Cours: VISUALISATION DE DONNEES

Master MIV, 2020/2021

Prof. Slimane LARABI

Chapitre 3:
Visualisation de données Géospatiales

Sommaire

Prof. Slimane LARABI

Chapitre 3: Visualisation de données Géospatiales

3.1: Introduction

Pour créer une carte Web, il possible d'utiliser:

- Une bibliothèque spécialisée comme Google Maps, Leaflet ou OpenLayers
- D3 qui fournit suffisamment de fonctionnalités de base pour créer tout type de carte.

Utiliser une bibliothèque dédiée telle que l'API Google Maps est due aux fonctionnalités supplémentaires de cet écosystème, telles que Street View.
Si l'écosystème n'est pas utilisé, D3 est conseillé.

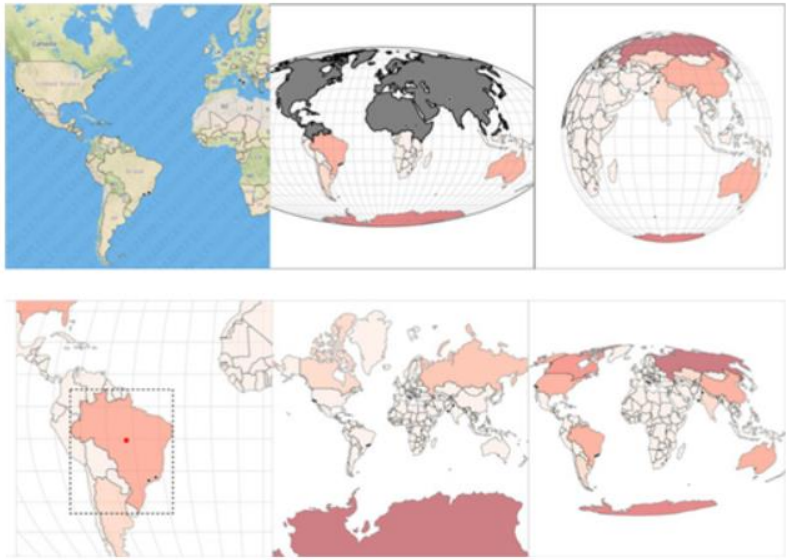
D3 dispose d'une fonctionnalité intégrée robuste pour charger et afficher des données géospatiales. Une bibliothèque, TopoJSON, offre davantage de fonctionnalités pour la visualisation des informations géospatiales.

Chapitre 3: Visualisation de données Géospatiales

3.1. Introduction

La cartographie avec D3 prend de nombreuses formes et offre de nombreuses options, y compris:

- les cartes traditionnelles,
- les opérations TopoJSON de pointe,
- les globes,
- les calculs spatiaux
- les cartes basées sur les données utilisant de nouvelles projections.



3.2. Création de cartes de base

Avec D3, on peut créer est une carte vectorielle utilisant:
- les éléments SVG <path>
- les éléments SVG <circle> pour représenter les pays et les villes.

cities.csv, pour les coordonnées,
- Autres données nécessaires pour représenter les pays (les rendre sous forme de zones, de lignes ou de points sur une carte).
- Ajouter de l'interactivité, le passage de la souris sur une région permet de calculer et afficher son centre.

3.2. Création de cartes de base

```
path.countries {stroke-width: 1; stroke: black;  opacity: .5; fill: red;}
circle.cities { stroke-width: 1; stroke: black; fill: white;}
circle.centroid { fill: red; pointer-events: none;}
path.graticule { fill: none; stroke-width: 1;  stroke: black;}
path.graticule.outline {  stroke: black; }
```

3.2. Création de cartes de base

Les données géographiques peuvent prendre plusieurs formes.

Latitude et la longitude pour les points comme des villes.

cities.csv

"label","population","country","x","y"
"San Francisco", 750000,"USA",-122,37
"Fresno", 500000,"USA",-119,36
"Lahore",12500000,"Pakistan",74,31
"Karachi",13000000,"Pakistan",67,24
"Rome",2500000,"Italy",12,41
"Naples",1000000,"Italy",14,40
"Rio",12300000,"Brazil",-43,-22
"Sao Paolo",12300000,"Brazil",-46,-23

3.2. Création de cartes de base

Le traitement de données géospatiales complexes (formes, lignes) nécessite des formats données plus complexes data formats tels que GeoJSON qui est un standard pour le (web-mapping data).

3.2. Création de cartes de base

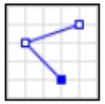
GeoJSON

GeoJSON (geojson.org) codage de données géospatiales format JSON.

Les géométries de base dans GeoJSON sont des points, des chaînes de lignes et des polygones. La description de base d'une entité GeoJSON utilise la syntaxe suivante:

```
{    "type": name of the type of geometry (point, line string, ...)
     "coordinates": one or more tuple of latitude / longitude
}
```

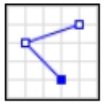
```
{   "type": "Point",    "coordinates": [30, 10]}
{    "type": "LineString",
"coordinates": [[30, 10], [10, 30], [40, 40] ]}
```



3.2. Création de cartes de base

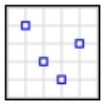
GeoJSON

```
{    "type": "Point",     "coordinates": [30, 10]}
```

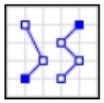


```
{     "type": "LineString",
"coordinates": [[30, 10], [10, 30], [40, 40] ]}
```

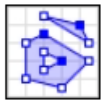
```
{ "type": "Polygon",
  "coordinates":     [ [[30, 10], [40, 40], [20, 40], [10, 20], [30, 10]]]}
```



```
{"type": "MultiPoint",
"coordinates":      [[10, 40], [40, 30],       [20, 20], [30, 10]]}
```



```
{ "type": MultiLineString",
"coordinates":  [ [[10, 10], [20, 20], [10, 40]],
                  [[40, 40], [30, 30], [40, 20], [30, 10]] ]}
```



```
{ "type": "MultiPolygon",
  "coordinates": [[ [[40, 40], [20, 45], [45, 30], [40, 40]] ],
                   [ [[20, 35], [10, 30], [10, 10], [30, 5], [45, 20], [20, 35]],
                  [[30, 20], [20, 15], [20, 25], [30, 20]]  ]  ]}
```

3.2. Création de cartes de base

GeoJSON

GeoJSON (geojson.org) codage de données géospatiales format JSON.

Ces géométries de base peuvent être enveloppées dans une entité (feature).

Feature contient une géométrie et également un ensemble de propriétés.
À titre d'exemple, Feature consiste en une géométrie ponctuelle et qui a une seule propriété, nom.

```
{  "type": "Feature",
"geometry": {    "type": "Point",
                 "coordinates": [46.862633, -114.011593]     },
                 "properties": {    "name": "Missoula"     }}
```

3.2. Création de cartes de base

GeoJSON

En utilisant un niveau de plus dans la hiérarchie, *featureCollection permet de* définir ce que l'on appelle une collection de features.

Chaque *feature* dans une *featureCollection* est un objet JSON qui définit la frontière de feature avec un tableau de *coordonnées et des* meta données relatives à feature.

Pour dessiner un rectangle on utilise des sommets [-74.0479, 40.6829], [-74.0479, 40.8820], [-73.9067, 40.8820], [-73.9067, 40.6829].

