Übung Open Data:

Geomapping

Termin 9, 23. April 2015

Dr. Matthias Stürmer und Prof. Dr. Thomas Myrach

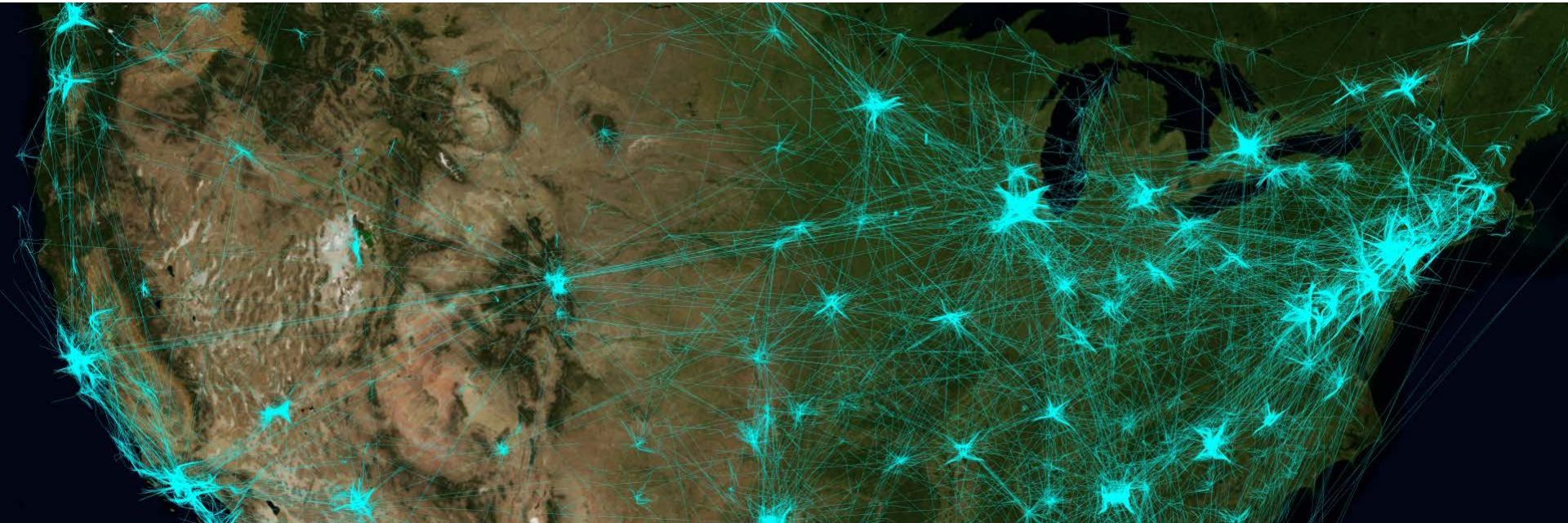
Universität Bern, Institut für Wirtschaftsinformatik
Abteilung Informationsmanagement
Forschungsstelle Digitale Nachhaltigkeit

Zwischenpräsentation

- **Donnerstag, 30. April 2015**
  13:15h bis 15:00h an der Engehaldenstrasse 8, Raum 001
- **Demo aktueller Stand der Open Data App auf IWI Sandbox**
  http://sandbox.iwi.unibe.ch
- **Obligatorische Teilnahme** für alle Studierenden-Teams
- **Vorstellung der App enthält:**
  1. Was für Daten wurden verwendet? (Data Coach, andere Quellen)
  2. Wie war das Vorgehen bis jetzt? (Daten transformiert, programmiert…)
  3. Aktueller Stand der App, Demonstration der Visualisierung
- **Anmeldung (oder begründete Abmeldung) :**
  Bis HEUTE Donnerstag, 23. April 2015 per Email an Mirjam Läderach:
  mirjam.laederach@iwi.unibe.ch

