analytics, Infogr.am, dashbuilder, BIRT, Qlik, Tableau
- APIs
  - HTML e SVG DOM, **D3.js**, Leaflet, OpenLayers
  - APIs permitem mais flexibilidade, eficiência e facilidade de otimização

- **D3.js** (Data Driven Objects)_ é uma das mais populares **bibliotecas JavaScript** para visualização de dados.

- Criada por Mike Bostock (NY Times)

- D3 transforma dados em **objetos** do **DOM** (Document Object Model)

- Objetos são **renderizados** em HTML ou SVG: pontos, linhas, polígonos, etc.

D3 passo-a-passo

- Inclua **<script>** vinculando página a arquivo d3.js

```
<script src="d3.js"></script>
```

- Carregue os dados:

```
d3.json("dados.json", function(array) { ... }
```

- Selecione objetos para acomodar os dados

```
var selecao = d3.selectAll("div");
```

- Selecione array de dados que serão usados

```
selecao.data(array);
```

- Faça um join dos dados com os objetos gráficos

```
selecao.enter()
```

- Adicione um elemento para **cada** item

```
selecao.append("div")
```

geonames.org

- Contém lista de lugares (arquivo de 1.3GB)
- Lista de cidades (mais de 5000 habitantes)
- TSV

geonameid	name	asciiname	alternatenames	latitude	longitude	feature_class	feature_code	country_code
3039154	El Tarter	El Tarter	Ehl Tarter,Эл Тартер	42.57952	1.65362	P	PPL	AD	02	1052
3039163	Sant Julià de Lòria	Sant Julia de Loria	San Julia,San Julià,Sant Julia de Loria,Sant Julià de Lòria,Sant-Zhulija							
3039604	Pas de la Casa	Pas de la Casa	Pas de la Kasa,Пас де ла Каса	42.54277	1.73361	P	PPL	AD	03	
3039678	Ordino	Ordino	Ordino,ao er di nuo,orudino jiao qu,Ордино,オルディノ教区,奥尔迪诺	42.55623	1.53319	P	PPLA			
3040051	les Escaldes	les Escaldes	Ehskal'des-Ehndzhordani,Escaldes,Escaldes-Engordany,Les Escaldes,esukarudesu=engo							
3040132	la Massana	la Massana	La Macana,La Massana,La Maçana,La-Massana,la Massana,ma sa na,ла-Массана,ラ・マサナ教区,马萨							
3040686	Encamp	Encamp	Ehnkam,Encamp,en kan pu,enkanpu jiao qu,Энкам,エンカンプ教区,恩坎普	42.53474	1.58014	P	PPLA			
3041204	Canillo	Canillo	Canillo,Kanil'o,ka ni e,kaniryo jiao qu,Канильо,カニーリョ教区,卡尼略	42.5676	1.59756	P	PPLA			
3041519	Arinsal	Arinsal	Arinsal,Arinsal',Arinsalis,arynsal,Аринсал,Аринсаль,آرینسال	57205.42	48453.1	P	PPL	AD		
3041563	Andorra la Vella	Andorra la Vella	ALV,Ando-la-Vyey,Andora,Andora la Vela,Andora la Velja,Andora lja Vehl'ja							
290594	Umm al Qaywayn	Umm al Qaywayn	Oumm al Qaiwain,Oumm al Qaïwaïn,Um al Kawain,Um al Quweim,Umm al Qaiwain,Umm al							
291074	Ras al-Khaimah	Ras al-Khaimah	Julfa,Khaimah,RKT,Ra's al Khaymah,Ra's al-Chaima,Ras al Khaimah,Ras al-Khaimah,Ra							
291279	Muzayri‘	Muzayri`	Mezaira'a,Mezaira’a,Mizeir`ah,Mizeir‘ah,Mozayri`,Mozayri‘,Muzairi,Muzayri`,Muzayri‘,Myza							
291696	Khawr Fakkān	Khawr Fakkan	Fakkan,Fakkān,Khawr Fakkan,Khawr Fakkān,Khawr al Fakkan,Khawr al Fakkān,Khor Fak							
292223	Dubai	Dubai	DXB,Dabei,Dibai,Dibay,Doubayi,Dubae,Dubai,Dubai emiraat,Dubaija,Dubaj,Dubajo,Dubajus,Dubay,Dubay							
292231	Dibba Al-Fujairah	Dibba Al-Fujairah	Al-Fujairah,BYB,Dibba Al-Fujairah,dba alfjyrt,دبا الفجيرة	59246.25						
292239	Dibba Al-Hisn	Dibba Al-Hisn	BYB,Daba,Daba al-Hisn,Dabā,Dabā al-Ḥiṣn,Diba,Diba al Hisn,Dibah,Dibba,Dibba Al'-H							
292672	Sharjah	Sharjah	Al Sharjah,Ash 'Mariqah,Ash Shariqa,Ash Shariqah,Ash Shāriqa,Ash Shāriqah,Ash ‘Mariqah,Ash-Shari							
292688	Ar Ruways	Ar Ruways	Ar Ru'ays,Ar Ruways,Ar Ru’ays,Ar-Ruvais,Ruwais,Ар-Рувайс	24.11028	52.73056	P				
292878	Al Fujayrah	Al Fujayrah	Al Fujayrah,Al-Fudjayra,Al-Fujayrah' emiraat,FJR,Fudschaira,Fudzhejra,Fujaira,Fujairah,F							
292913	Al Ain	Al Ain	AAN,Ainas,Al Ain,Al Ajn,Al Ayn,Al `Ayn,Al Ɛayn,Al ‘Ayn,Al-Ain,Al-Ajn,Al-Ayin,Al-Ayn,Al-Aïn,Ehl'-A							
292932	Ajman	Ajman	Ajman,Al Ajman,QAJ,Ujman,‘jman,عجمان	41111.25	43504.55	P	PPLA	AE	02	
292953	Adh Dhayd	Adh Dhayd	Adh Dhaid,Adh Dhayd,Al Daid,Al-Dhayd,Dayd,Dhaid,Dhayd,Duhayd,Ihaid,aldhyd,الذيد,Ḍayd25.2							
292968	Abu Dhabi	Abu Dhabi	A-pu-that-pi,AEbu Saby,AUH,Aboe Dhabi,Abou Dabi,Abu Dabi,Abu Dabis,Abu Daby,Abu Daibi,Abu							

D3 gerando HTML

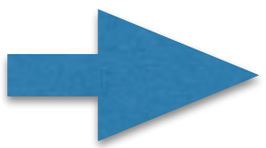
Shanghai: 22315474
Buenos Aires: 13076300
Mumbai: 12691836
Mexico City: 12294193
Beijing: 11716620
Karachi: 11624219
İstanbul: 11174257
Tianjin: 11090314
Guangzhou: 11071424
Delhi: 10927986
Moscow: 10381222
Shenzhen: 10358381
Dhaka: 10356500
Seoul: 10349312
São Paulo: 10021295
Wuhan: 9785388
Lagos: 9000000
Jakarta: 8540121
Tokyo: 8336599
New York City: 8175133
Dongguan: 8000000
Taipei: 7871900
Kinshasa: 7785965
Lima: 7737002
Cairo: 7734614
Bogotá: 7674366
London: 7556900
City of London: 7556900
Chongqing: 7457600

```
<div class="cities">Shanghai: 22315474</div>
<div class="cities">Buenos Aires: 13076300</div> == $0
<div class="cities">Mumbai: 12691836</div>
<div class="cities">Mexico City: 12294193</div>
<div class="cities">Beijing: 11716620</div>
<div class="cities">Karachi: 11624219</div>
<div class="cities">İstanbul: 11174257</div>
<div class="cities">Tianjin: 11090314</div>
<div class="cities">Guangzhou: 11071424</div>
<div class="cities">Delhi: 10927986</div>
<div class="cities">Moscow: 10381222</div>
<div class="cities">Shenzhen: 10358381</div>
<div class="cities">Dhaka: 10356500</div>
<div class="cities">Seoul: 10349312</div>
<div class="cities">São Paulo: 10021295</div>
<div class="cities">Wuhan: 9785388</div>
<div class="cities">Lagos: 9000000</div>
<div class="cities">Jakarta: 8540121</div>
<div class="cities">Tokyo: 8336599</div>
<div class="cities">New York City: 8175133</div>
<div class="cities">Dongguan: 8000000</div>
<div class="cities">Taipei: 7871900</div>
<div class="cities">Kinshasa: 7785965</div>
<div class="cities">Lima: 7737002</div>
<div class="cities">Cairo: 7734614</div>
<div class="cities">Bogotá: 7674366</div>
<div class="cities">London: 7556900</div>
<div class="cities">City of London: 7556900</div>
<div class="cities">Chongqing: 7457600</div>
<div class="cities">Chengdu: 7415590</div>
<div class="cities">Baghdad: 7216000</div>
<div class="cities">Nanjing: 7165292</div>
<div class="cities">Tehran: 7153309</div>
<div class="cities">Nanchong: 7150000</div>
<div class="cities">Hong Kong: 7012738</div>
<div class="cities">Xi'an: 6501190</div>
```

D3 gerando HTML

```
d3.tsv("cities15000.tsv",function(error,cidades) {

        cidades.sort(function(a,b) {
            return d3.descending(+a.population,+b.population);
        });

        console.log(incomingData)

        d3.select("body").selectAll("div.cities")
                .data(cidades)
                .enter()
                .append("div")
                .attr("class","cities")
                .html(function(d,i) {return d.name
                                    + ": " + d.population})

    });
```

