Data-Driven Documents

D3.js - Visualisation de données géographiques

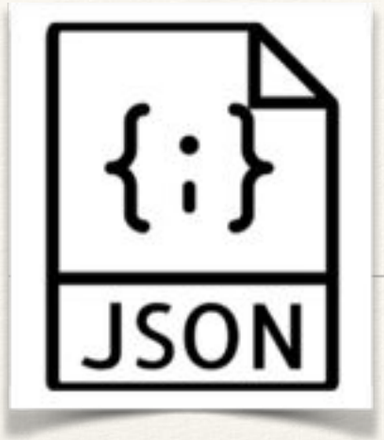
Alexandra BI
Idriss BRAHAMI
Lisa DESOUBEAUX
Luis VENTURA

Objectif

Utiliser des données géographiques au format **GeoJSON** pour dessiner une **carte interactive** dans un document HTML à l'aide de la bibliothèque JavaScript **D3.js**.

Sommaire

1. JSON
2. GeoJSON
3. Fonctions de Projection
4. Path SVG
5. Générateur de Path
6. Chargement de donnée d3
7. Chargement de donnée II
8. Création d’axes
9. Référence



JSON stockage de donnée

- **J ava**
- **S cript**
- **O bject**
- **N otation**

- { **"clé" : valeur**}
- **Plusieurs types, eg:** *string, int...*
- **Accèss facile eg:** *family.jason.age = 24*
- **Un objet de JSON peut est une chaîne de paires de {clé1: valeur1, clé2: valeur2, …}**

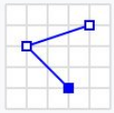
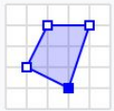
```
1   var family = {
2       "jason" : {
3           "name" : "Jason Lengstorf",
4           "age" : "24",
5           "gender" : "male"
6       },
7       "kyle" : {
8           "name" : "Kyle Lengstorf",
9           "age" : "21",
10          "gender" : "male"
11      }
12  }
```

GeoJSON

- **GeoJSON : JSON géographique**

GeoJSON s'appuie sur le format ouvert JSON et y ajoute des conventions de formatage supplémentaire pour l'encodage de données géographiques.

- Exemples de géométries GeoJSON :
  - `Point`
  - `LineString`
  - `Polygon`
  - `MultiPoint`
  - `MultiLineString`
  - `MultiPolygon`

Point		`{ "type": "Point", "coordinates": [30, 10] }`
LineString		`{ "type": "LineString", "coordinates": [ [30, 10], [10, 30], [40, 40] ] }`
Polygon		`{ "type": "Polygon", "coordinates": [ [[30, 10], [40, 40], [20, 40], [10, 20], [30, 10]] ] }`
		`{ "type": "Polygon", "coordinates": [ [[35, 10], [45, 45], [15, 40], [10, 20], [35, 10]], [[20, 30], [35, 35], [30, 20], [20, 30]] ] }`

https://en.wikipedia.org/wiki/GeoJSON

GeoJSON

Ce GeoJSON contient le tableau d'objets JSON "features" avec 3 objets représentants :

- Africa
- Australia
- Timbuktu

Chaque objet/feature comporte un objet *'geometry'* et un objet *'properties'*.

- 'geometry' contient des information spatiales, le type de forme, les coordonnées sphériques.
- 'properties' contient des informations non-spatiales comme le *nom de l'endroit, la population, GDP etc*

D3.js s'occupe de gérer la complicité de toutes les sortes d'informations contenues dans un objet GeoJSON


```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "name": "Africa"
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [[[-6, 36], [33, 30], ... , [-6, 36]]]
      }
    },
    {
      "type": "Feature",
      "properties": {
        "name": "Australia"
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [[[143, -11], [153, -28], ... , [143, -11]]]
      }
    },
    {
      "type": "Feature",
      "properties": {
        "name": "Timbuktu"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [-3.0026, 16.7666]
      }
    }
  ]
}
```

GeoJSON

- Les données sont stockées dans une liste *“features”*.