3.2. Création de cartes de base

GeoJSON

{ "type": "FeatureCollection",
"features": [
{ "type": "Feature",
"geometry": {"type": "Point", "coordinates": [102.0, 0.5]},
"properties": {"prop0": "value0"} },

{ "type": "Feature",
"geometry": {"type": "LineString", "coordinates": [[102.0, 0.0], [103.0, 1.0],[104.0, 0.0], [105.0, 1.0]] },
"properties": { "prop0": "value0", "prop1": 0.0 }},

{ "type": "Feature",
"geometry": {"type": "Polygon", "coordinates": [[[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0], [100.0, 0.0]]] },
"properties": {"prop0": "value0", "prop1": {"this": "that"} } }] }

3.2. Création de cartes de base

TOPOJSON

Mais l'un des problèmes de GeoJSON est qu'il est très détaillé et que certaines géométries et fonctionnalités ne peuvent pas être réutilisées. Si la même géométrie est requise à plusieurs endroits, elle doit être complètement spécifiée une deuxième fois.

Pour résoudre cette situation, TopoJSON a été créé.
TopoJSON fournit des constructions supplémentaires pour le codage de la topologie et la réutilisation.

Au lieu de décrire discrètement chaque géométrie, TopoJSON permet de définir des géométries, puis de les assembler à l'aide de concepts appelés arcs.

Arcs permet à TopoJSON d'éliminer la redondance et de fournir une représentation beaucoup plus compacte que GeoJSON. Il est indiqué que TopoJSON peut généralement fournir une compression de 80% sur GeoJSON.

3.2. Création de cartes de base

TOPOJSON { "type": "Topology",
"objects": {"example": {"type": "GeometryCollection",
"geometries": [
{ "type": "Point", "properties": {"prop0": "value0" }, "coordinates": [102, 0.5]},
{ "type": "LineString", "properties": {"prop0": "value0", "prop1": 0 }, "arcs": [0] },
{ "type": "Polygon", "properties": {"prop0": "value0", "prop1": {"this": "that" } },
"arcs": [[-2]]}] }},
"arcs": [[[102, 0], [103, 1], [104, 0], [105, 1]],
[[100, 0], [101, 0], [101, 1], [100, 1], [100, 0]]
]
}

Cet objet TopoJSON a trois propriétés: le type, les objets et les arcs.
La valeur de type est toujours "topology". La propriété des objets consiste en une collection de géométries similaires à celles de GeoJSON, à la différence qu'au lieu de spécifier des coordonnées, l'objet peut, à la place, spécifier un ou plusieurs arcs.

3.2. Création de cartes de base

TOPOJSON { "type": "Topology",
"objects": {"example": {"type": "GeometryCollection",
"geometries": [
{ "type": "Point", "properties": {"prop0": "value0" }, "coordinates": [102, 0.5]},
{ "type": "LineString", "properties": {"prop0": "value0", "prop1": 0 }, "arcs": [0] },
{ "type": "Polygon", "properties": {"prop0": "value0", "prop1": {"this": "that" } },
"arcs": [[-2]]}] }},
"arcs": [[[102, 0], [103, 1], [104, 0], [105, 1]],
[[100, 0], [101, 0], [101, 1], [100, 1], [100, 0]]
]
}

L'objet polygone fait référence à un arc avec la valeur -2. Une valeur d'arc négative spécifie que le complément à un de l'arc qui doit être utilisé. Cela implique essentiellement que les positions dans l'arc doivent être inversées. Par conséquent, -2 indique d'obtenir la position inversée du deuxième arc. C'est l'une des stratégies utilisées par TopoJSON pour réutiliser et compresser les données.

3.2. Création de cartes de base

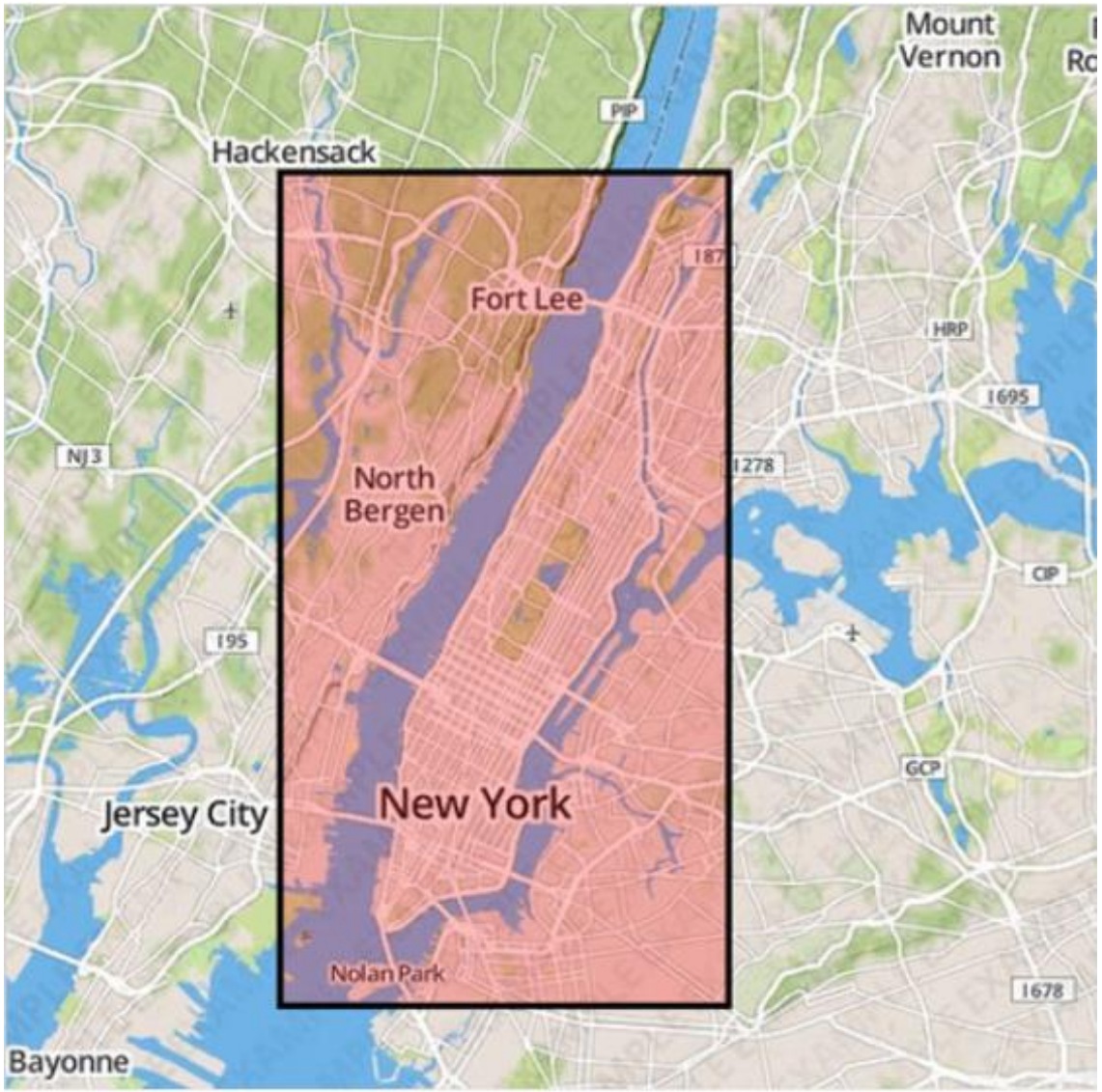
GeoJSON

En utilisant un niveau de plus dans la hiérarchie, *featureCollection permet de* définir ce que l'on appelle une collection de features.