Dados geram objetos



SVG

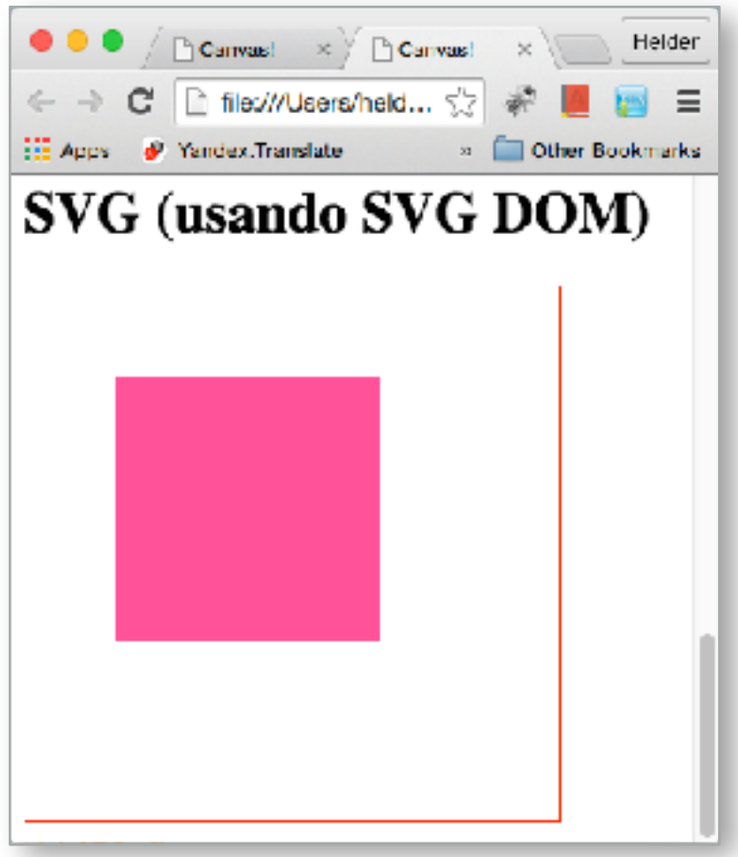
```
<html>
    <head>
    <style>
        svg { border: red dashed 1px;
              height: 200px; width: 200px;
        }
        circle {fill: red}
    </style>
    </head>
    <body>
        <h1>SVG embutido</h1>

        <svg viewBox="0 0 200 200">
            <circle r="50" cx="100" cy="100" fill="green" />
        </svg>
    </body>
</html>
```

comoembutir.html

SVG DOM

- Existe uma API de **Document Object Model** para SVG

```
<h1>SVG (usando SVG DOM)</h1>
<svg id="desenho2" height="300" width="300"></svg>

<script>
    // svg dom
    var svgns = "http://www.w3.org/2000/svg";
    var ansvg = document.getElementById("desenho2");
    var rect = document.createElementNS(svgns, "rect");
    rect.setAttribute("x",50);
    rect.setAttribute("y",50);
    rect.setAttribute("height",150);
    rect.setAttribute("width",150);
    rect.setAttribute("fill","#ff459a");
    ansvg.appendChild(rect);
</script>
```

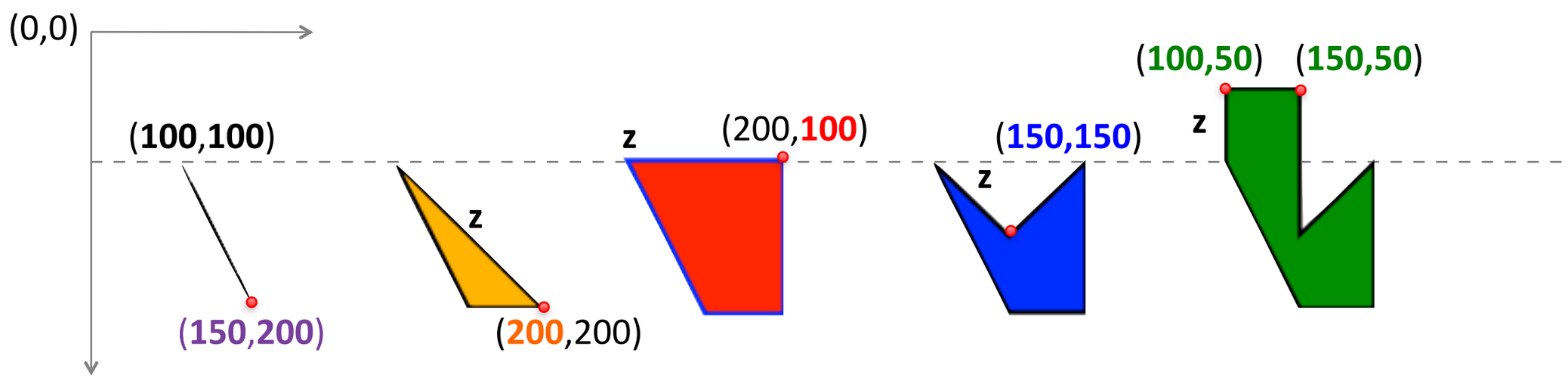


canvas_svg1.html

Caminhos

- **H** ou **h** + coordenadas x linhas retas horizontais
- **V** ou **v** + coordenadas y linhas retas verticais
- **L** ou **l** + pares de coords x,y linhas retas em qq direção

```
1. M100,100 L150,200 z
2. M100,100 L150,200 h50 z
3. M100,100 L150,200 h50 v-100 z
4. M100,100 L150,200 h50 v-100 l-50,50 z
5. M100,100 L150,200 h50 v-100 l-50,50 L150,50 100,50 z
```



Componentes gráficos

<rect> – Retângulo

<circle> – Círculo

<ellipse> – Elipse

<line> – Linha reta

<polyline> – Linha com múltiplos segmentos

<polygon> – Polígono

<path> - Caminho arbitrário (curvas, linhas, etc.)

<image> - Imagem bitmap

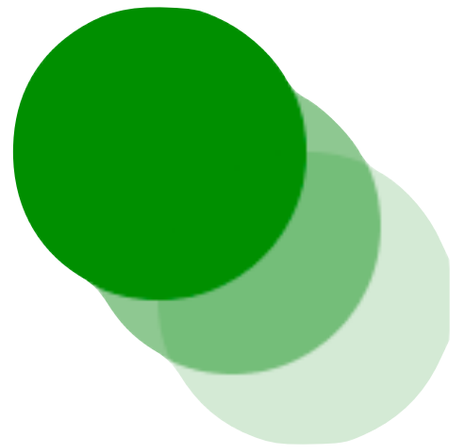
<text> - Texto

Preenchimento

- Use **fill** (atributo ou CSS) para cor de preenchimento
  - `<rect ... fill="rgb(255,255,0%)" />`
  - `<rect ... style="fill: rgb(100%,100%,0%)" />`

fill.svg

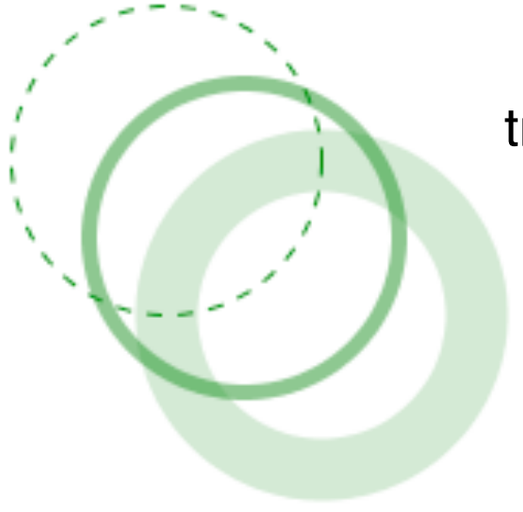
- Use **fill-opacity** para o componente alfa (transparência)
  - Varia de **0** (transparente) a **1** (opaco).



```
<svg xmlns="http://www.w3.org/2000/svg">
    <circle r="50" cx="100" cy="100" fill="green"/>
    <circle r="50" cx="125" cy="125" fill="#008000"
                                     fill-opacity="0.5"/>
    <circle r="50" cx="150" cy="150" fill="#080"
                                     fill-opacity="0.2"/>
</svg>
```

Traço

- **stroke**: cor do traço
- **stroke-width**: espessura
- **stroke-opacity**: transparência (alfa)
- **stroke-dasharray**
  - Lista de valores para tracejado (seqüência de traços e vazios)

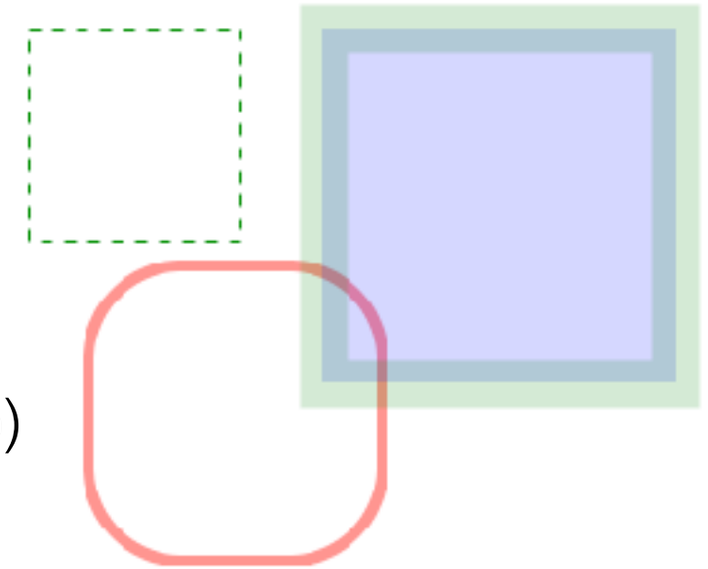


traco.svg

```
<?xml version="1.0" encoding="UTF-8"?>
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 300 300">
    <circle r="50" cx="100" cy="100" stroke="green" fill-opacity="0"
        stroke-width="1"  stroke-dasharray="5 5"/>
    <circle r="50" cx="125" cy="125" stroke="green" fill-opacity="0"
        stroke-width="5"  stroke-opacity="0.5"/>
    <circle r="50" cx="150" cy="150" stroke="green" fill-opacity="0"
        stroke-width="20" stroke-opacity="0.2"/>
</svg>
```

Retângulo <rect>

rect.svg



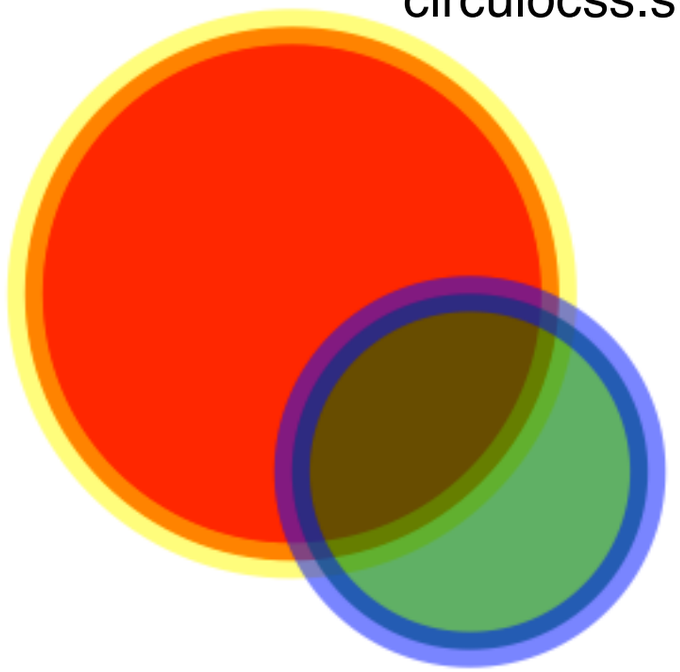
- Atributos
  - **x**, **y** – coords de posição (default: 0)
  - **width**, **height** – largura, altura (obrigatório)
  - **rx**, **ry** – raios horizontal e vertical para cantos arredondados (opcional)

```
<rect x="25" y="50" width="90" height="90"
      stroke="green" fill-opacity="0" stroke-width="1"
      stroke-dasharray="5 5"/>
<rect x="50" y="150" width="125" height="125" rx="40" ry="40"
      stroke="red" fill-opacity="0" stroke-width="5"
      stroke-opacity="0.5" />
<rect x="150" y="50" width="150" height="150"
      stroke="green" fill="blue" fill-opacity="0.2"
      stroke-width="20" stroke-opacity="0.2" />
```