Agenda

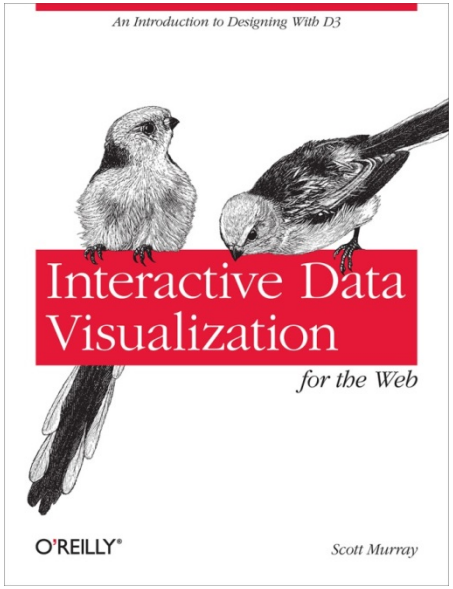
1. **GeoJSON**
2. Paths and Projections
3. Choropleth



Interactive Data Visualization for the Web

- **Chapter 12. Geomapping:**

- http://chimera.labs.oreilly.com/books/1230000000345/ch12.html

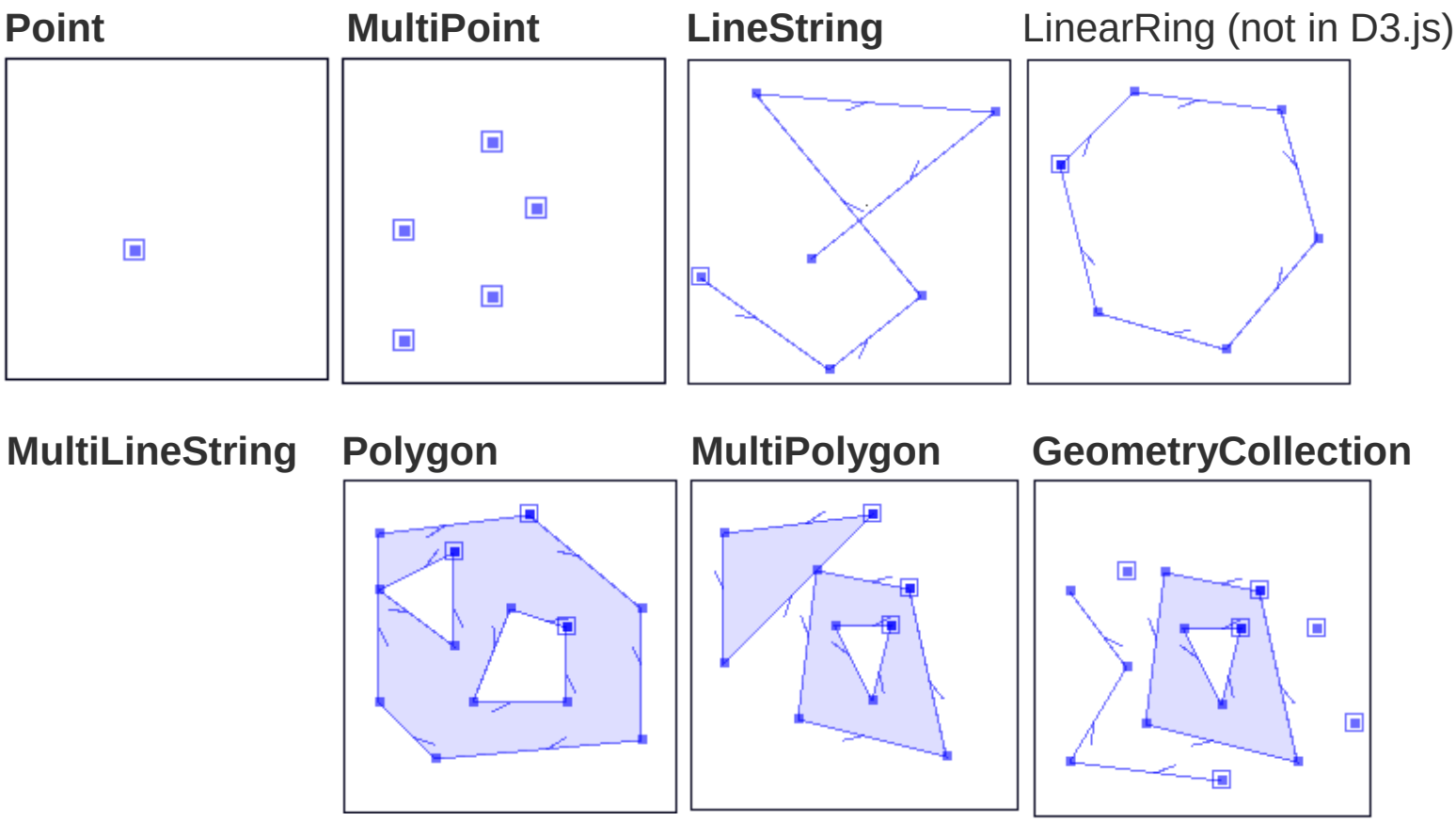


GeoJSON

GeoJSON: JSON-based standard for encoding geodata for web applications (very specific use of JSON, **one giant JavaScript object)**

```
{
  "type": "FeatureCollection",
  "features": [
     {
       "type": "Feature",
       "id": "01",
       "properties": { "name": "Alabama" },
       "geometry": {
         "type": "Polygon",
         "coordinates": [[[-87.359296,35.00118],
           [-85.606675,34.984749],[-85.431413,34.124869],
           [-85.184951,32.859696],[-85.069935,32.580372],
           [-84.960397,32.421541],[-85.004212,32.322956],
           [-84.889196,32.262709],[-85.058981,32.13674] …
          ]]
       }
     }
```

GeoJSON geometry types



Source: http://www.vividsolutions.com/jts/discussion.htm

GeoJSON geometry objects

- The GeoJSON object may have any number of **members** (name/value pairs).
- The GeoJSON object must have a member with the name **`type`.**
- The value of the **`type`** member must be one of: **`Point, MultiPoint, LineString`** etc.
- The **`coordinates`** is composed of
  - **one position** (in the case of a Point geometry)