Chaque *feature* dans une *featureCollection* est un objet JSON qui définit la frontière de feature avec un tableau de *coordonnées et des* meta données relatives à feature.

Pour dessiner un rectangle on utilise des sommets [-74.0479, 40.6829], [-74.0479, 40.8820], [-73.9067, 40.8820], [-73.9067, 40.6829].

3.2. Création de cartes de base



3.2. Création de cartes de base

GeoJSON


```
{  "type": "FeatureCollection",
   "features": [   {
        "type": "Feature",
        "id": "LUX",
        "properties": {
            "name": "Luxembourg"},
            "geometry": {
            "type": "Polygon",
            "coordinates": [  [  [  6.043073,   50.128052 ],
                [6.242751, 49.902226 ],
                [6.18632, 49.463803 ],
                [5.897759, 49.442667 ],
                [5.674052, 49.529484],
                [5.782417, 50.090328],
                [6.043073, 50.128052] ]]}}]}
```

3.3. Projections

Créer un fichier GeoJSON

world.geojson (emeeks.github.io/d3ia/world.geojson), contient les pays du monde avec une faible résolution.

Projection

projection signifie le processus du rendu des points du globe sur une surface plane.

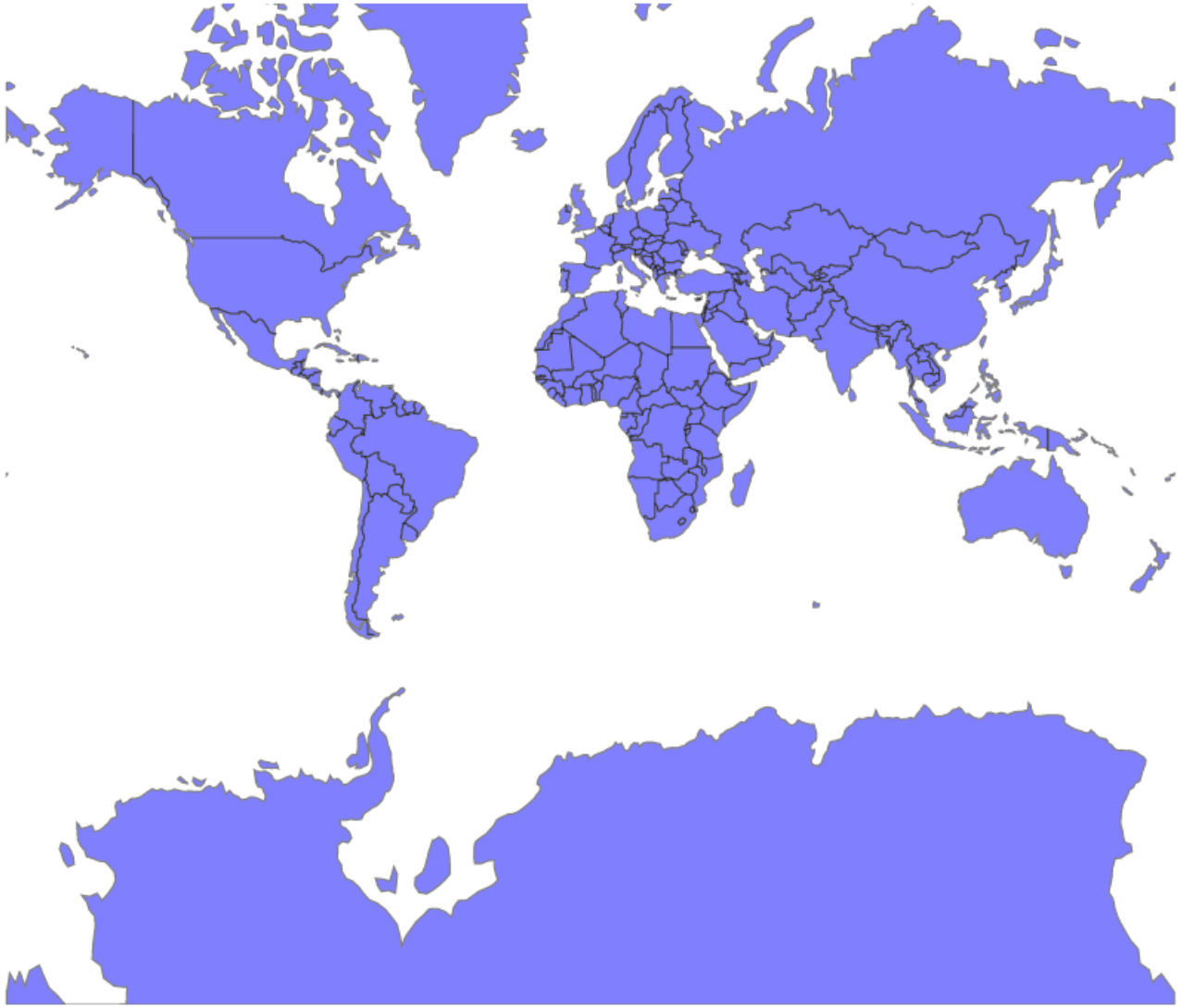
On peut faire différentes projections.

3.3. Projections

La projection Mercator est l'une des projections géographiques les plus courantes, qui est utilisée dans la plupart des cartes Web. C'est devenu la norme de facto car c'est la projection utilisée par Google Maps.
Pour utiliser la projection Mercator, il faut inclure une extension de D3, **d3.geo.projection.js**.
d3.geo.path, dessine les données géospatiales à l'écran en fonction de la projection sélectionnée.

```
d3.json("world.geojson", createMap);
function createMap(countries) {
  var aProjection = d3.geo.mercator();
  var geoPath = d3.geo.path().projection(aProjection);
  d3.select("svg").selectAll("path").data(countries.features)
   .enter()
   .append("path")
   .attr("d", geoPath)
   .attr("class", "countries");};
```

3.3. Projections



3.3. Projections

La projection Mercator ne montrent qu'une partie du monde sur le canevas SVG. Chaque projection a un .translate () et .scale () qui suivent la syntaxe de la transformation en SVG.

scale

Pour la projection Mercator, si nous divisons la largeur de l'espace disponible par 2 et divisons le quotient par π, alors le résultat sera l'échelle appropriée pour afficher le monde entier dans l'espace disponible.

3.3. Projections

Différentes familles de projections ont des échelles par défaut différentes.

d3.geo.albers-Usa est par défaut de 1070,
d3.geo.mercator est par défaut de 150.

La valeur par défaut est obtenu en appelant la fonction sans lui passer de valeur:

d3.geo.mercator().scale()
d3.geo.albersUsa().scale()

En ajustant la translation et l'échelle, nous pouvons ajuster la projection pour afficher différentes parties des données géospatiales

```
> d3.geo.albersUsa().scale()
<· 1000
> d3.geo.mercator().scale()
<· 500
>
```

3.3. Projections

```
function createMap(countries) {
  var width = 500;
  var height = 500;
  var aProjection = d3.geo.mercator()
  .scale(200)
  .translate([width / 2, height / 2]);

var geoPath = d3.geo.path().projection(aProjection);
d3.select("svg").selectAll("path").data(countries.features)
                .enter()
                .append("path")
                .attr("d", geoPath)
                .attr("class", "countries");
                };
```

3.3. Projections

3.4. Dessin d'un point sur une map

La projection est aussi utilisée pour placer des points individuels.
Exemple: les villes sont représentées avec un seul point sur une carte.