- “nom_json.features” donne l’accès aux données.

```
deps.selectAll("path")
    .data(geojson.features)
    .enter()
```

Penser bien à ajouter *.features* quand vous chargez les données.


```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "name": "Africa"
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [[[-6, 36], [33, 30], ... , [-6, 36]]]
      }
    },
    {
      "type": "Feature",
      "properties": {
        "name": "Australia"
      },
      "geometry": {
        "type": "Polygon",
        "coordinates": [[[143, -11], [153, -28], ... , [143, -11]]]
      }
    },
    {
      "type": "Feature",
      "properties": {
        "name": "Timbuktu"
      },
      "geometry": {
        "type": "Point",
        "coordinates": [-3.0026, 16.7666]
      }
    }
  ]
}
```

GeoJSON

Avec des attribues non-spatiales : Feature , FeatureCollection (ref: http://geojsonlint.com)

GeoJSONLint | Point | LineString | Polygon | Feature | FeatureCollection | GeometryCollection

Use this site to validate and view your GeoJSON. For details about GeoJSON, read the spec.

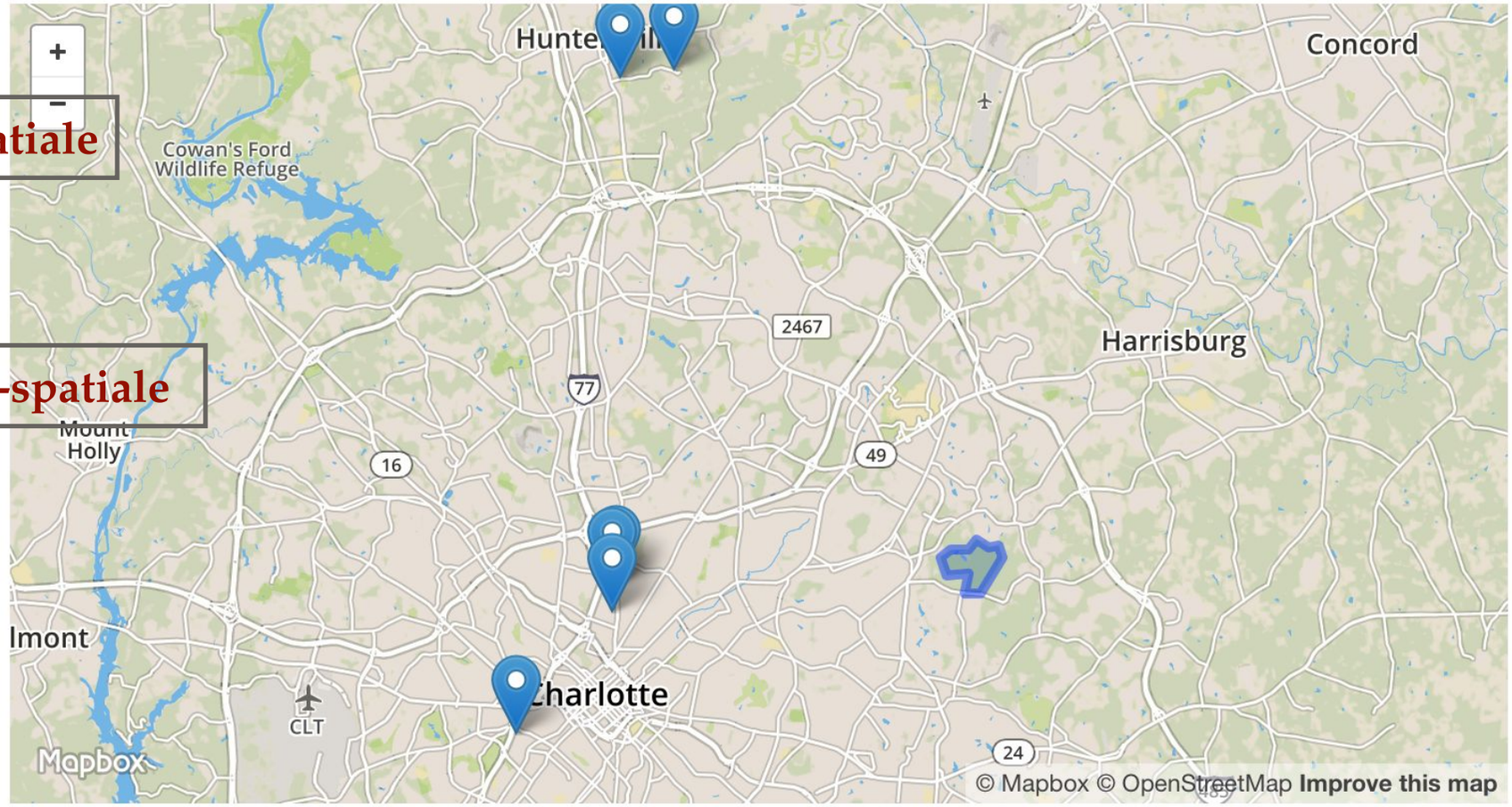
```
"type": "FeatureCollection",
"features": [
  {
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -80.870885,
        35.215151
      ]
    },
    "properties": {
      "name": "ABBOTT NEIGHBORHOOD
PARK",
      "address": "1300  SPRUCE ST"
    }
```