  - **an array of positions** (LineString or MultiPoint geometries)
  - **an array of arrays of positions** (Polygons, MultiLineStrings)
  - **a multidimensional array of positions** (MultiPolygon)

```
{ "type": "Point",
  "coordinates":
  [100.0, 0.0]
}

{ "type":
  "LineString",
  "coordinates":
  [ [100.0, 0.0],
  [101.0, 1.0] ]
}

{ "type": "Polygon",
  "coordinates":
  [[ [100.0, 0.0],
     [101.0, 0.0],
     [101.0, 1.0],
     [100.0, 1.0],
     [100.0, 0.0] ]]
}
```

Source: http://geojson.org/geson-spec.html

GeoJSON feature objects

- A GeoJSON object with the type **`Feature`** is a **feature object.**
- A feature object must have a member with the name **`properties`**. The value of the properties member is an **object** (any JSON object or a JSON null value).
- A feature object must have a member with the name **`geometry`**. The value of the geometry member is a **geometry object** as defined above or a JSON null value.
- If a feature has a commonly used **identifier,** that identifier should be included as a member of the feature object with the name **`id`.**

```
{
  "type": "Feature",
  "id": "01",
  "properties":
  {
    "name": "Alabama“
  },
  "geometry": {
    "type": "Polygon",
    "coordinates":
    [[
      [-87.35,35.00],
      [-85.60,34.98],
      [-85.43,34.12],
    ]]
  }
}
```