Círculo <circle>

circulocss.svg

- Atributos
  - **cx**, **cy** – coords. do centro (default: 0)
  - **r** – raio (obrigatório)



```css
circle {
  stroke-width: 10;
  stroke-opacity: 0.5;
}
#c1 {
  fill: green;
  stroke: blue;
  fill-opacity: 0.6;
}
#c2 {
  fill: #f00;
  stroke: yellow;
}
```

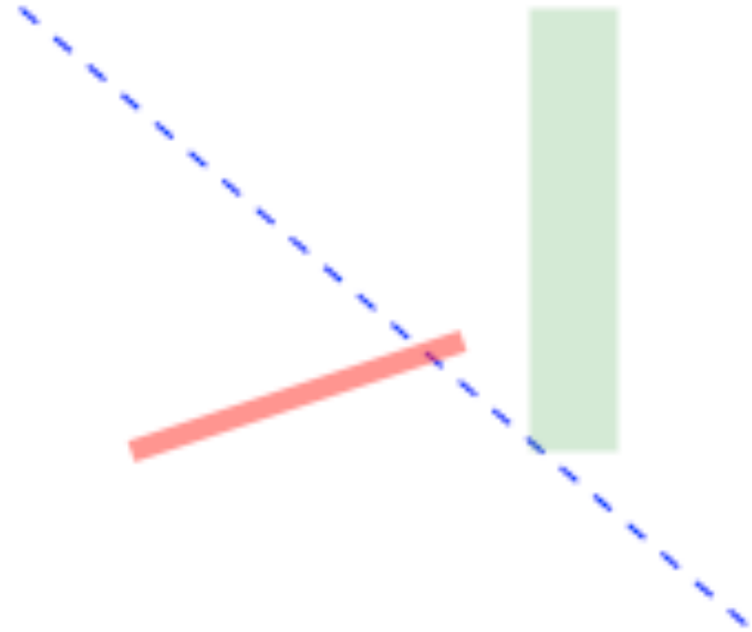
circulos.css

```svg
<?xml-stylesheet type="text/css"
                 href="circulos.css"?>

<svg xmlns="http://www.w3.org/2000/svg"
     viewBox="0 0 300 300">

   <circle r="75" cx="100" cy="100" id="c2"/>
   <circle r="50" cx="150" cy="150" id="c1"/>

</svg>
```

Linha simples <line>

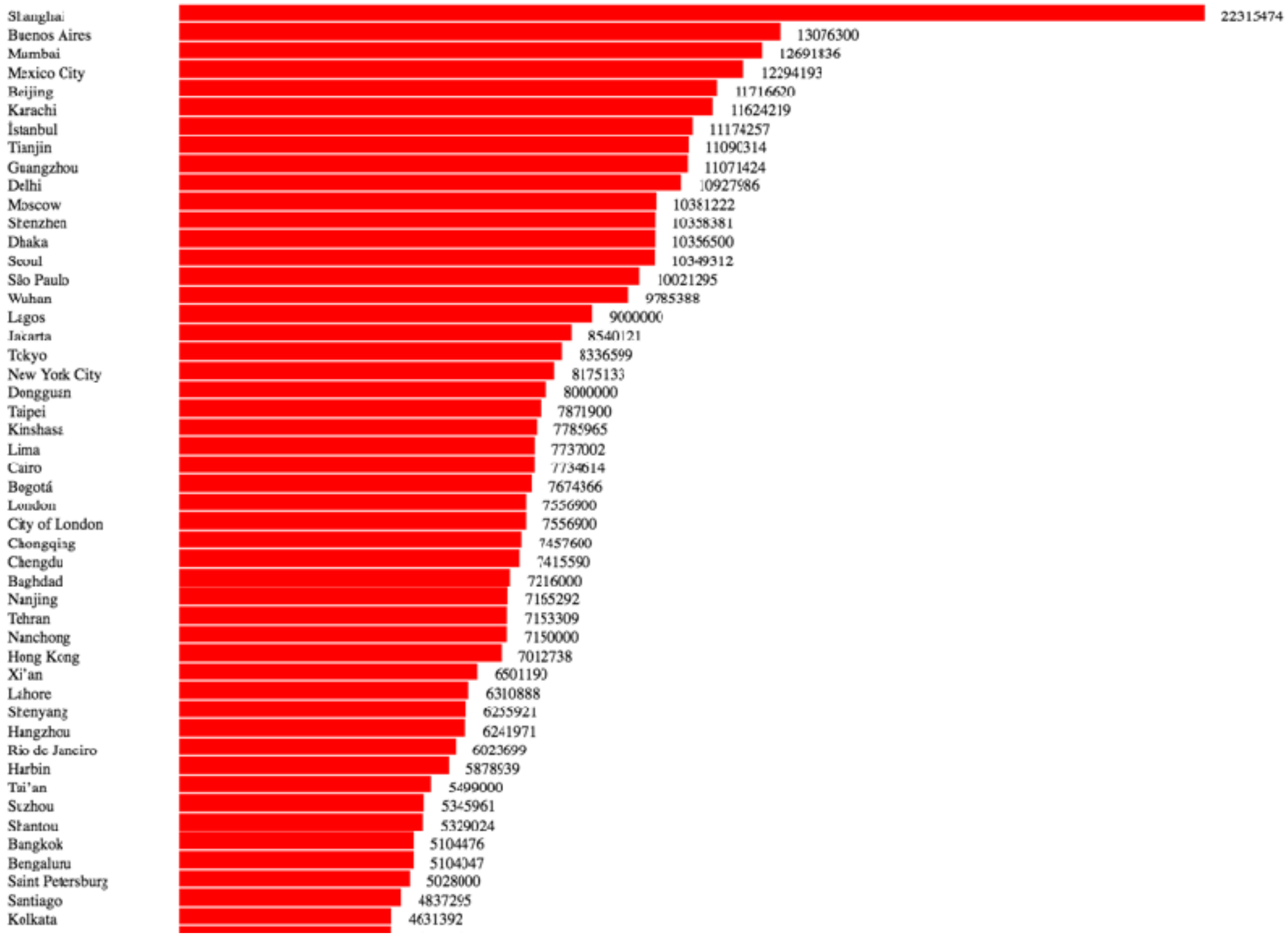
line.svg

- Atributos (default: 0)
  - **x1**, **y1** – coords do primeiro ponto
  - **x2**, **y2** – coords do segundo ponto



```
<line x1="25" y1="50" x2="190" y2="190"
   stroke="blue" stroke-width="1" stroke-dasharray="5 5"/>

<line x1="50" y1="150" x2="125" y2="125"
   stroke="red" stroke-width="5" stroke-opacity="0.5"/>

<line x1="150" y1="50" x2="150" y2="150"
   stroke="green" stroke-width="20" stroke-opacity="0.2" />
```

D3 gerando SVG

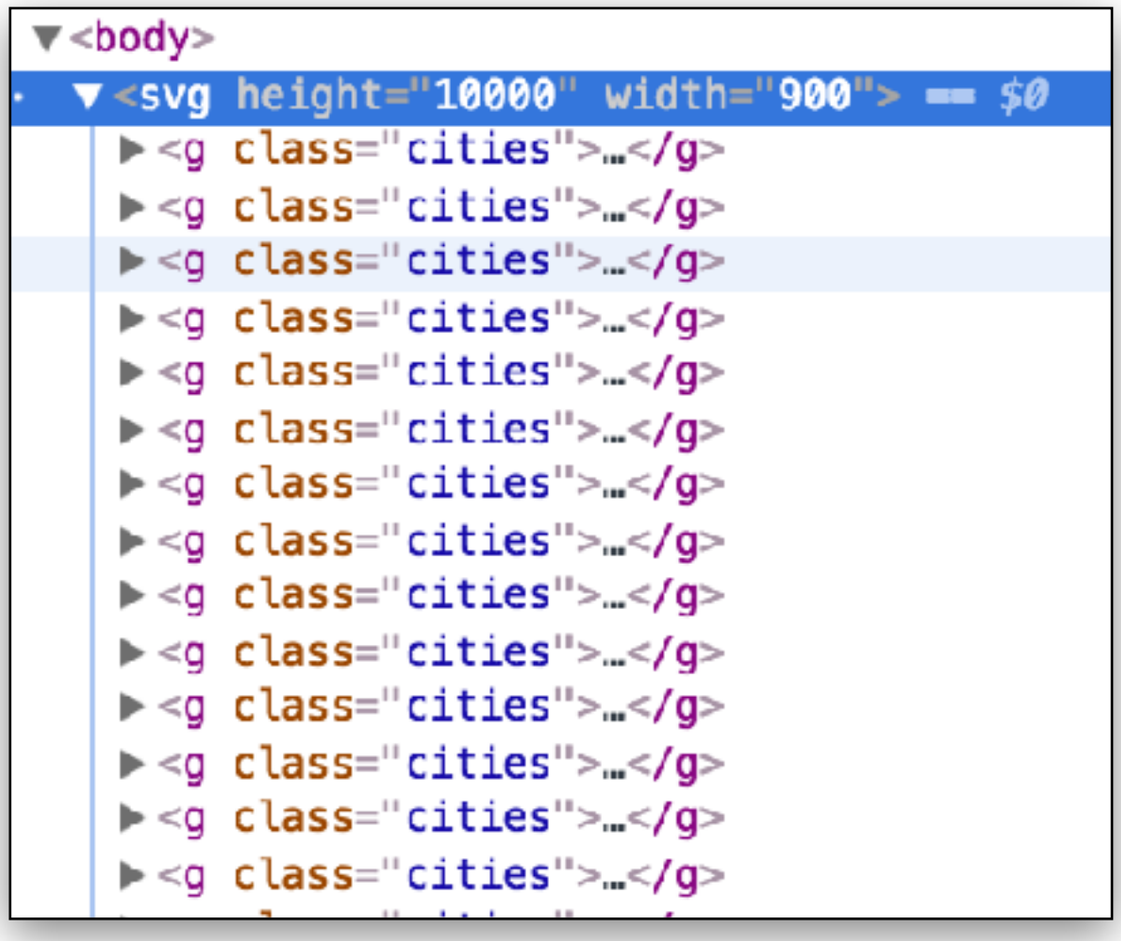


Exemplo D3 com SVG

```
<svg height="10000" width="900"></svg>

<script>
    var width = 900;

    d3.tsv("cities15000.tsv",function(error,data) {

        data.sort(function(a,b) {
            return d3.descending(+a.population,+b.population);
        })
        console.log(data);

        var bar_scale = d3.scale.linear()
                .range([0,width - 300])
                .domain([0, d3.max(data, function(d) {
                    return +d.population;
                })])
...

    });

</script>
```