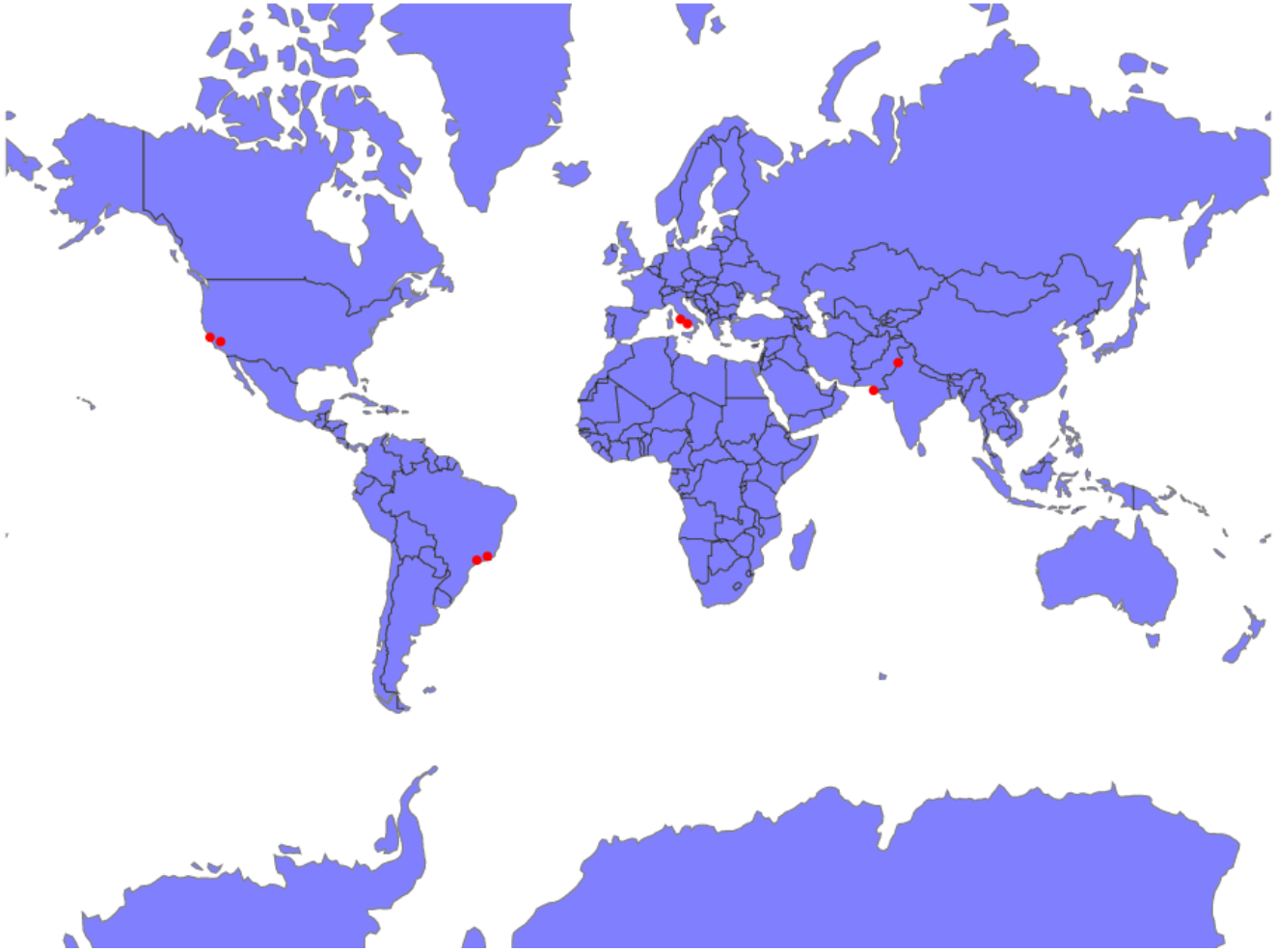
Une projection D3 peut être utilisée non seulement dans un geo.path () mais aussi en tant que fonction seule avec un tableau d'une paire de coordonnées de latitude et de longitude,
il renvoie les coordonnées d'écran nécessaires pour placer ce point.

San Francisco
aProjection([-122,37]) deux valeurs x', y'

3.3. Dessin d'un point sur une map

```
d3.select("svg").selectAll("circle").data(cities)
      .enter() .append("circle")
      .style("fill", "red") .attr("class", "cities")
      .attr("r", 3) .attr("cx", function(d) {return projection([d.x,d.y])[0]})
      .attr("cy", function(d) {return projection([d.x,d.y])[1]});
```

3.3. Dessin d'un point sur une map



3.4. Projections et surfaces (areas)

La taille graphique des objets dépend de la projection géographique utilisée.

Il est impossible d'afficher parfaitement les coordonnées sphériques sur une surface plane.
Différentes projections sont conçues pour afficher visuellement la zone géographique des régions terrestres ou océaniques, ou la distance mesurable, ou des formes particulières.

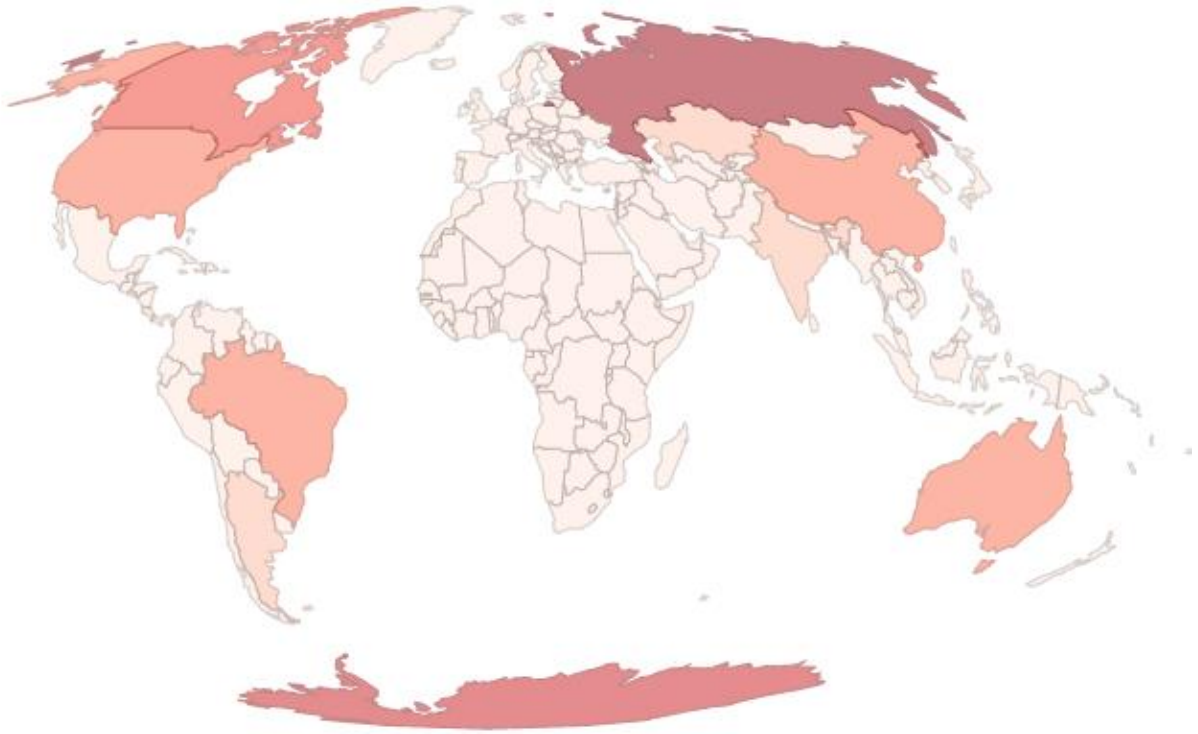
d3.geo.projection.js, permet d'accéder à un certain nombre de projections supplémentaires dont l'une est la projection de Mollweide (**projection elliptique)**.