Information spatiale

Information non-spatiale

Clear Current Features

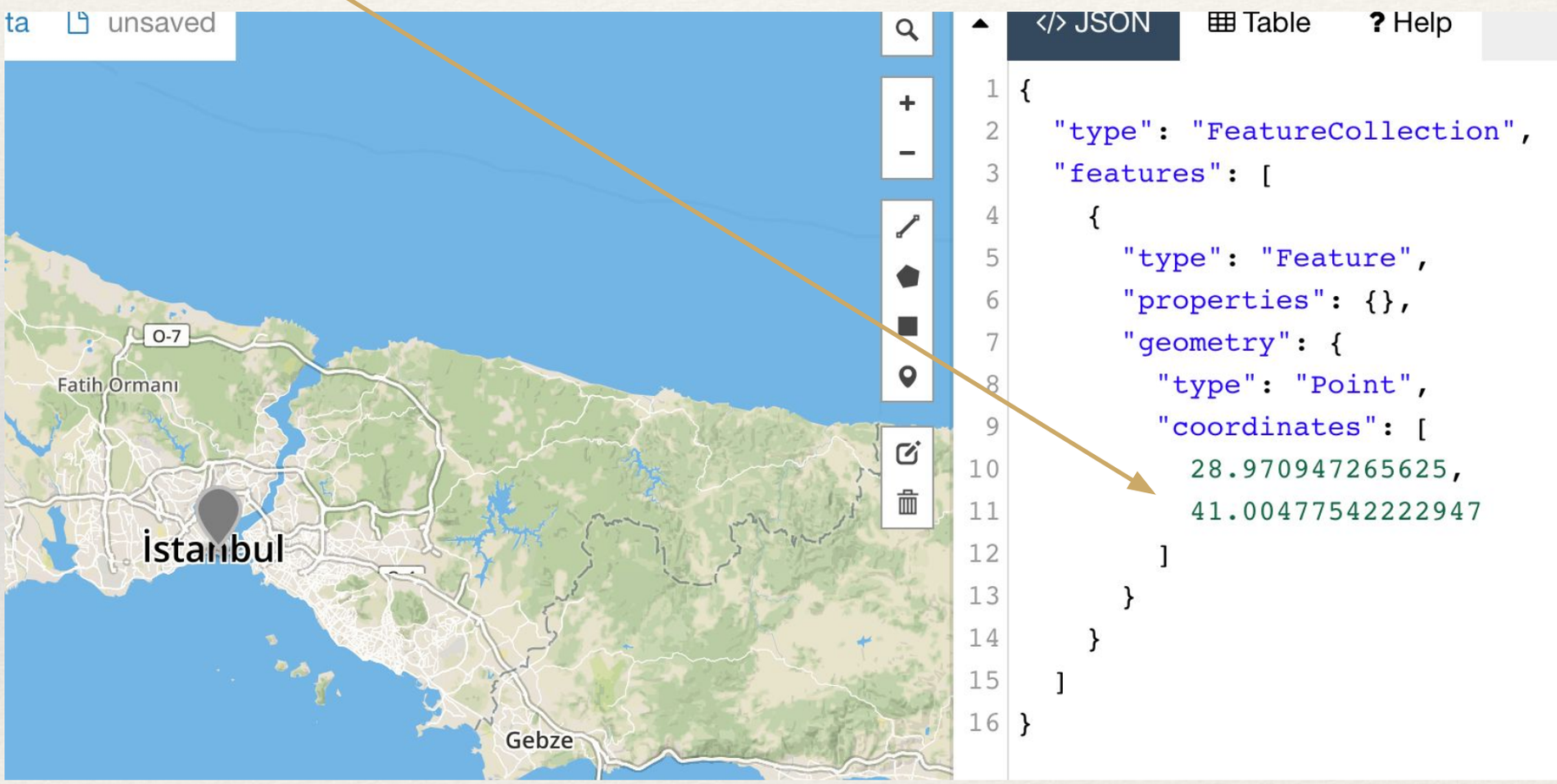
Test GeoJSON | Clear



GeoJSON

GeoJSON contient des coordonnées géographiques ,
ou [**longitude, latitude**]

eg:

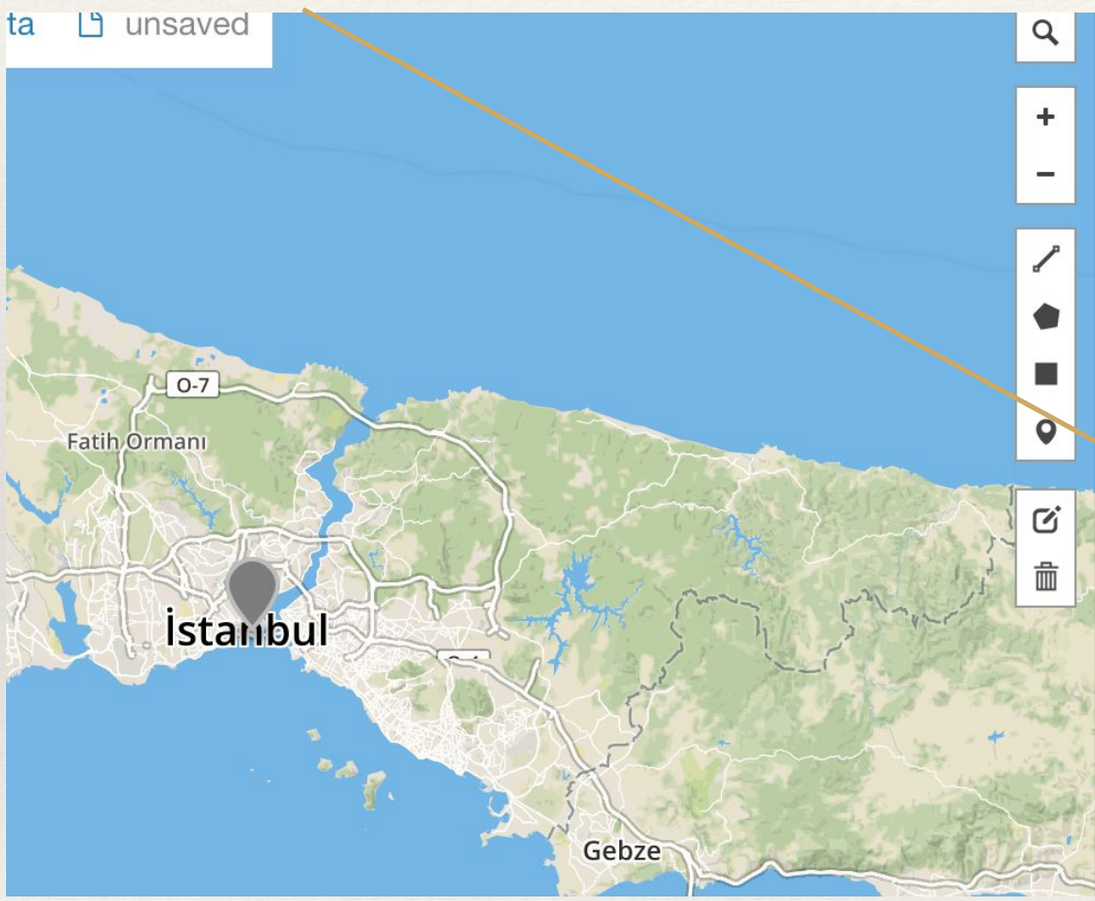


reference: http://geojson.io/#map=2/20.0/0.0

GeoJSON

GeoJSON contient des coordonnées géographiques ,
[longitude, latitude]

eg:



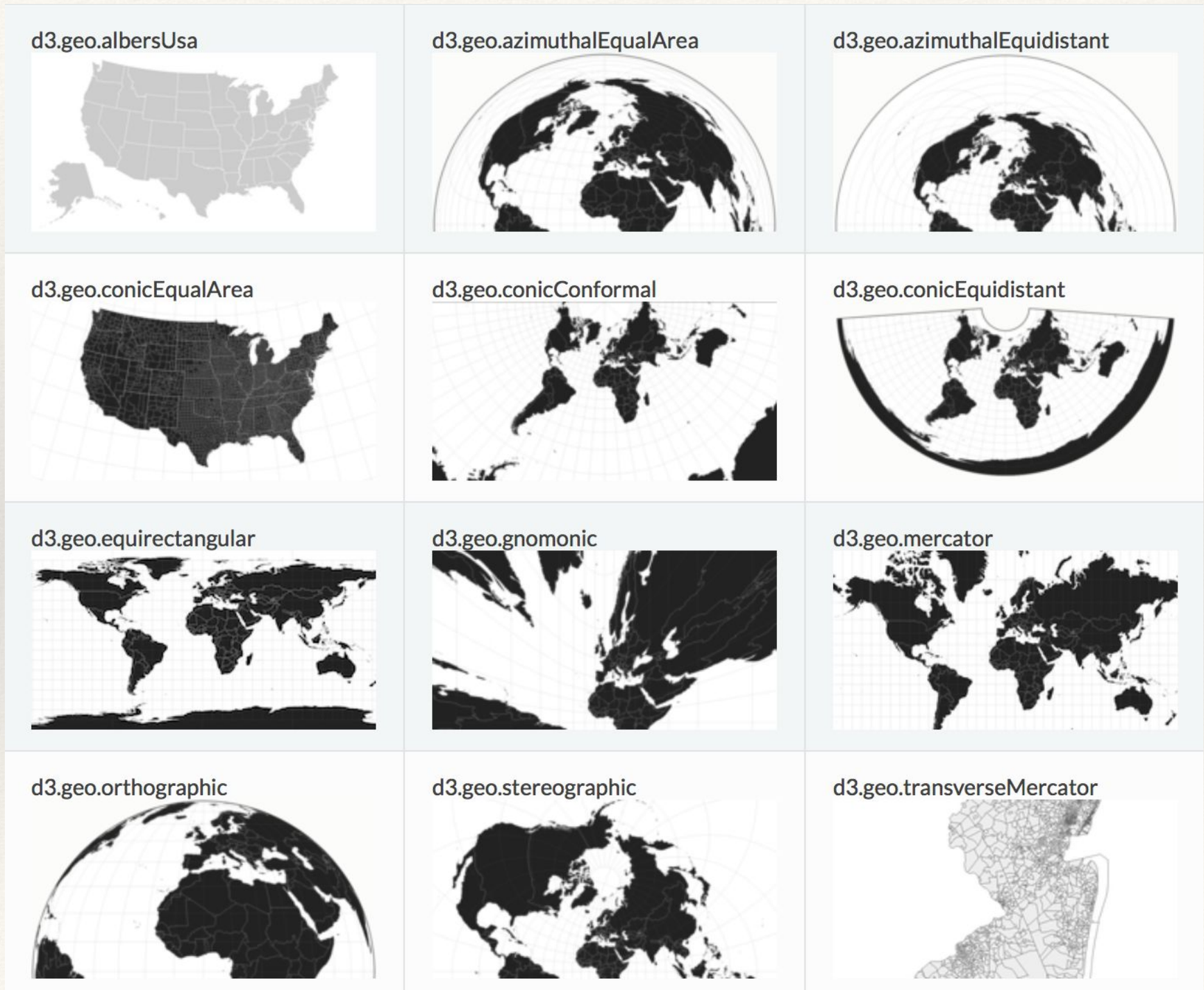

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {},
      "geometry": {
        "type": "Point",
        "coordinates": [
          28.970947265625,
          41.00477542222947
        ]
      }
    }
  ]
}
```


Il faut les convertir en système Cartésien [x, y] via une fonction de projection !

reference: http://geojson.io/#map=2/20.0/0.0

Projection

- D3.js contient différentes méthodes de projection pour passer de *[lon, lat]* à *[x, y]*



Objet de Projection

```js

var projection = d3.geoEquirectangular();

projection( [-3.0026, 16.7666] )

// returns [474.7594743879618, 220.7367625635119]
```

L'instanciation de *d3.geoEquirectangular()* crée un objet de *'projection'*.
Ici la projection utilisée est de type "Equirectangular()"

D3.js dessine avec des Path

- SVG permet de créer des formes complexes grâce aux balises <path>, par exemple pour dessiner une *LineString* (ensemble de lignes reliées les unes aux autres) :
  - *eg: <path d="*M0,80L100,100L200,30L300,50L400,40L500,80*">*
  - Move to (0,80), Line(100,100), Line(200,30), Line(300, 50), Line(400, 40), Line(500, 80)
- D3.js dispose d'un *Générateurs de Path* pour créer *des listes de commande* à partir d'un ensemble de coordonnées cartésiennes

Générateur de Path

- d3.line()
- **Générateur de Path en Ligne**

JS

```js
var lineGenerator = d3.line();

// lineGeneraeur prend comme variale d'entrée une liste de coordonnées

var points = [
  [0, 80],[100, 100],[200, 30],[300, 50],
  [400, 40],[500, 80]
];


var pathData=lineGenerator(points);
// pathData is "M0,80L100,100L200,30L300,50L400,40L500,80"
```