Exemplo D3 com SVG

```
d3.tsv("cities15000.tsv",function(error,data) {

...

    var bar = d3.select("svg").selectAll("g.cities")
            .data(data)
            .enter()
            .append("g")
            .attr("class","cities");


});
```



Exemplo D3 com SVG

```
d3.tsv("cities15000.tsv",function(error,data) {

...

        var bar = d3.select("svg").selectAll("g.cities")
                .data(data)
                .enter()
                .append("g")
                .attr("class","cities");

            bar.append("rect")
                .attr("height", "10")
                .attr("x", 100)
                .attr("y", function(d, i) {
                    return i*11 + 50;
                })
                .attr("fill", "red")
                .attr("width", function(d) {
                    return bar_scale(d.population);
                });

    });
```

Exemplo D3 com SVG

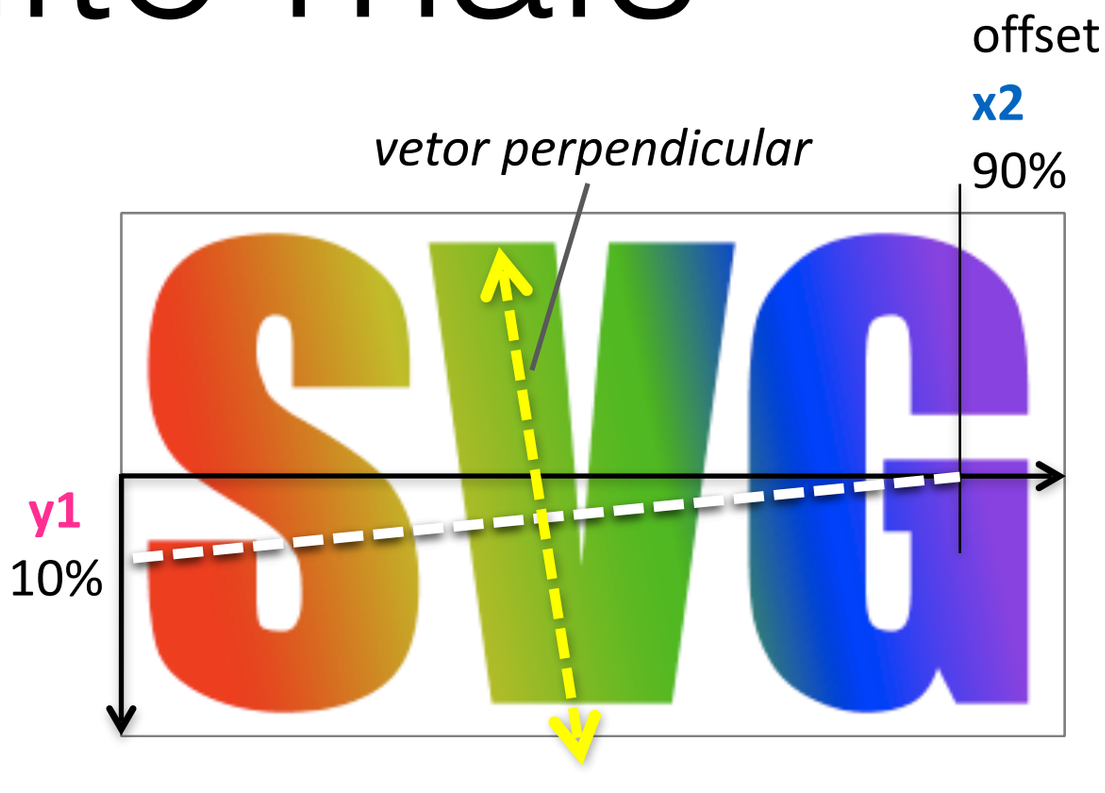
```
d3.tsv("cities15000.tsv",function(error,data) {

    ...

    bar.append("text")
            .text(function(d) { return d.name})
            .attr("x", 0)
            .attr("y", function(d, i) {
                return i*11 + 60;
            })
            .style("font-size", 9);

        bar.append("text")
            .text(function(d) { return d.population})
            .attr("x", function(d) {
                return bar_scale(d.population) + 10 + 100;
            })
            .attr("y", function(d, i) {
                return i*11 + 60;
            })
            .style("font-size", 9);

});
```

Exemplo D3 com SVG

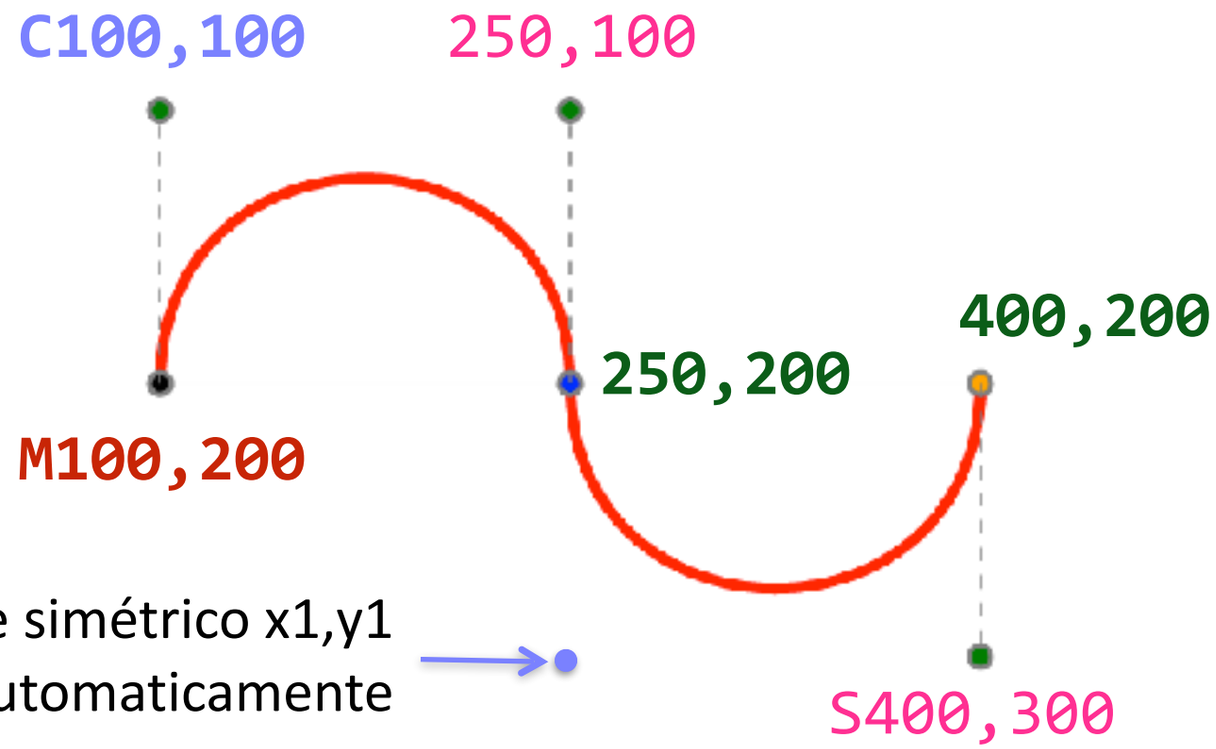
```
▼<svg height="10000" width="900">
  ▼<g class="cities">
      <rect height="10" x="100" y="50" fill="red" width="600"></rect>
      <text x="0" y="60" style="font-size: 9px;">Shanghai</text>
      <text x="710" y="60" style="font-size: 9px;">22315474</text>
    </g>
  ▼<g class="cities">
      <rect height="10" x="100" y="61" fill="red" width="351.58473443136364"></rect>
      <text x="0" y="71" style="font-size: 9px;">Buenos Aires</text>
      <text x="461.58473443136364" y="71" style="font-size: 9px;">13076300</text>
    </g>
  ▼<g class="cities">
      <rect height="10" x="100" y="72" fill="red" width="341.2475845236359"></rect>
      <text x="0" y="82" style="font-size: 9px;">Mumbai</text>
      <text x="451.2475845236359" y="82" style="font-size: 9px;">12691836</text>
    </g>
  ▼<g class="cities">
      <rect height="10" x="100" y="83" fill="red" width="330.55608856885584"></rect>
      <text x="0" y="93" style="font-size: 9px;">Mexico City</text>
      <text x="440.55608856885584" y="93" style="font-size: 9px;">12294193</text>
    </g>
  ▼<g class="cities"> == $0
      <rect height="10" x="100" y="94" fill="red" width="315.02678365693686"></rect>
      <text x="0" y="104" style="font-size: 9px;">Beijing</text>
      <text x="425.02678365693686" y="104" style="font-size: 9px;">11716620</text>
    </g>
  ▶<g class="cities">…</g>
  ▶<g class="cities">…</g>
```

SVG é muito mais

- Figuras, caminhos, bezier, filtros, gradientes, animações, CSS, DOM




```
<path
 d="M100,200
    C100,100 250,100 250,200
    S400,300 400,200" ... />
```



Por que D3.js e nao **xyz.js**

D3 não é uma biblioteca gráfica!

- D3 fornece toda a **infraestrutura** para que dados gerem visualizações com D3, sem reinventar a roda

- Aproveita recursos do JavaScript (e adiciona outros, para facilitar manipulação de dados)

- Gráficos usam **padrões abertos**: HTML e SVG

- Abrangente biblioteca para dados geográficos (converte transparentemente coordenadas em caminhos de path para SVG e canvas)

- Suporta plug-ins

Uso de memória com D3.js

- Cada dado = 1 objeto (SVG) do DOM
- Funciona bem para centenas e até **milhares de objetos**, nas plataformas atuais
- Mas o que acontece com **milhões** de objetos?

