Übung Open Data:

Geomapping

Termin 10, 12. Mai 2016

Dr. Matthias Stürmer und Prof. Dr. Thomas Myrach

Forschungsstelle Digitale Nachhaltigkeit
Institut für Wirtschaftsinformatik
Universität Bern

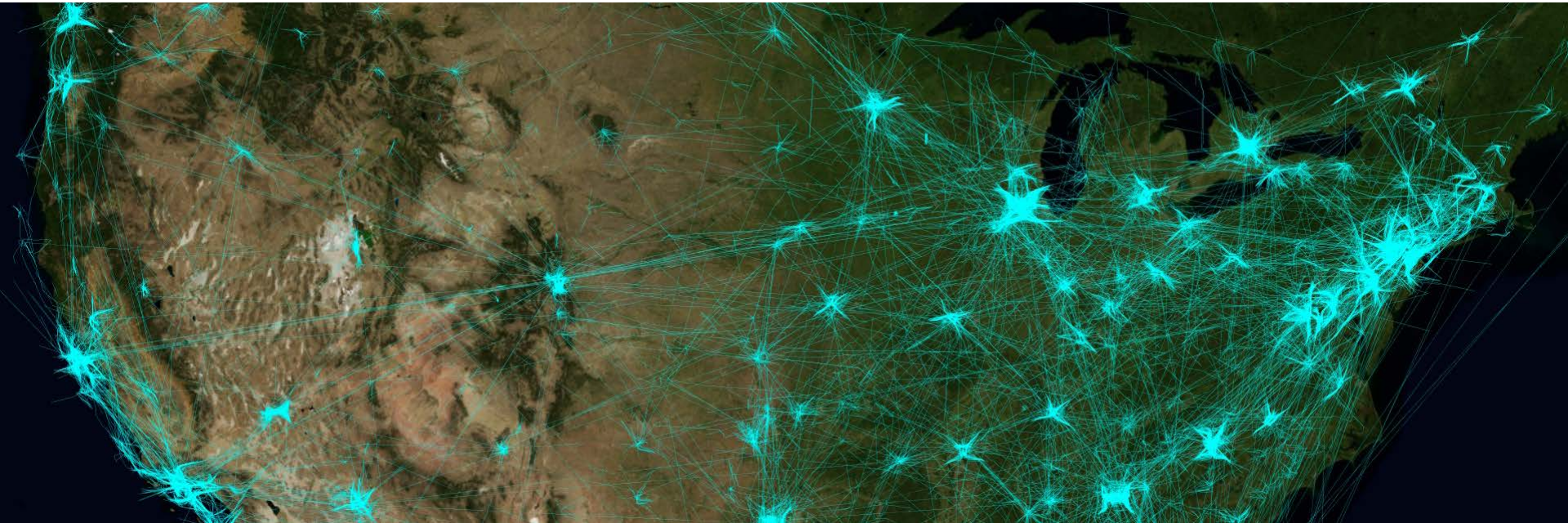
UNIVERSITÄT BERN

Terminübersicht Übungen

- 25.02.2016: Informationen zu den Übungen, App-Demos & Einführung in Tools
- 03.03.2016: Einführung Web-Programmierung
- 10.03.2016: Open Data Speed Dating
- 17.03.2016: Einführung D3.js & Daten einbinden in D3.js
- 24.03.2016: Anpassen von bestehenden Apps & Bibliotheken die D3.js verwenden
- 31.03.2016: Osterferien
- 07.04.2016: Daten visualisieren & Layouts
- 14.04.2016: Skalen und Achsen & Responsive Design
- 21.04.2016: User Experience, Usability Patterns
- 28.04.2016: Zwischenpräsentation & Datenaktualisierung und Transitionen
- 05.05.2016: Auffahrt
- **12.05.2016: ~~Interactivity &~~ Geomapping & Programming Coaching**
- 19.05.2016: 3D Web-Programmierung mit Three.js & Programming Coaching
- 26.05.2016: Abschlusspräsentationen
- 02.06.2016: frei

Agenda

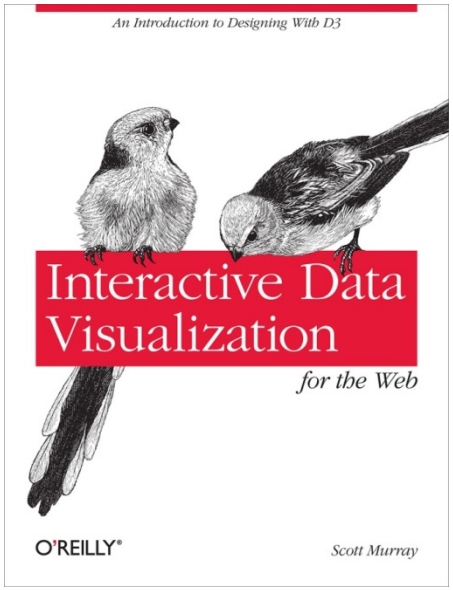
1. **GeoJSON and TopoJSON**
2. Paths and Projections
3. Choropleth
4. Adding Points