Source: http://geojson.org/geson-spec.html

GeoJSON feature collection objects

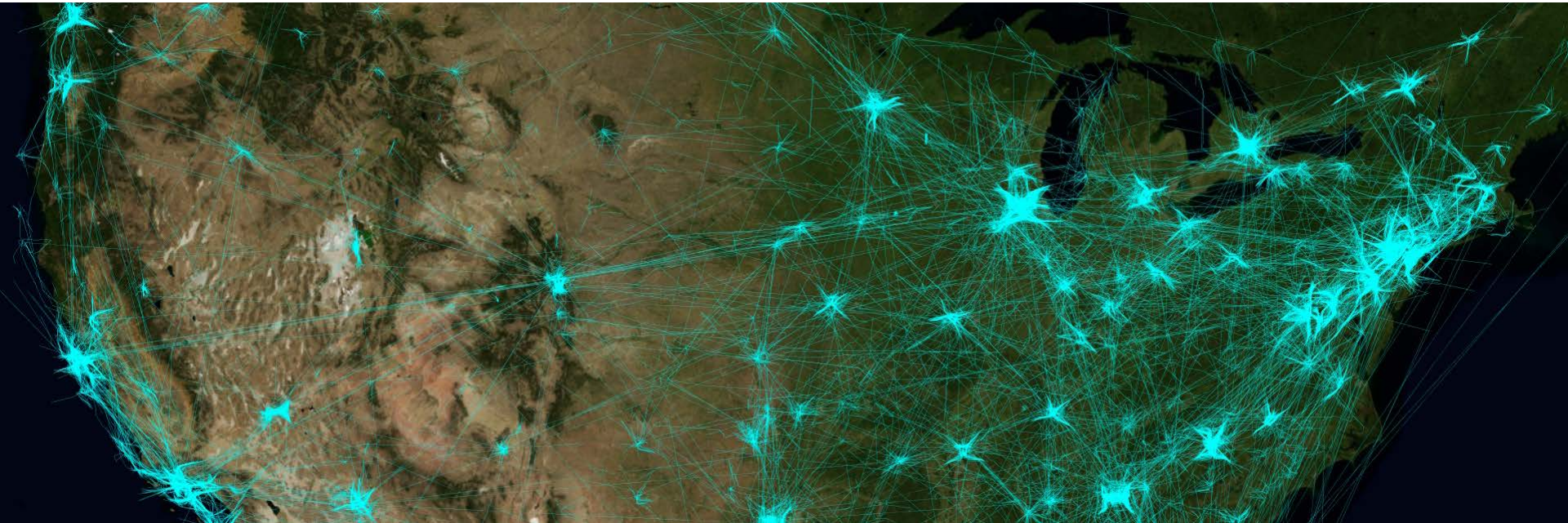
- A GeoJSON object with the type **`FeatureCollection`** is a **feature collection object.**
- An object of type **`FeatureCollection`** must have a member with the name **`features`**. The value of **`features`** is an **array.** Each element in the array is a **feature object** as defined above.


```
{"type":"FeatureCollection","features":[
{"type":"Feature","id":"01","properties":
{"name":"Alabama"},"geometry":
{"type":"Polygon","coordinates":
[[[-87.359296,35.00118],[-85.606675,34.984749],
[-85.431413,34.124869],[-85.184951,32.859696],
[-85.069935,32.580372],[-84.960397,32.421541],
[-85.004212,32.322956],[-84.889196,32.262709],
[-85.058981,32.13674],[-85.053504,32.01077],
[-85.141136,31.840985],[-85.042551,31.539753],
[-85.113751,31.27686],[-85.004212,31.003013],
[-85.497137,30.997536],[-87.600282,30.997536],
[-87.633143,30.86609],[-87.408589,30.674397],
[-87.446927,30.510088],[-87.37025,30.427934],
[-87.518128,30.280057],[-87.655051,30.247195],
[-87.90699,30.411504],[-87.934375,30.657966],
```