GeoJSON (geojson.org)

- Principal formato vetorial para mapas usado na Web
- Padrão JSON (JavaScript Object Notation) para GIS
- GeoJSON interativo: http://geojson.io/

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

Várias ferramentas convertem de SHP para GeoJSON

GeoJSON


```
{"type":"FeatureCollection","features":[
{"type":"Feature","id":"AFG","properties":{"name":"Afghanistan"},"geometry":{"type":"Polygon","coordinates":
[[[61.210817,35.650072],[62.230651,35.270664],[62.984662,35.404041],[63.193538,35.857166],[63.982896,36.007957],
[64.546479,36.312073],[64.746105,37.111818],[65.588948,37.305217],[65.745631,37.661164],[66.217385,37.39379],
[66.518607,37.362784],[67.075782,37.356144],[67.83,37.144994],[68.135562,37.023115],[68.859446,37.344336],
[69.196273,37.151144],[69.518785,37.608997],[70.116578,37.588223],[70.270574,37.735165],[70.376304,38.138396],
[70.806821,38.486282],[71.348131,38.258905],[71.239404,37.953265],[71.541918,37.905774],[71.448693,37.065645],
[71.844638,36.738171],[72.193041,36.948288],[72.63689,37.047558],[73.260056,37.495257],[73.948696,37.421566],
[74.980002,37.41999],[75.158028,37.133031],[74.575893,37.020841],[74.067552,36.836176],[72.920025,36.720007],
[71.846292,36.509942],[71.262348,36.074388],[71.498768,35.650563],[71.613076,35.153203],[71.115019,34.733126],
[71.156773,34.348911],[70.881803,33.988856],[69.930543,34.02012],[70.323594,33.358533],[69.687147,33.105499],
[69.262522,32.501944],[69.317764,31.901412],[68.926677,31.620189],[68.556932,31.71331],[67.792689,31.58293],
[67.683394,31.303154],[66.938891,31.304911],[66.381458,30.738899],[66.346473,29.887943],[65.046862,29.472181],
[64.350419,29.560031],[64.148002,29.340819],[63.550261,29.468331],[62.549857,29.318572],[60.874248,29.829239],
[61.781222,30.73585],[61.699314,31.379506],[60.941945,31.548075],[60.863655,32.18292],[60.536078,32.981269],
[60.9637,33.528832],[60.52843,33.676446],[60.803193,34.404102],[61.210817,35.650072]]]}},
{"type":"Feature","id":"AGO","properties":{"name":"Angola"},"geometry":{"type":"MultiPolygon","coordinates":
[[[[16.326528,-5.87747],[16.57318,-6.622645],[16.860191,-7.222298],[17.089996,-7.545689],[17.47297,-8.068551],
[18.134222,-7.987678],[18.464176,-7.847014],[19.016752,-7.988246],[19.166613,-7.738184],[19.417502,-7.155429],
[20.037723,-7.116361],[20.091622,-6.94309],[20.601823,-6.939318],[20.514748,-7.299606],[21.728111,-7.290872],
[21.746456,-7.920085],[21.949131,-8.305901],[21.801801,-8.908707],[21.875182,-9.523708],[22.208753,-9.894796],
[22.155268,-11.084801],[22.402798,-10.993075],[22.837345,-11.017622],[23.456791,-10.867863],[23.912215,-10.926826],
[24.017894,-11.237298],[23.904154,-11.722282],[24.079905,-12.191297],[23.930922,-12.565848],[24.016137,-12.911046],
[21.933886,-12.898437],[21.887843,-16.08031],[22.562478,-16.898451],[23.215048,-17.523116],[21.377176,-17.930636],
[18.956187,-17.789095],[18.263309,-17.309951],[14.209707,-17.353101],[14.058501,-17.423381],[13.462362,-16.971212],
[12.814081,-16.941343],[12.215461,-17.111668],[11.734199,-17.301889],[11.640096,-16.673142],[11.778537,-15.793816],
[12.123581,-14.878316],[12.175619,-14.449144],[12.500095,-13.5477],[12.738479,-13.137906],[13.312914,-12.48363],
[13.633721,-12.038645],[13.738728,-11.297863],[13.686379,-10.731076],[13.387328,-10.373578],[13.120988,-9.766897],
[12.87537,-9.166934],[12.929061,-8.959091],[13.236433,-8.562629],[12.93304,-7.596539],[12.728298,-6.927122],
[12.227347,-6.294448],[12.322432,-6.100092],[12.735171,-5.965682],[13.024869,-5.984389],[13.375597,-5.864241],
[16.326528,-5.87747]]],[[[12.436688,-5.684304],[12.182337,-5.789931],[11.914963,-5.037987],[12.318608,-4.60623],
[12.62076,-4.438023],[12.995517,-4.781103],[12.631612,-4.991271],[12.468004,-5.248362],[12.436688,-5.684304]]]]}},
{"type":"Feature","id":"ALB","properties":{"name":"Albania"},"geometry":{"type":"Polygon","coordinates":
[[[20.590247,41.855404],[20.463175,41.515089],[20.605182,41.086226],[21.02004,40.842727],[20.99999,40.580004],
[20.674997,40.435],[20.615,40.110007],[20.150016,39.624998],[19.98,39.694993],[19.960002,39.915006],[19.406082,40.250773],
[19.319059,40.72723],[19.40355,41.409566],[19.540027,41.719986],[19.371769,41.877548],[19.304486,42.195745],
[19.738051,42.688247],[19.801613,42.500093],[20.0707,42.58863],[20.283755,42.32026],[20.52295,42.21787],
[20.590247,41.855404]]]}},
...
```

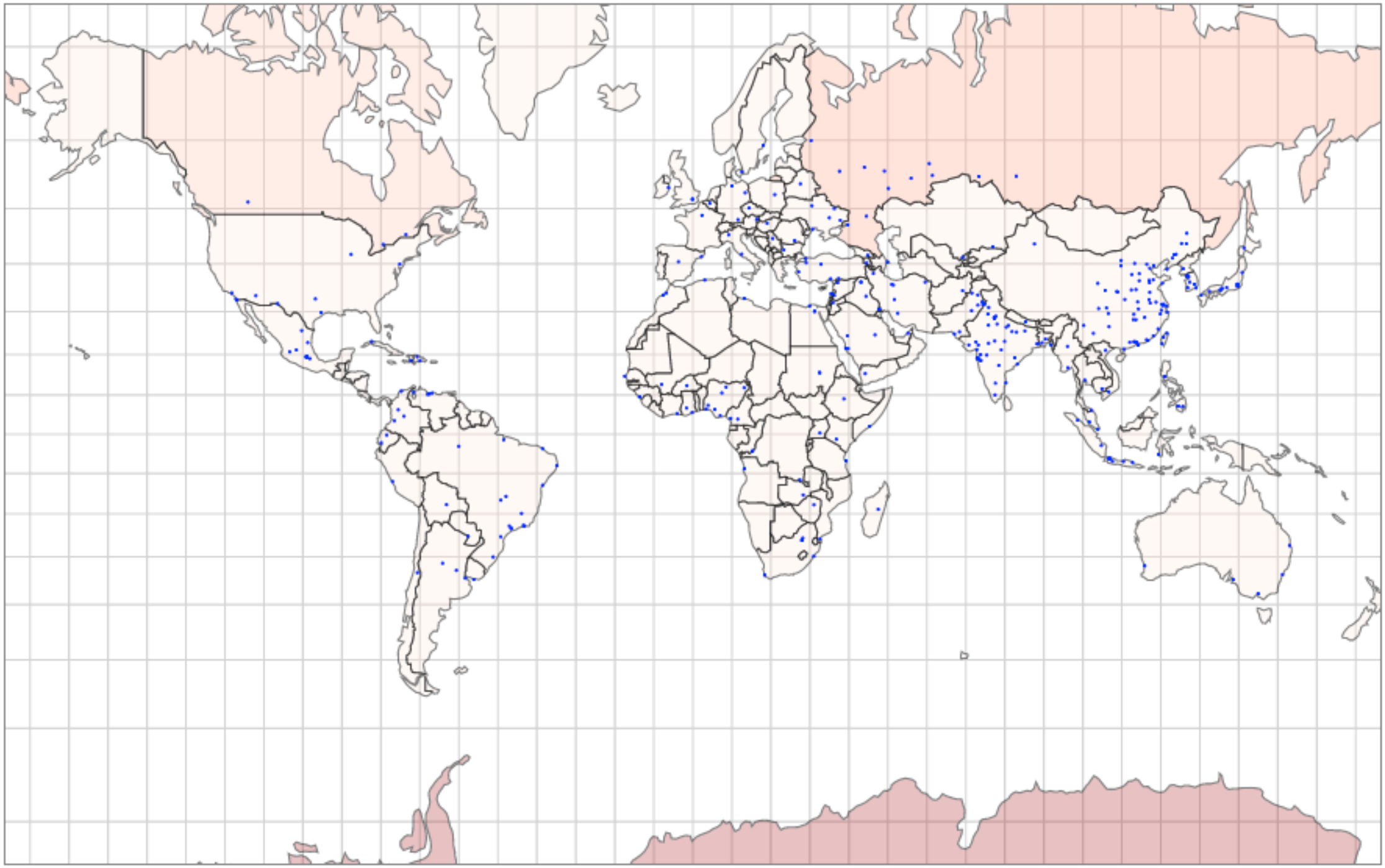
TSV + GeoJSON

```
queue()
        .defer(d3.json, "world.geojson")
        .defer(d3.tsv, "cities5000.tsv")
        .await(function(error, file1, file2) {


    width = 800;
    height = 500;
    projection = d3.geo.mercator()
                    .scale(130)
                    .translate([width / 2, height / 2]);
    geoPath = d3.geo.path().projection(projection);

    var mapZoom = d3.behavior.zoom()
        .translate(projection.translate())
        .scale(projection.scale()).on("zoom", function(d) {...});

    d3.select("svg").call(mapZoom);


});
```

TSV + GeoJSON

```
...

countryColor = d3.scale.quantize()
        .domain(featureSize)
        .range(colorbrewer.Reds[7]);

d3.select("svg").selectAll("path.countries").data(countries.features)
        .enter()
        .append("path")
        .attr("d", geoPath)
        .attr("class", "countries")
        .style("fill", function(d) {
            return countryColor(geoPath.area(d))
        })
        .style("stroke-width", 1)
        .style("stroke", "black")
        .style("opacity", .5).
```