Dans le code suivant, un affichage correct est réalisé avec une projection Mollweide des données géospatiales.
La zone calculée des pays est la zone graphique, et non leur zone physique réelle.

3.4. Projections et surfaces (areas)

```
d3.json("world.geojson", createMap);
function createMap(countries) {
var projection = d3.geoMollweide().scale(120).translate([250, 250]);
var geoPath = d3.geoPath().projection(projection);
var featureSize = d3.extent(countries.features, d => geoPath.area(d))
var countryColor = d3.scaleQuantize()
                        .domain(featureSize).range(colorbrewer.Reds[7]);
d3.select("svg").selectAll("path")
            .data(countries.features)
            .enter()
            .append("path")
            .attr("d", geoPath)
            .attr("class", "countries")
            .style("fill", d => countryColor(geoPath.area(d)))
            .style("stroke", d => d3.rgb(countryColor(geoPath.area(d))).darker())
}                                                   };
```

3.4. Projections et surfaces (areas)



3.4. Projections et surfaces (areas)

Choix de la bonne projection dépend des objectifs de la carte à créer.

Pour le mappage traditionnel de tuiles → Mercator.
Pour l'échelle mondiale, utiliser une projection à surface comme Mollweide qui ne déforme pas la zone visuelle des entités géographiques.

Etant donné que D3 propose une variété de projections, il est conseillé de faire des essais pour voir celle qui convient le mieux à la carte.

3.4. Projections et surfaces (areas)

CHOROPLETH MAP

Le terme carte choropleth est utilisé pour désigner une carte qui encode les données en utilisant la couleur d'une région.

On peut utiliser les caractéristiques géographiques existantes (pays) pour afficher des données statistiques, telles que le PIB d'un pays, sa population ou sa langue la plus utilisée.

Pour le faire avec D3 soit en:

- Obtenant les données géospatiales à partir du champ de propriétés contenant ces informations,
- Ou bien en liant une table de données géospatiales où elles ont toutes les deux les mêmes valeurs d'ID uniques en commun.

3.5. Interactivité

Une grande partie du code lié aux données géospatiales dans D3 est livré avec des fonctionnalités intégrées dont vous aurez généralement besoin lorsque vous travaillez avec des données géographiques.

D3 permet de calculer rapidement le centre d'une zone géographique (connue sous le nom de centroïde) et sa boîte englobante (voir figure et le code suivants).

On peut ajouter des événements de survol de la souris et dessiner un cercle au centre de chaque zone géographique, ainsi qu'une zone de délimitation autour d'elle

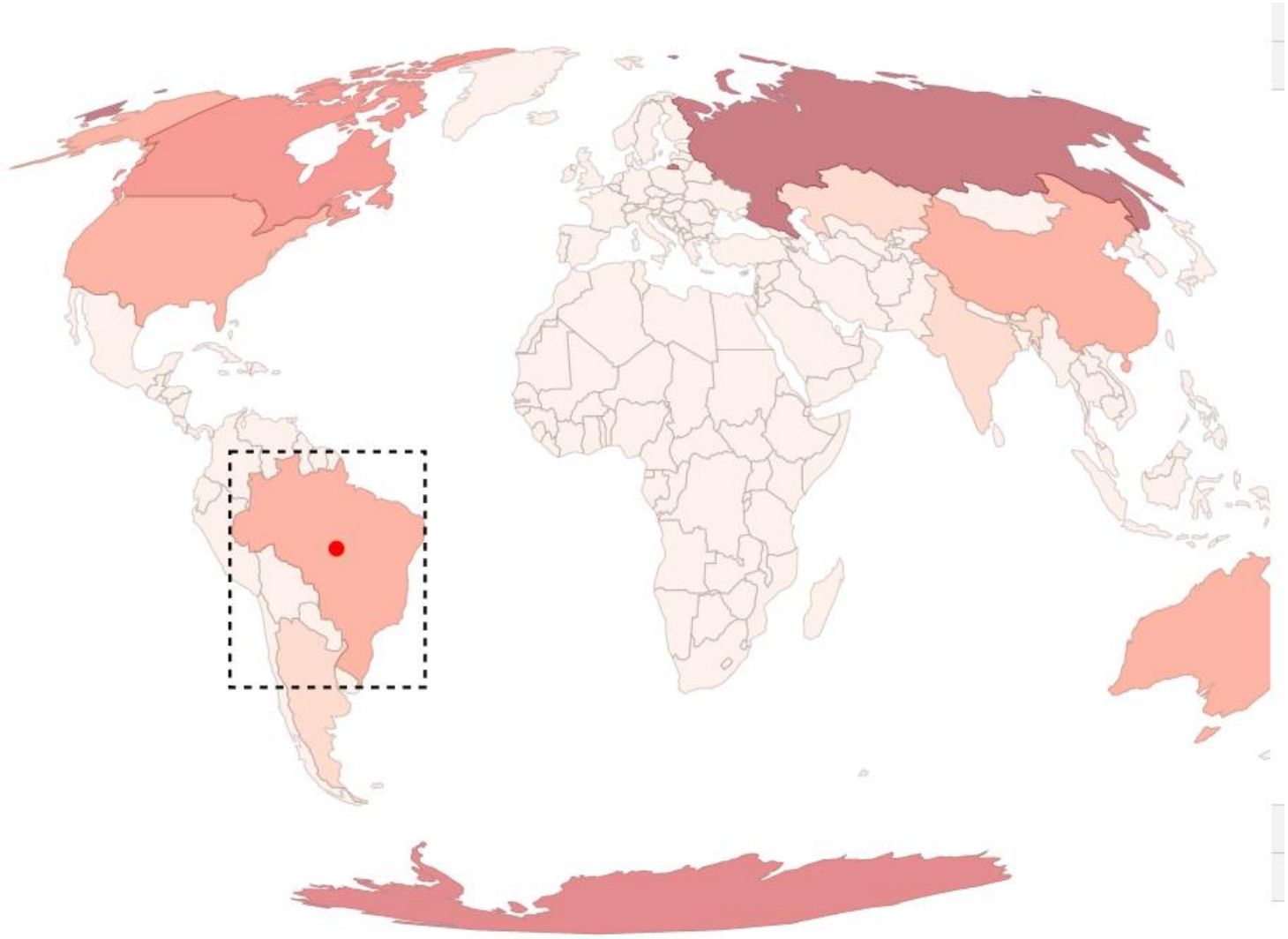
3.5. Interactivité

```
function createMap(countries) {
var projection = d3.geoMollweide()
.....................

d3.selectall("path.countries")
                .on("mouseover", centerbounds)
                .on("mouseout", clearcenterbounds);

function centerbounds(d,i) {
var thisbounds = geopath.bounds(d);
var thiscenter = geopath.centroid(d);
d3.select("svg")
                .append("rect")
                .attr("class", "bbox")
                .attr("x", thisbounds[0][0])
                .attr("y", thisbounds[0][1])
                .attr("width", thisbounds[1][0] - thisbounds[0][0])
                .attr("height", thisbounds[1][1] - thisbounds[0][1])
                .style("fill", "none")
                .style("stroke-dasharray", "5 5")
                .style("stroke", "black")
                .style("stroke-width", 2)
                .style("pointer-events", "none");
```

3.5. Interactivité

```
d3.select("svg")
                .append("circle")
                .attr("class", "centroid")
                .style("fill", "red")
                .attr("r", 5)
                .attr("cx", thisCenter[0])
                .attr("cy", thisCenter[1])
                .style("pointer-events", "none");
        };

function clearCenterBounds() {
    d3.selectAll("circle.centroid").remove();
   d3.selectAll("rect.bbox").remove();    };
};
```

3.5. Interactivité



3.6. Graticule

Pour rendre les cartes plus lisibles, on peut utiliser les fonctionnalités intégrées de d3.geo:

- le générateur de graticule, fournit des lignes de quadrillage qui facilitent la lecture d'une carte
- le zoom.