Source: http://geojson.org/geson-spec.html http://geojsonlint.com

Agenda

1. GeoJSON
2. **Paths and Projections**
3. Choropleth



Path generation

Path generator function for translation of GeoJSON into **SVG path:**

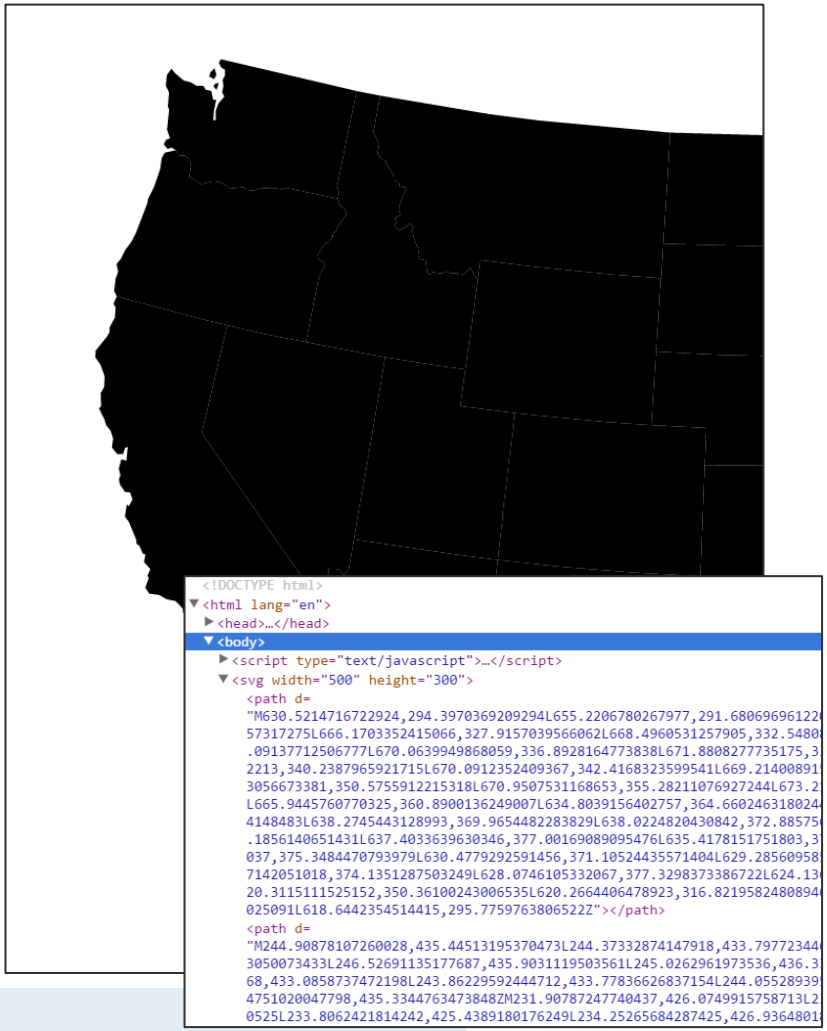
```
var path = d3.geo.path();
```

d3.json() takes **two arguments.** First, it takes a **string** pointing to the path of the file to load in. Second, it takes a **callback function** that is fired when the JSON file has been loaded and parsed.

```
d3.json("us-states.json", function(json) {
    svg.selectAll("path")
        .data(json.features)
        .enter()
        .append("path")
        .attr("d", path);
});
```