Générateur de GéoPath

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

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

- *d3.geoPath()* **est appelé pour générer un Path à partir des coordonnées cartésiennes**
- **Il possède une méthode** *.projection()* **pour définir la méthode de** *Projection* **à utiliser**
- *d3.geoPath.projection()* **prends en paramètre la fonction de projection**

Générateur de GéoPath

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

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

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

geoGenerator(geoJson);
// returns "M464.0166237760863,154.09974265651798L491.1506253268278,154.8895088551978 ...
L448.03311471280136,183.1346693994119Z"
```

GeoProjection [lon, lat] en [x,y]

Créer un Geo-générateur de path

Passer l'objet de Geo-projection

Générer le *'path'* final

Le Géo-Générateur de path, ie *geoGenerator*, reconnaît le format GeoJSON pour générer un Path *'machine-friendly'*.

Chargement de données JSON

```
path = d3.geopath();
path.projection(d3.geoConicConformal());

d3.json('https://www.site.me/donnesgeo.json').then(function(rawgeojson) {

    var svg = d3.select("#mon-svg").append("g")
    svg.selectAll("path")
        .data(rawgeojson.features)
        .enter()
        .append("path")
        .attr("d", path)
        ;
});
```

GeoProjection [lon, lat] en [x,y]

Charge les données JSON, PUIS appelle la fonction en lui passant l'ensemble des données

D3.js, en interne, boucle sur notre tableau de features et utilise le générateur de path "path" pour créer une balise <path> pour chacune des géométries du GeoJSON