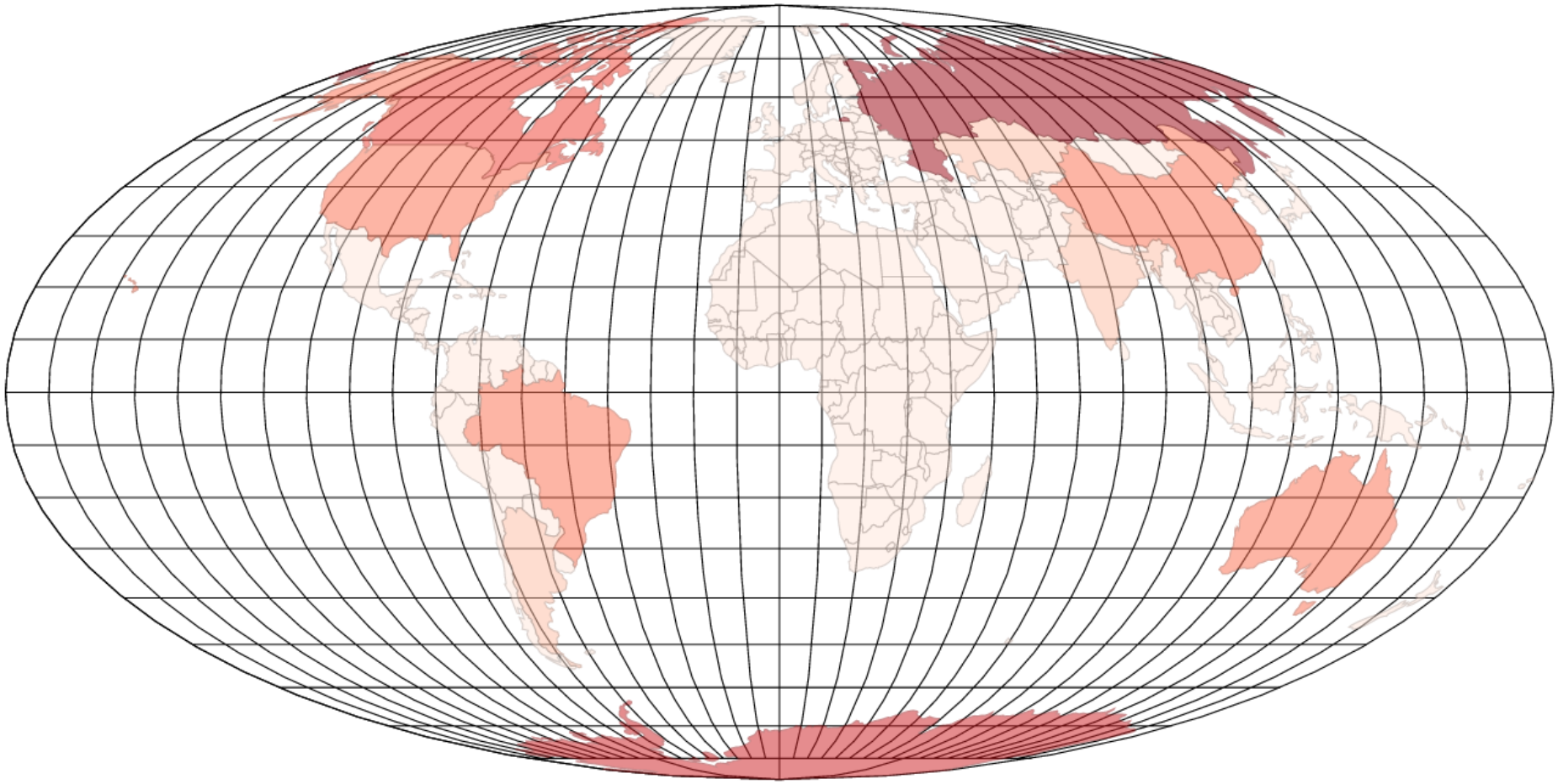
Un **Graticule** est une ligne de quadrillage sur une carte.
Le code suivant montre comment dessiner un **Graticule** sous les pays dessinés.

Au lieu de .data, nous utilisons .datum, qui est une fonction pratique qui nous permet de lier un seul point de données à une sélection afin qu'il n'ait pas besoin d'être dans un tableau.

3.6. Graticule

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

    d3.select("svg").append("path")
       .datum(graticule)
       .attr("class", "graticule line")
       .attr("d", geoPath)
       .style("fill", "none")
       .style("stroke", "lightgray")
       .style("stroke-width", "1px");

    d3.select("svg").append("path")
       .datum(graticule.outline)
       .attr("class", "graticule outline")
       .attr("d", geoPath)
       .style("fill", "none")
       .style("stroke", "black")
       .style("stroke-width", "1px");
```

3.6. Graticule



3.7. Utilisation de Topojson

TOPOJSON

Parmi les problèmes de GeoJSON:
- il est très détaillé
- certaines géométries et fonctionnalités ne peuvent pas être réutilisées. Si la même géométrie est requise à plusieurs endroits, elle doit être complètement spécifiée une deuxième fois.

Pour résoudre cette situation, TopoJSON a été créé.
TopoJSON fournit des constructions supplémentaires pour le codage de la topologie et la réutilisation.

Au lieu de décrire discrètement chaque géométrie, TopoJSON permet de définir des géométries, puis de les assembler à l'aide de concepts appelés arcs.

Arcs permet à TopoJSON d'éliminer la redondance et de fournir une représentation beaucoup plus compacte que GeoJSON. TopoJSON peut généralement fournir une compression de 80% sur GeoJSON.

3.7. Utilisation de Topojson

TOPOJSON { "type": "Topology",
"objects": {"example": {"type": "GeometryCollection",
"geometries": [
{ "type": "Point", "properties": {"propo": "valueo" }, "coordinates": [102, 0.5]},
{ "type": "LineString", "properties": {"propo": "valueo", "prop1": 0 }, "arcs": [0] },
{ "type": "Polygon", "properties": {"propo": "valueo", "prop1": {"this": "that" } },
"arcs": [[-2]]}] }},
"arcs": [[[102, 0], [103, 1], [104, 0], [105, 1]],
[[100, 0], [101, 0], [101, 1], [100, 1], [100, 0]]
]
}
Cet objet TopoJSON a trois propriétés: le type, les objets et les arcs.
La valeur de type est toujours "topology". La propriété des objets consiste en une collection de géométries similaires à celles de GeoJSON, à la différence qu'au lieu de spécifier des coordonnées, l'objet peut, à la place, spécifier un ou plusieurs arcs.

3.7. Utilisation de Topojson