TSV + GeoJSON

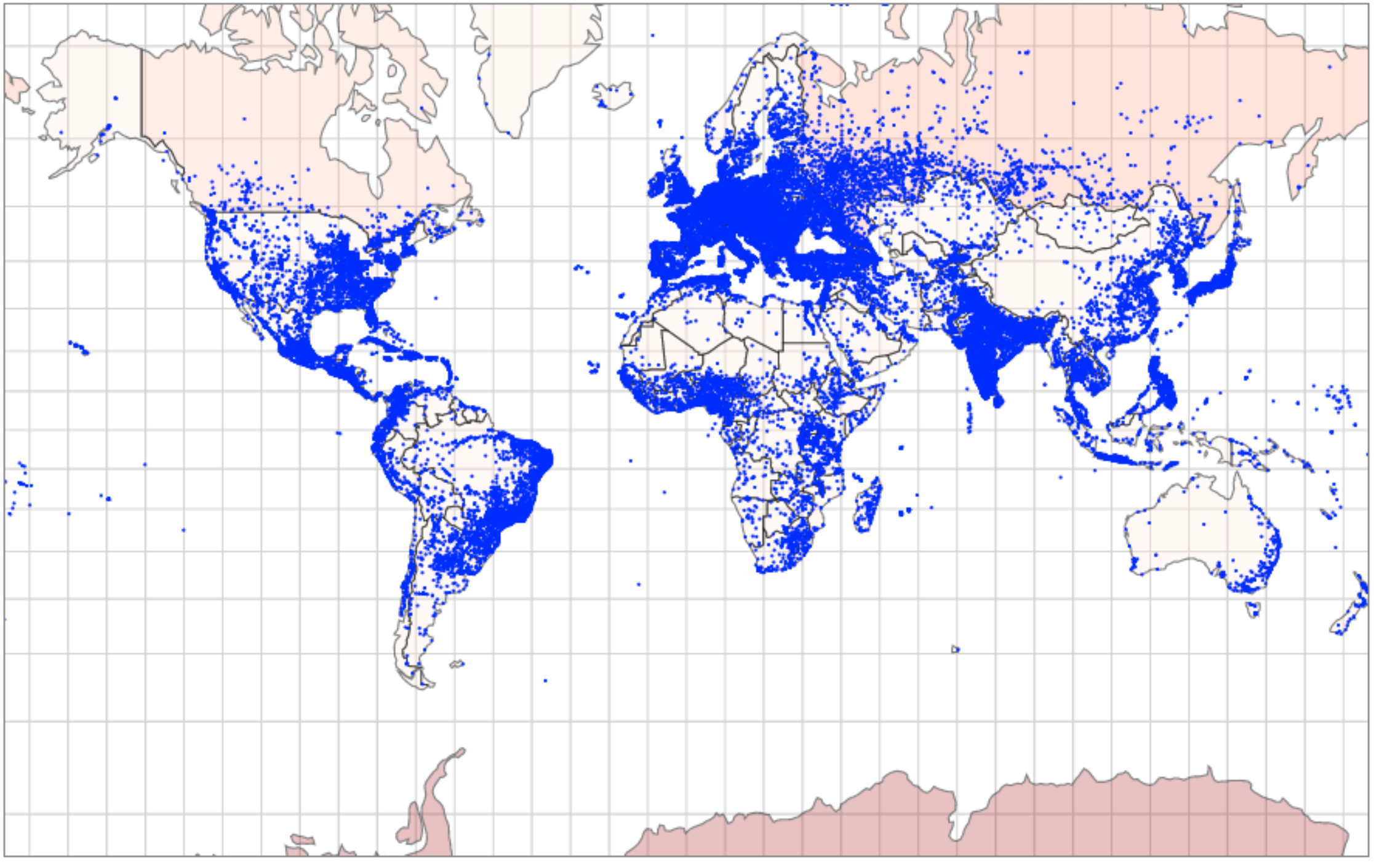


Projetando as cidades

```
...

var cities = d3.select("svg").selectAll("circle").data(visibleCities)
cities.enter()
        .append("circle")
        .attr("class", "cities")
        .style("fill", "blue")
        .style("stroke-width", 1)
        .attr("r", 1)
        .attr("cx", function(d) {
            return projection([d.longitude,d.latitude])[0]
})
.attr("cy", function(d) {
            return projection([d.longitude,d.latitude])[1]
});
```

```
<circle class="cities" r="1" cx="665.8209597536315" cy="167.3900588076179" style=
"fill: blue; stroke-width: 1; display: none;"></circle>
<circle class="cities" r="1" cx="658.8065460670487" cy="172.96700918502427" style=
"fill: blue; stroke-width: 1; display: none;"></circle>
<circle class="cities" r="1" cx="646.6383985014708" cy="166.9581524860935" style=
"fill: blue; stroke-width: 1; display: none;"></circle>
<circle class="cities" r="1" cx="647.1511762357067" cy="167.17636161045198" style=
"fill: blue; stroke-width: 1; display: none;"></circle>
<circle class="cities" r="1" cx="667.9192643935492" cy="192.685813730218" style=
"fill: blue; stroke-width: 1; display: none;"></circle>
<circle class="cities" r="1" cx="672.9261078720378" cy="174.4465071885578" style=
"fill: blue; stroke-width: 1; display: none;"></circle>
<circle class="cities" r="1" cx="659.2628501826778" cy="177.0573801383551" style=
"fill: blue; stroke-width: 1; display: none;"></circle>
<circle class="cities" r="1" cx="639.9218496863502" cy="166.29293921983162" style=
"fill: blue; stroke-width: 1; display: none;"></circle>
<circle class="cities" r="1" cx="665.8654307429723" cy="153.34665033240975" style=
```

Mapa com cidades



Problemas com Big Data

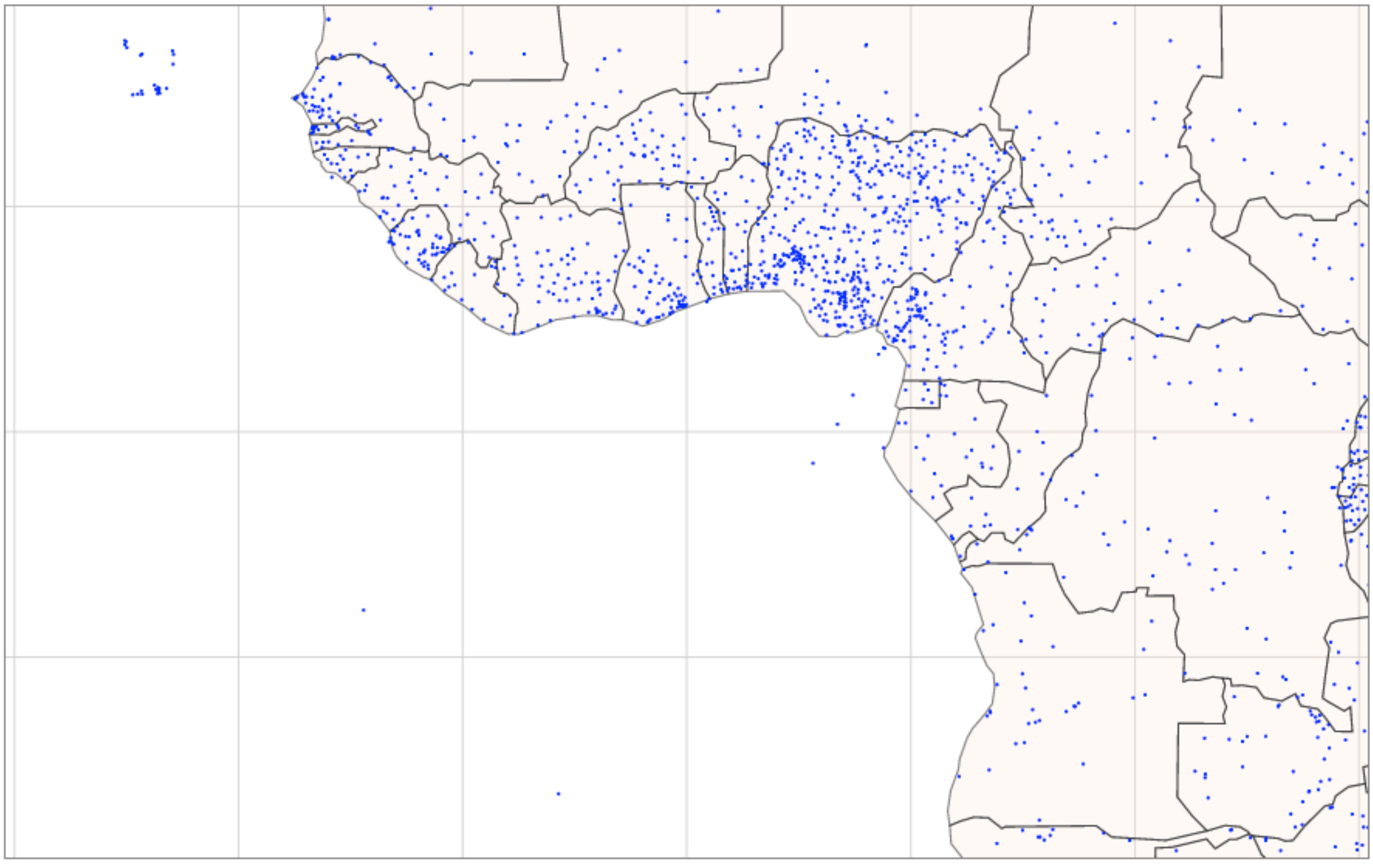
- Como **visualizar** muitos dados ao mesmo tempo?
  - http://syntagmatic.github.io/parallel-coordinates/examples/slickgrid.html

- Como administrar os **recursos** (memória, processador) para uma visualização eficiente

Reduzir os dados

- Você precisa mesmo **mostrar** todos esses pontos?
  - Eles são mesmo todos visíveis, distinguíveis?
  - É importante que todos estejam na tela?
    - É importante que todos sejam objetos distintos?
- O **nível de abstração x escala** é adequado? Experimente usar abstrair mais (mostrar menos detalhes)
- A **estratégia de exibição** é adequada? Experimente outros tipos de gráfico ou projeções