1) selectAll("path") crée, une sélection de balise <path>
2) data() lit la collection de features du geoJSON
3) enter() crée autant de balise <path> que de géométries dans le geoJSON et entame une boucle en interne, **les méthodes qui suivent vont donc être exécutées autant de fois qu'il y a d'éléments dans rawgeojson.features**
4) On peut alors ajouter la balise au DOM et utiliser notre générateur de path pour la mise à jour des attributs de chaque balise <path>

Encore! Chargement de données

```
8   var projection = d3.geoEquirectangular();
9   var geoGenerator = d3.geoPath()
10      .projection(projection);
11  /*
12    ' .select() ' --selectionner un DOM objet qui va contenir le dessin
13    ' .selectAll() ' --selectionner tous les elements de type 'path', AVANT de les créer
14    ' .data() ' --lier les données GeoJSON à ces element de 'path'
15   */
16  var u = d3.select('g')
17      .selectAll('path')
18      .data(geojson.features);
19  /*
20      ' .enter() ' balaie la liste de donnée JSON, donnée par donnée. à chaque fois,
21        une caise virtuelle est crée pour y insérer plus tard un element de type 'path'
22
23      ' .append() ' insère un element de type 'path' dans une caise pour chaque donnée
24
25      ' .attr() '  contient 'd' la donnée JSON et
26        la fonction 'geoGenerator' transforme JSON vers le format désiré
27  */
28    u.enter()
29      .append('path')
30      .attr('d', geoGenerator);
```

Création d'axes et utilisation

```
//Je créé x et y qui sont les échelles et les domaines de défnition  de mes deux axes
x = d3.scaleBand().range([0, 400]).domain(["2014","2015","2016"]).padding("0.5");



y = d3.scaleLinear().range([600, 0]).domain([0,1000]);


//Je construis mes axes
var xAxis = d3.axisBottom(x);

var yAxis = d3.axisLeft(y);


//J'ajoute les axes au DOM :
SVG = d3.select('#mon-svg')

SVG.append("g")
        .attr("transform","translate(0,"+heightChart+")")
        .call(xAxis)

SVG.append("g")
      .call(yAxis)
      .attr("transform","translate("+margin.left+",0)")
```

.range() paramètre l'étendue de l'axe en pixel dans le DOM

.domain() définit le domaine de définition de l'axe cad les valeurs qu'il va afficher

d3.scaleBand() pour créer un schema des columns
d3.scaleLinear() pour créer une ligne/axe droite

Par défaut l'axe horizontal est placé en haut, il faut donc le translater de la hauteur du SVG sur y

Il existe plusieurs types de "scale" pour les axes selon ce que l'on veut afficher : linéaire, Band (valeurs discrètes), Time (valeurs temporelles)...

Pour positionner [x='2014',y=200] sur nos axes, on utilise les méthodes **x('2014')** qui renvoie la position en pixel sur x et **y(200)** qui renvoie la position en pixel sur y

Réferences

GeoJSON

https://en.wikipedia.org/wiki/GeoJSON

http://geojsonlint.com

http://geojson.io/#map=2/20.0/0.0

Sources Tutoriel

https://www.dashingd3js.com/table-of-contents (EN)

https://d3indepth.com/geographic/ (EN)

https://www.datavis.fr/index.php#d3js (FR)

Exemples D3.js

https://www.datavis.fr/index.php?page=map-population (population de la France)

https://www.datavis.fr/index.php?page=linearchart (time -series)

https://www.datavis.fr/index.php?page=earthquake (l'échelle des séismes du monde 1900- 2014)

Les compétences pré-requises

- Créer un *SVG*
- Créer des objets au sein du SVG
- Un objet de '*group*', ou '*g*'
- Savoir modifier les attributs d'un élément
- Lier des ensemble de données à des éléments path
    - attribut 'd'
    - méthode enter()

D3.JS - Data **D**riven **D**ocuments

Les compétences à acquérir

D3.JS - Data **D**riven **D**ocuments

- Les données pour construire une carte: *JSON, GeoJSON, (TopoJSON)*
- Conversion des Geo-JSON: *projeter Geo-JSON sur une surface de 2D*
- Notion de ‘path’ pour D3.js
- Générateur de ‘path’ et projection
- Elément groupe ‘g’