```
TOPOJSON {  "type": "Topology",
"objects": {"example": {"type": "GeometryCollection",
"geometries": [
{ "type": "Point", "properties": {"propo": "valueo" }, "coordinates": [102, 0.5]},
{ "type": "LineString", "properties": {"propo": "valueo", "prop1": 0 }, "arcs": [0]  },
{ "type": "Polygon", "properties": {"propo": "valueo", "prop1": {"this": "that" } },
"arcs": [[-2]]} ] }},
"arcs": [ [[102, 0], [103, 1], [104, 0], [105, 1]],
            [[100, 0], [101, 0], [101, 1], [100, 1], [100, 0]]
          ]
}
```

les indices positifs commencent à 0 : 0, 1, 2… les indices négatifs vont eux commencer à -1 : -1, -2, -3. Ainsi le “négatif” de 0 est -1, celui de 1 est -2,

3.7. Utilisation de Topojson

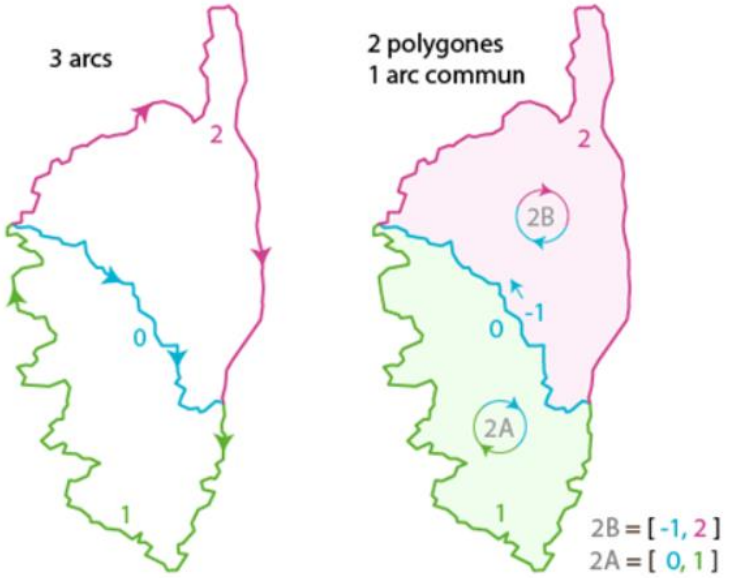
```
TOPOJSON {  "type": "Topology",
"objects": {"example": {"type": "GeometryCollection",
"geometries": [
{ "type": "Point", "properties": {"propo": "valueo" }, "coordinates": [102, 0.5]},
{ "type": "LineString", "properties": {"propo": "valueo", "prop1": 0 }, "arcs": [0]  },
{ "type": "Polygon", "properties": {"propo": "valueo", "prop1": {"this": "that" } },
"arcs": [[-2]]} ] }},
"arcs": [ [[102, 0], [103, 1], [104, 0], [105, 1]],
            [[100, 0], [101, 0], [101, 1], [100, 1], [100, 0]]
          ]
}
```

L'objet polygone fait référence à un arc avec la valeur -2. Une valeur d'arc négative spécifie que le complément à un de l'arc qui doit être utilisé. Cela implique essentiellement que les positions dans l'arc doivent être inversées. Par conséquent, -2 indique d'obtenir la position inversée du deuxième arc. C'est l'une des stratégies utilisées par TopoJSON pour réutiliser et compresser les données.

3.7. Utilisation de Topojson

```
TOPOJSON {  "type": "Topology",
"objects": {"example": {"type": "GeometryCollection",
"geometries": [
{ "type": "Point", "properties": {"propo": "valueo" }, "coordinates": [102, 0.5]},
{ "type": "LineString", "properties": {"propo": "valueo", "prop1": 0 }, "arcs": [0]  },
{ "type": "Polygon", "properties": {"propo": "valueo", "prop1": {"this": "that" } },
"arcs": [[-2]]} ] }},
"arcs": [ [[102, 0], [103, 1], [104, 0], [105, 1]],
            [[100, 0], [101, 0], [101, 1], [100, 1], [100, 0]]
          ]
}
d3.json('file.json', function(d) {
…
.data(topojson.feature(d, d.objects.example).features)
```

3.7. Utilisation de Topojson



France par région

- 26 régions
- 26 objets
- 107 arcs

3.7. Utilisation de Topojson

{"type":"Topology",
"transform":{"scale":[0.03600360036003601,0.017366249624962495],
"translate":[-180,-90]},
"objects":{"land":{"type":"MultiPolygon","arcs":[[[0]],[[1]],[

]}
,"countries":{"type":"GeometryCollection","geometries":[{"type":"Polygon","id":4,"arcs":[[297,298,299,300,301,302]]},{"type":"MultiPolygon","id":24,"arcs":[[[303,304,211,305]],[[213,306,307]]]},

{"type":"MultiPolygon","id":10,"arcs":[[[0]],[[1]],[[2]],[[3]

3.7. Utilisation de Topojson

Exemple: Les arcs partagés dans la carte des USA sont indiqués entre des points.

3.7. Utilisation de Topojson

Intérêts de la topologie:

- Précision dans le dessin des formes.
- Il permet le développement d'application traitant les zones élémentaires. Réduit la taille du fichier décrivant la région.

3.8. Librairie Leaflet

```
// initialize the map on the "map" div with a given center and zoom
var map = L.map('map', { center: [51.505, -0.09], zoom: 13 });
var mymap = L.map('mapid').setView([36.59412, 2.44322], 16);
```

Instancie un objet de carte en fonction de l'ID DOM d'un élément avec des options de carte.

```
L.tileLayer('http://{s}.tile.osm.org/{z}/{x}/{y}.png', {attribution: '&copy; <a
href="http://osm.org/copyright">OpenStreetMap</a> contributors'
    }).addTo(mymap);
```

Utilisé pour charger et afficher les couches de tuiles sur la carte
en utilisant les serveurs indiqués.

3.8. Librairie Leaflet

Ajouter un marqueur (icône par défaut), un cercle à la carte.

```
var marker = L.marker([36.59412, 2.44322]).addTo(mymap);

var circle = L.circle([36.59412, 2.44322], {
    color: 'red',
    fillColor: '#f03',
    fillOpacity: 0.5,
    radius: 10
}).addTo(mymap);
```

3.8. Librairie Leaflet

Afficher à l'aide de L.popup().

```
var popup = L.popup();

        function onMapClick(e) {
        popup
            .setLatLng(e.latlng)
            .setContent("Coordonnées long,lat : " + e.latlng.toString())
            .openOn(mymap);
          }
mymap.on('click', onMapClick);
```

3.8. Librairie Leaflet

Ajouter un icône autre que l'icone par défaut