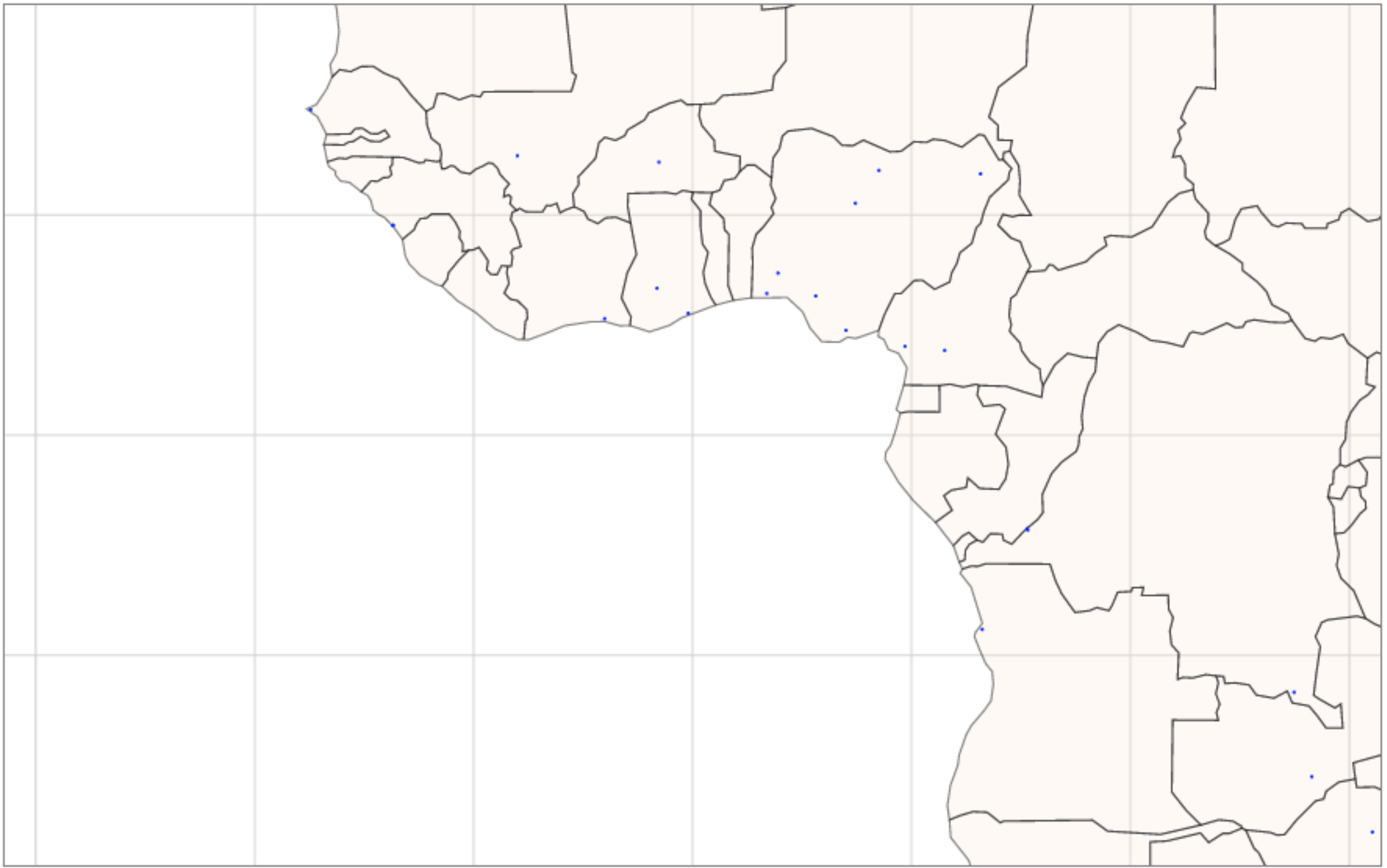
Zoom



Zoom com filtro

```
if(mapZoom.scale() > 1000) {
    visibleCities = allCities;
    cities.exit().remove();
    cities.data(allCities)
            .enter()
            .append("circle").attr("class", "cities")
} else {
    if(visibleCities === allCities) {
        visibleCities = allCities.filter(function(d) {
            return parseFloat(d.population) > 1000000;
        });
        cities.exit().remove();
        cities.data(allCities)
                .enter()
                .append("circle").attr("class", "cities")
    }

}
```

Zoom com filtro



Enviar dados sob demanda

- Back-end: servidor envia dados filtrados sob demanda (ajax)
- Queries
- Cliente recebe apenas dados que precisa para o escopo e abrangência da visualização

Criar menos objetos

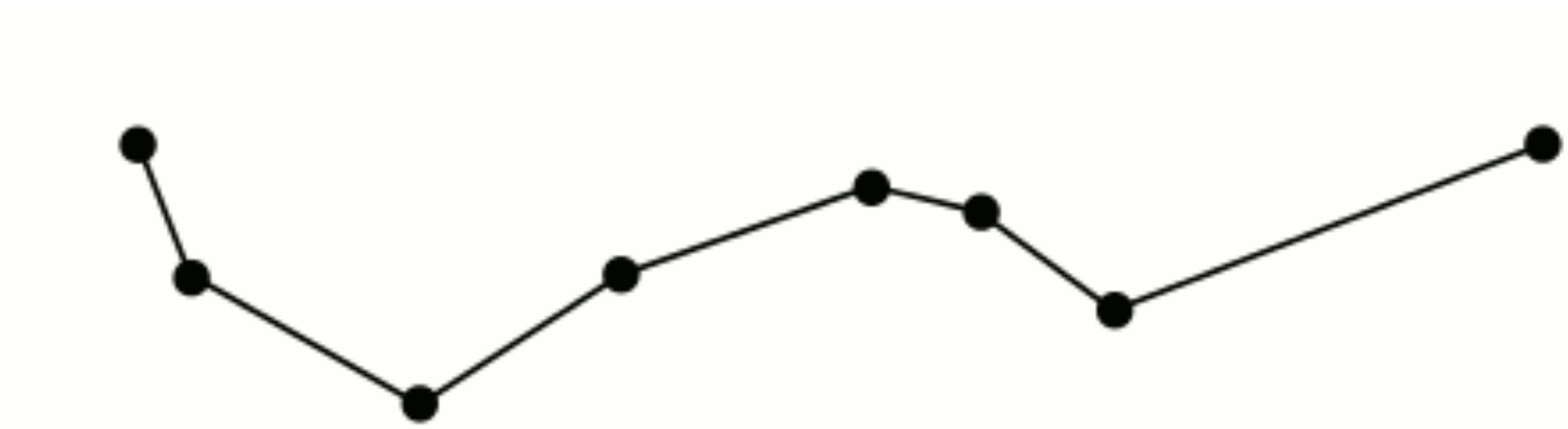
- Às vezes o problema não é o tamanho dos dados mas a **quantidade de objetos** criados no DOM

- Solução: reduzir a quantidade de objetos
  - Substituir grupos de dados por único objeto que representa o grupo
  - Copiar objetos para HTML Canvas
  - Algoritmos para eliminar redundâncias e reduzir pontos
  - Eliminar objetos não visíveis no zoom ou brush

Exemplo: redução de pontos

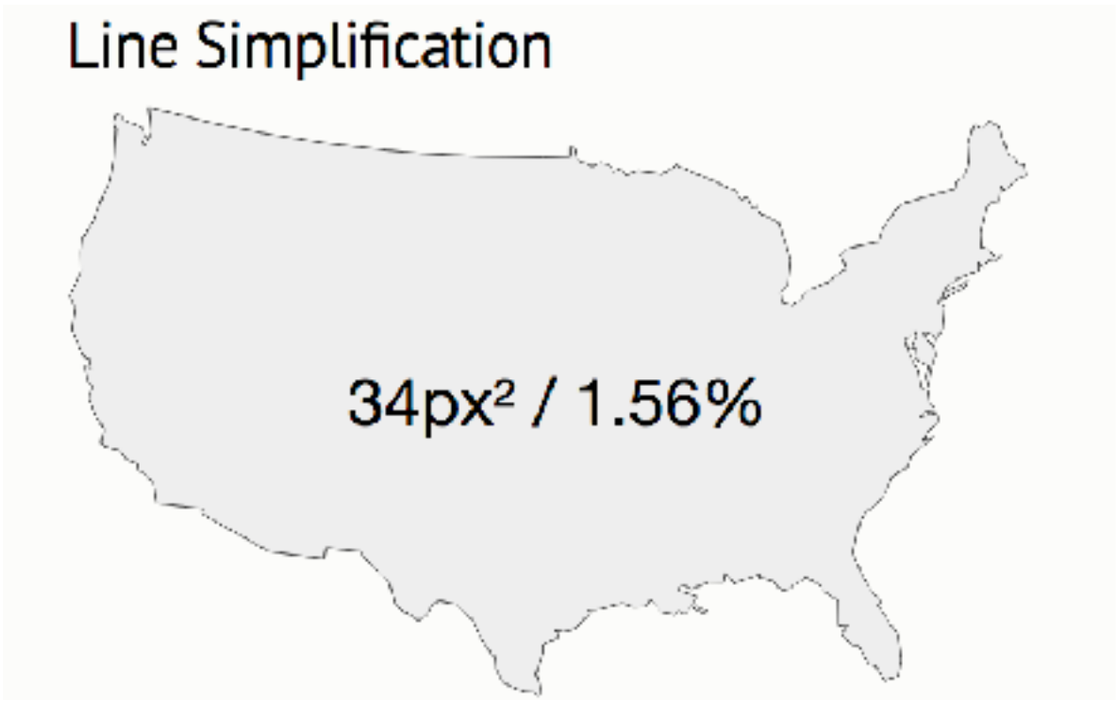
- Dependendo da escala, menos pontos são necessários para representar um mapa ou caminho

- Algoritmo Douglas-Peucker



Simplificação de linhas

- Algoritmo de Visvalingam (por M. Bostock)
- https://bost.ocks.org/mike/simplify/



TopoJSON

- Reduz em 90% o tamanho de um arquivo GeoJSON ao eliminar redundâncias

- Guarda topologias (facilita encontrar adjacências e aplicar algoritmos que dependem de adjacências)