Example 1: US States

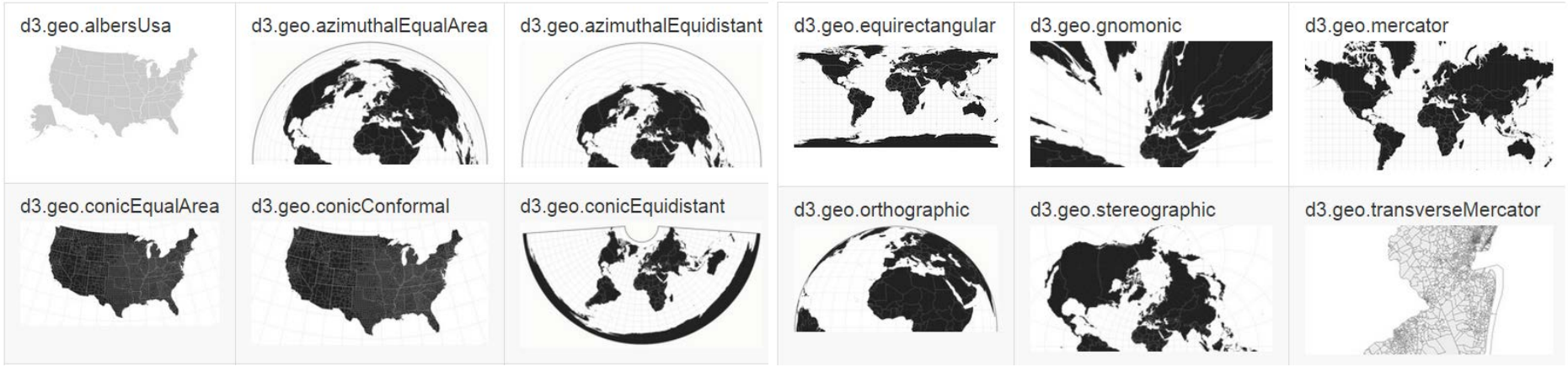
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>D3: Loading GeoJSON data and generating SVG paths</title>
    <script type="text/javascript" src="d3.v3.js"></script>
    <style type="text/css">
      /* No style rules here yet */
    </style>
  </head>
  <body>
    <script type="text/javascript">

      //Define default path generator
      var path = d3.geo.path();

      //Create SVG element
      var svg = d3.select("body")
            .append("svg")
            .attr("width", 500)
            .attr("height", 300);

      //Load in GeoJSON data
      d3.json("us-states.json", function(json) {

        //Bind data and create one path per GeoJSON feature
        svg.selectAll("path")
           .data(json.features)
           .enter()
           .append("path")
           .attr("d", path);

      });

    </script>
  </body>
</html>
```



Projections

A projection is an algorithm of compromise; it is the method by which **3D space is "projected" onto a 2D plane.**