```
var greenIcon = L.icon({
    iconUrl: 'leaf-green.png',
    shadowUrl: 'leaf-shadow.png',
    iconSize:     [38, 95], // size of the icon
    shadowSize:   [50, 64], // size of the shadow
    iconAnchor:   [22, 94], // point of the icon which will correspond to marker's location (log,lat)
    shadowAnchor: [4, 62],  // the same for the shadow
    popupAnchor:  [-3, -76] // point from which the popup should open relative to the iconAnchor
    });
```

Chapitre 3: Visualisation de données Géospatiales

3.8. Librairie Leaflet

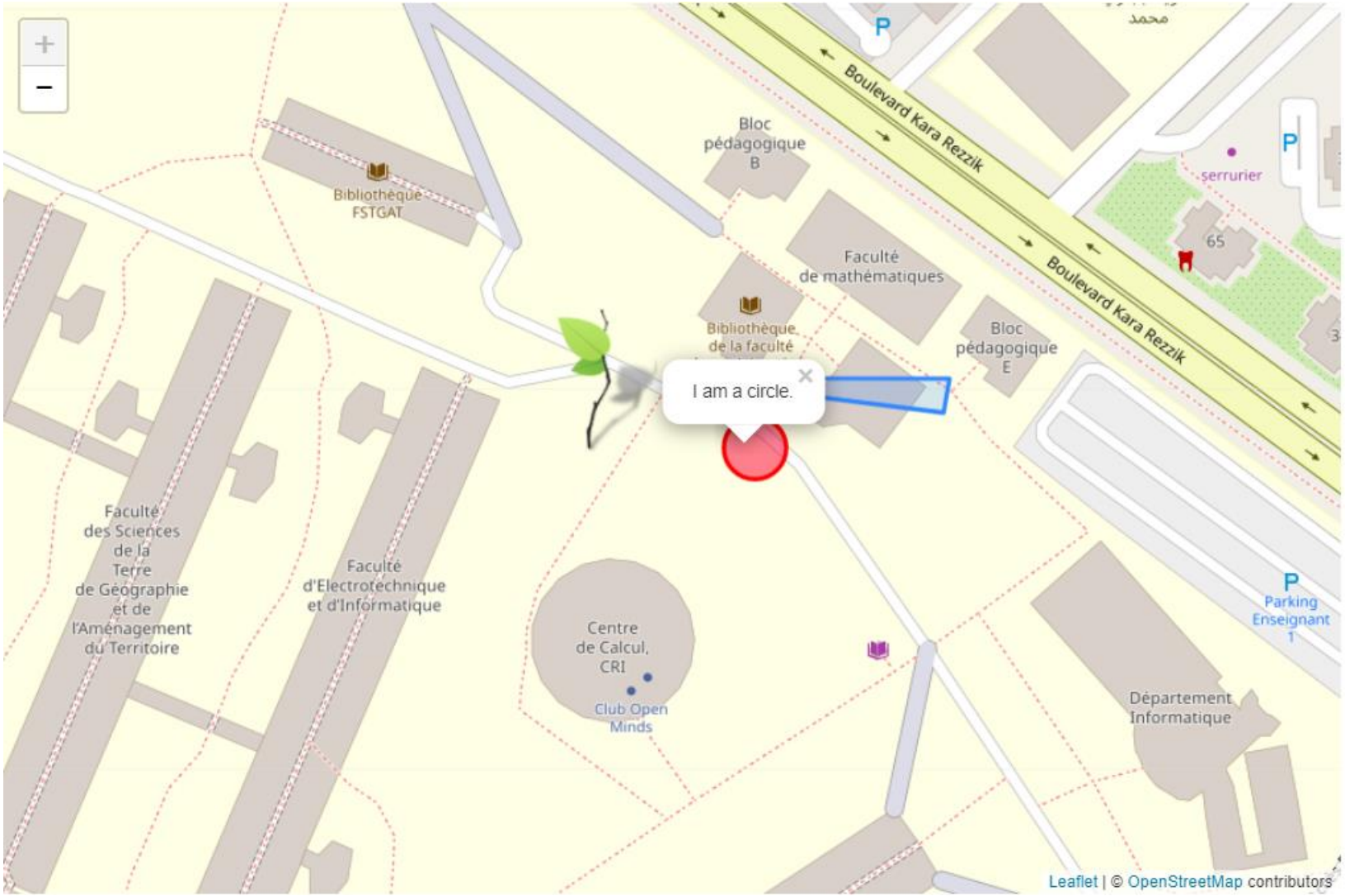
Affichage de message avec popup()

```
marker.bindPopup("<b>Vers la FEI</b><br>Bonne route.").openPopup();
circle.bindPopup("I am a circle.");
polygon.bindPopup("I am a polygon.");
```

Dessin de figures (polygone)

```
var polygon = L.polygon([
    [36.7168, 3.18428],
    [36.7169, 3.1833],
    [36.7169, 3.1843]
]).addTo(mymap);
```

Chapitre 3: Visualisation de données Géospatiales

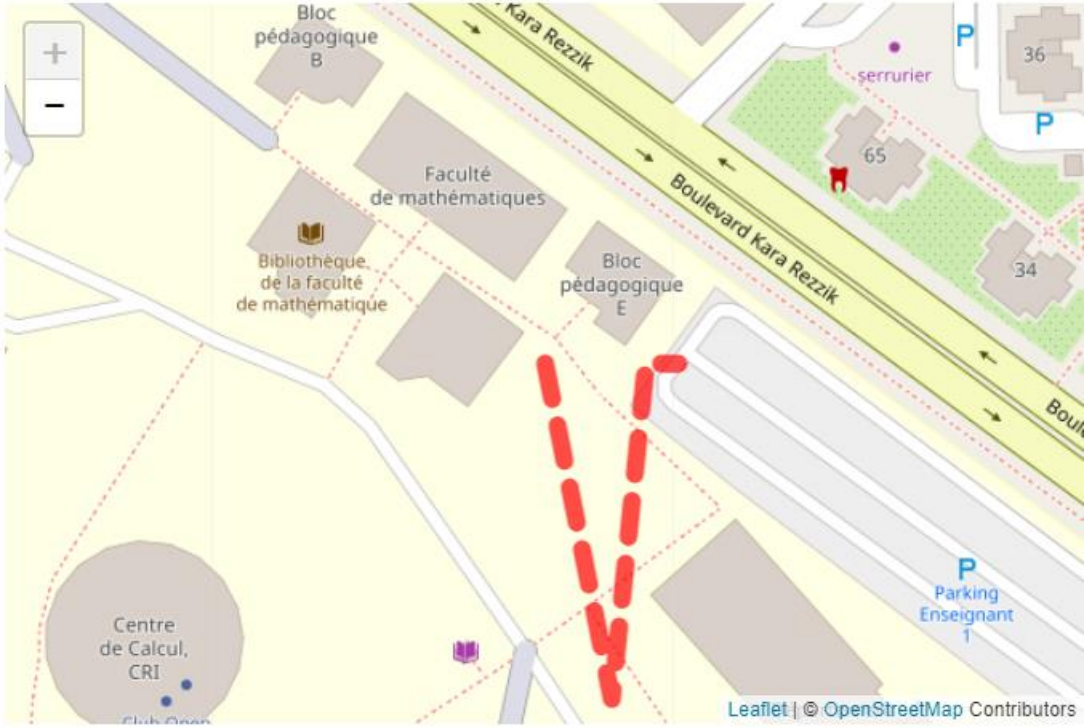


3.8. Librairie Leaflet

Ajout de ligne hachurée pour indiquer un chemin

```
var polyline = L.polyline([
        [36.7168, 3.18428],
        [36.7164, 3.18438],
        [36.7160, 3.18448],
        [36.7168, 3.18458],
        [36.7168, 3.18468]          ],
        { color: 'red',
           weight: 10,
           opacity: .7,
           dashArray: '20,15',
           lineJoin: 'round‘ }
        ). addTo(map);
```

3.8. Librairie Leaflet



3.8. Librairie Leaflet

Ajout du Marqueur standard

```
marker = new L.marker([36.7157, 3.1849]) // LATITUDE, LONGITUDE
                                         .bindPopup("test")
                                         .addTo(map);
```

3.8. Librairie Leaflet