- https://github.com/mbostock/topojson/wiki

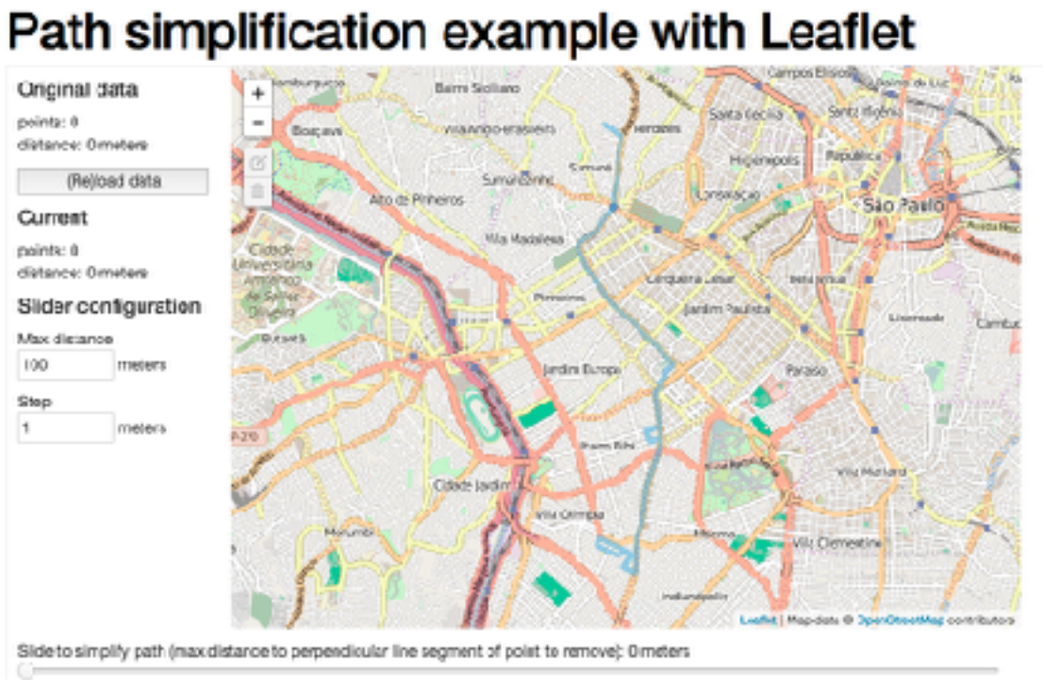
- Exemplos: https://bl.ocks.org/mbostock/4122298

- Colorindo um mapa com número limitado de cores: http://bl.ocks.org/rveciana/f46df2272b289a9ce4e7

Redução de caminho

- Dependendo da escala, não são necessários tantos pontos (usando Douglas-Peucker)

- https://bl.ocks.org/helderdarocha/47fff819307ef8488607007ebb4f8b92



HTML5 Canvas

- D3 permite desenhar no contexto de um Canvas
- Canvas é um **único** objeto: menos memória
- Ideal quando não é necessário acessar objetos individualmente (durante zoom)

Canvas

- Desenhar objetos ou pintar no canvas

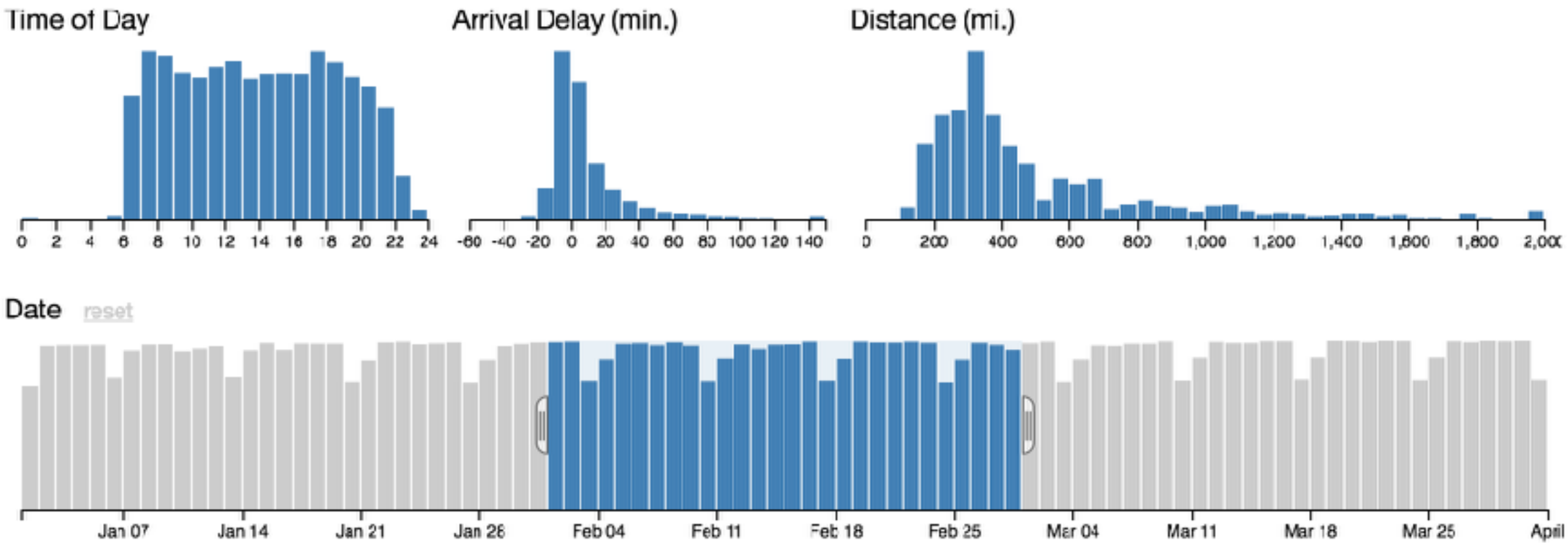
```
function cidadesObjetos() {
    d3.selectAll("canvas").style("display", "none");
    redesenha(addCities);
}
function cidadesCanvas() {
    var context = d3.select("canvas").node().getContext("2d");
    context.clearRect(0, 0, 500, 800);
    context.fillStyle = "blue";
    for (var x in allCities) {
        context.beginPath();
        context.arc(projection([x.longitude, x.latitude])[1],
                    projection([x.longitude, x.latitude])[0],
                    1, 0, Math.PI * 2, false);
        context.fill();
    }
}
```

Zoom, contexto e detalhe

- Dados devem ser baixados ou mostrados de acordo com a escala e nível de abstração
- Brush/zoom pode
  - Carregar dados sob demanda (ajax, queries)
  - Aplicar algoritmos de simplificação
  - Converter objetos em canvas, etc.

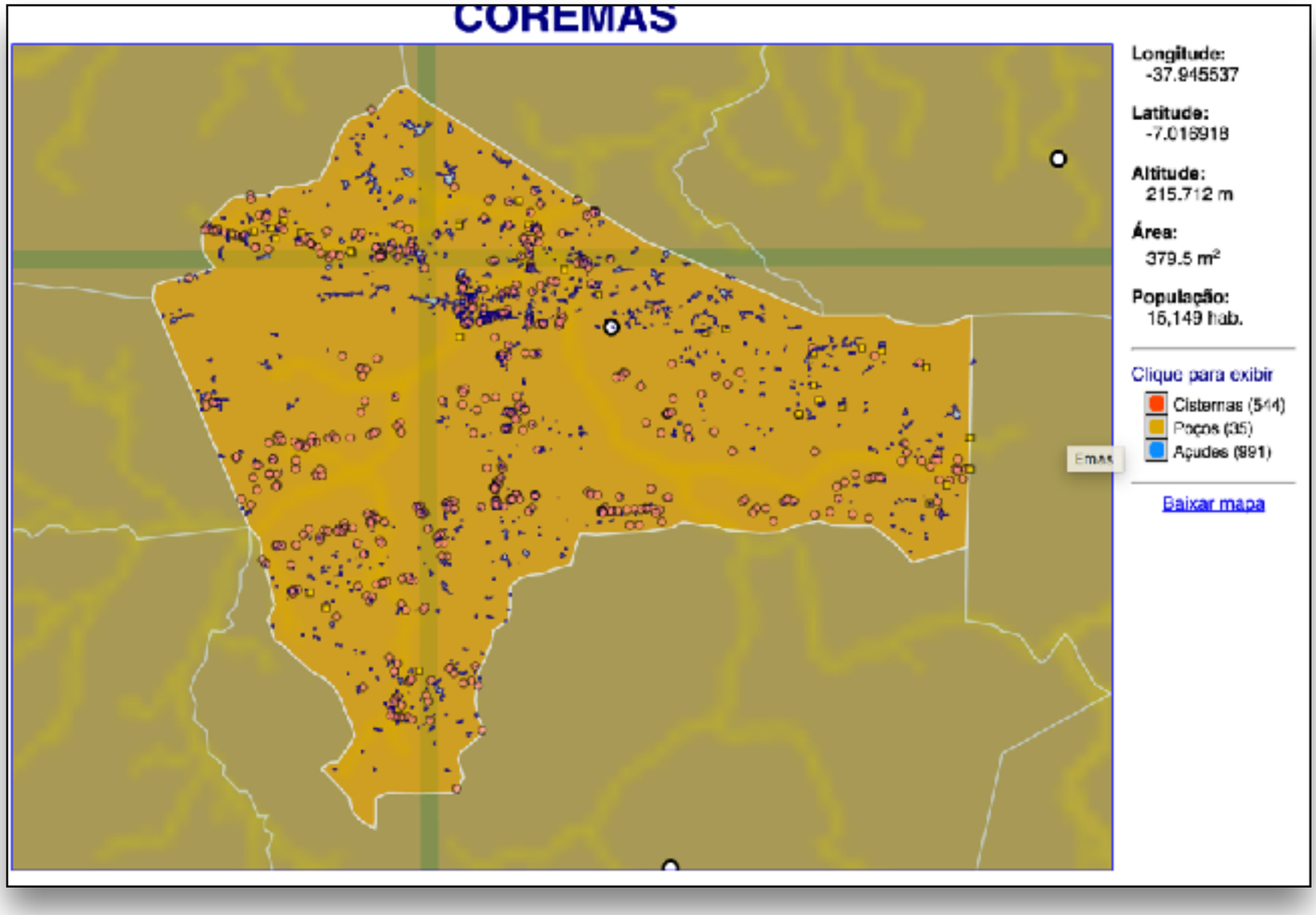
D3: crossfilter

- Crossfilter http://square.github.io/crossfilter/



Objetos sob demanda

- Ex: http://www.geociencias.ufpb.br/leppan/gepat/atlas/



Conclusões

- D3.js é uma biblioteca JavaScript que associa dados a componentes visuais usando **tecnologias abertas** (HTML, SVG).
- Não é uma biblioteca gráfica, mas oferece várias ferramentas para adaptar os dados para a visualização - ideal para filtrar e otimizar dados
- Estratégias para possibilitar a visualização de Big Data com D3 incluem:
  - Transferência de dados **sob demanda** (back-end)
  - **Algoritmos** para redução dos dados mostrados
  - Uso de **HTML5 canvas** para reduzir o número de objetos

#dataviz

visualização de BigData com D3.js

- Esta palestra estará disponível para download em
  - http://www.argonavis.com.br/download/tdc_2016_big_data_dataviz.html
- Referências dos gráficos, livros, código e links usados na apresentação também estarão disponíveis na página acima.

helder da rocha

helder@summa.com.br