```
var projection = d3.geo.albersUsa()
                       .translate([w/2, h/2]);
var path = d3.geo.path()
                    .projection(projection);
```

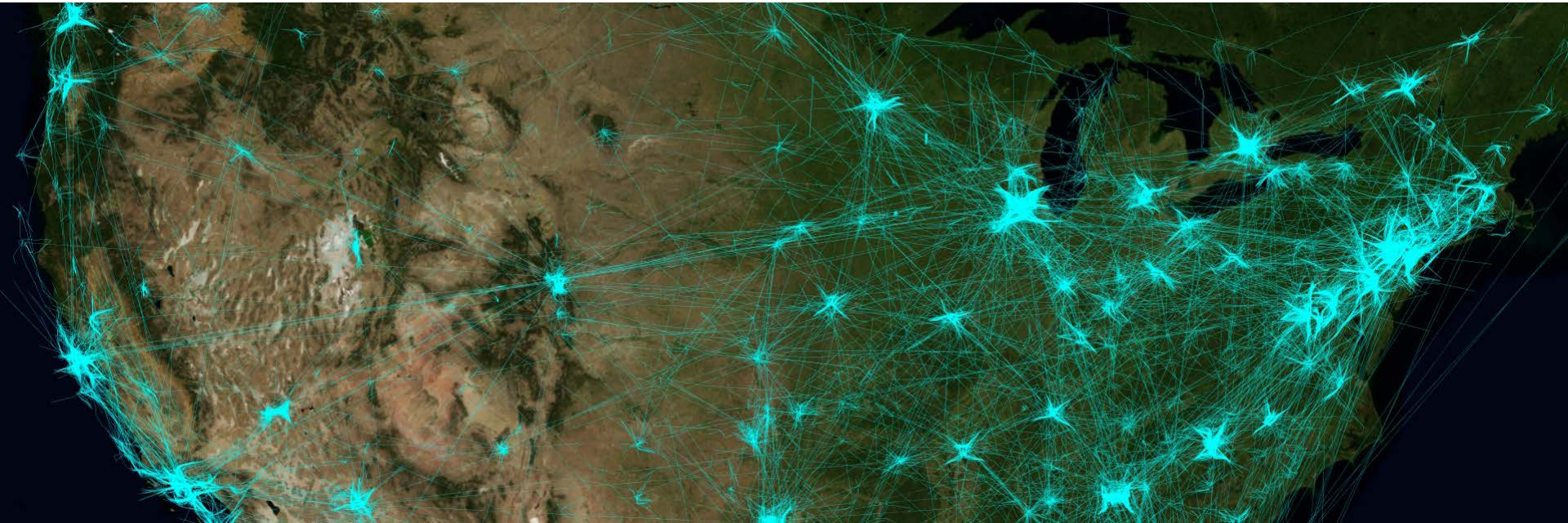


Source: https://github.com/mbostock/d3/wiki/Geo-Projections

UNIVERSITÄT BERN

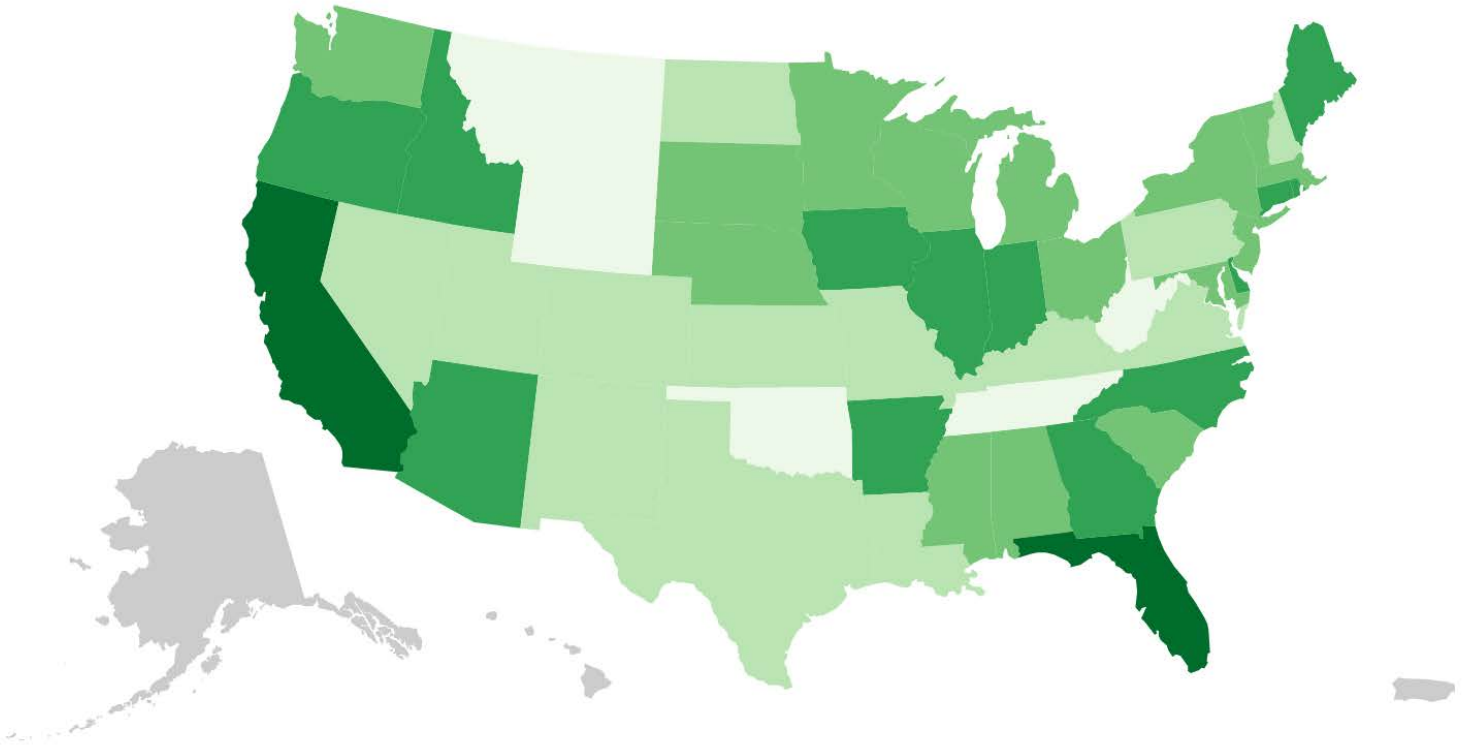
Agenda

1. GeoJSON
2. **Paths and Projections**
3. Choropleth



Choropleth

Choropleth: geomap with areas filled in with different values (light or dark) or colors to reflect associated data values.



Color scale

Create a **scale** that can take data values as input, and will return colors:

```
var color = d3.scale.quantize()
  .range(["rgb(237,248,233)", "rgb(186,228,179)",
          "rgb(116,196,118)", "rgb(49,163,84)","rgb(0,109,44)"]);
```

A **`quantize`** scale functions as a linear scale, but it outputs values from within a discrete range. These output values could be **numbers, colors** (as we've done here), or anything else you like. This is useful for sorting values into **"buckets."** In this case, we're using five buckets, but there could be as many as you like.

colorbrewer.js: collection of perceptually optimized colors

Source: https://github.com/mbostock/d3/tree/master/lib/colorbrewer

Load CSV data and set input domain

Load in the data with **`d3.csv()`**, the set the **`color`** quantize scale's input domain:

```
//Load in agriculture data
d3.csv("us-ag-productivity-2004.csv", function(data) {

    //Set input domain for color scale
    color.domain([
        d3.min(data, function(d) { return d.value; }),
        d3.max(data, function(d) { return d.value; })
    ]);
```

```
1  state,value
2  Alabama,1.1791
3  Arkansas,1.3705
4  Arizona,1.3847
5  California,1.7979
6  Colorado,1.0325
7  Connecticut,1.3209
8  Delaware,1.4345
9  Florida,1.6304
10 Georgia,1.3891
11 Iowa,1.5297
12 Idaho,1.4285
13 Illinois,1.5297
14 Indiana,1.4220
```

Data merging

Next, we load in the JSON geodata, as before. But what's new here is I want to **merge** the agricultural data **into** the GeoJSON. Why? Because we can only bind one set of data to elements at a time.

We definitely need the GeoJSON, from which the paths are generated, but we also need the **new agricultural data.** So if we can smush them into a single, monster **array,** then we can **bind** them to the new path elements all at the same time.

```
//Load in GeoJSON data
d3.json("us-states.json", function(json) {

  //Merge the ag. data and GeoJSON
  //Loop through once for each ag. data value
  for (var i = 0; i < data.length; i++) {

    //Grab state name
    var dataState = data[i].state;

    //Grab data value, and convert from string to float
    var dataValue = parseFloat(data[i].value);

    //Find the corresponding state inside the GeoJSON
    for (var j = 0; j < json.features.length; j++) {

      var jsonState = json.features[j].properties.name;

      if (dataState == jsonState) {

        //Copy the data value into the JSON
        json.features[j].properties.value = dataValue;

        //Stop looking through the JSON
        break;
      }
    }
  }
```

Path creation

Lastly, we create the **`paths`** just as before, only we make our **`style()`** value dynamic:

```
//Bind data and create one path per GeoJSON feature
svg.selectAll("path")
     .data(json.features)
     .enter()
     .append("path")
     .attr("d", path)
     .style("fill", function(d) {
         //Get data value
         var value = d.properties.value;

         if (value) {
           //If value exists…
           return color(value);
         } else {
           //If value is undefined…
           return "#ccc";
         }
   });
```