Interactive Data Visualization for the Web

- **Chapter 12. Geomapping:**

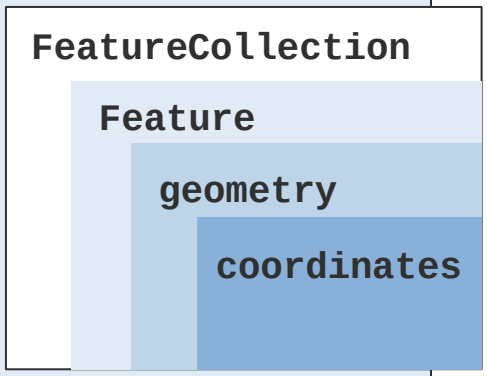
- http://chimera.labs.oreilly.com/books/1230000000345/ch12.html



GeoJSON

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

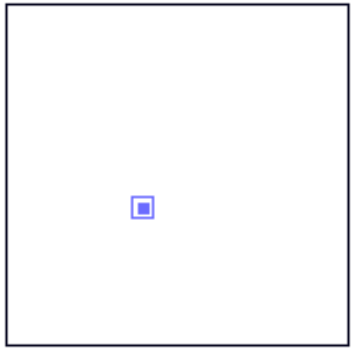
```
{
  "type": "FeatureCollection",
  "features": [
     {
       "type": "Feature",
       "id": "01",
       "properties": { "name": "Alabama" },
       "geometry": {
         "type": "Polygon",
         "coordinates": [[[-87.359296,35.00118],
           [-85.606675,34.984749],[-85.431413,34.124869],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



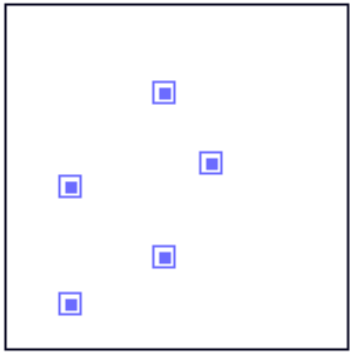
Source: https://raw.githubusercontent.com/alignedleft/d3-book/master/chapter_12/us-states.json

GeoJSON geometry types

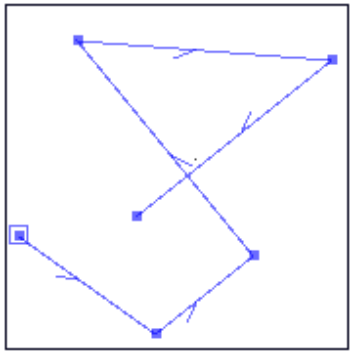
Point



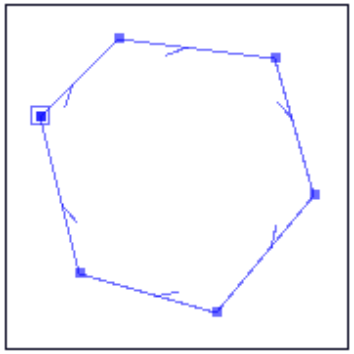
MultiPoint



LineString

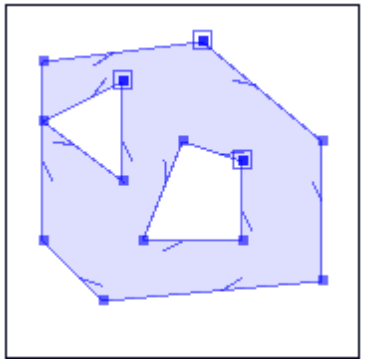


LinearRing (not in D3.js)

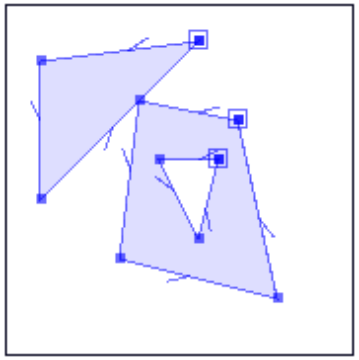


MultiLineString

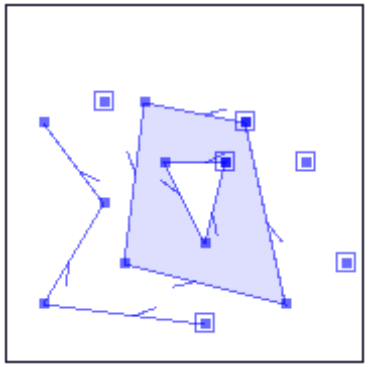
Polygon



MultiPolygon



GeometryCollection



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.

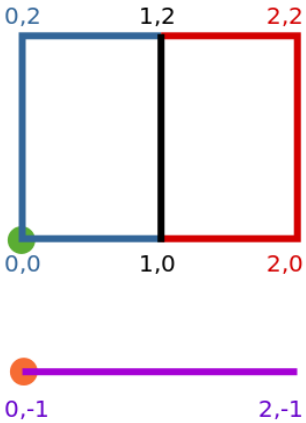

```
{"type":"FeatureCollection","features":[
{"type":"Feature","id":"01","properties":
{"name":"Alabama"},"geometry":
{"type":"Polygon","coordinates":
[[[-87.359296,35.00118],[-85.606675,34.984749],
[-85.431413,34.124869],[-85.184951,32.859696],
[-85.069935,32.580372],[-84.960397,32.421541],
[-85.004212,32.322956],[-84.889196,32.262709],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

TopoJSON

- TopoJSON is an extension of GeoJSON that encodes topology.
- Rather than representing geometries discretely, geometries in TopoJSON files are stitched together from shared line segments called arcs.
- **Arcs are sequences of points, while line strings and polygons are defined as sequences of arcs.**
- Each arc is defined only once, but can be referenced several times by different shapes, thus reducing redundancy and **decreasing the file size.**

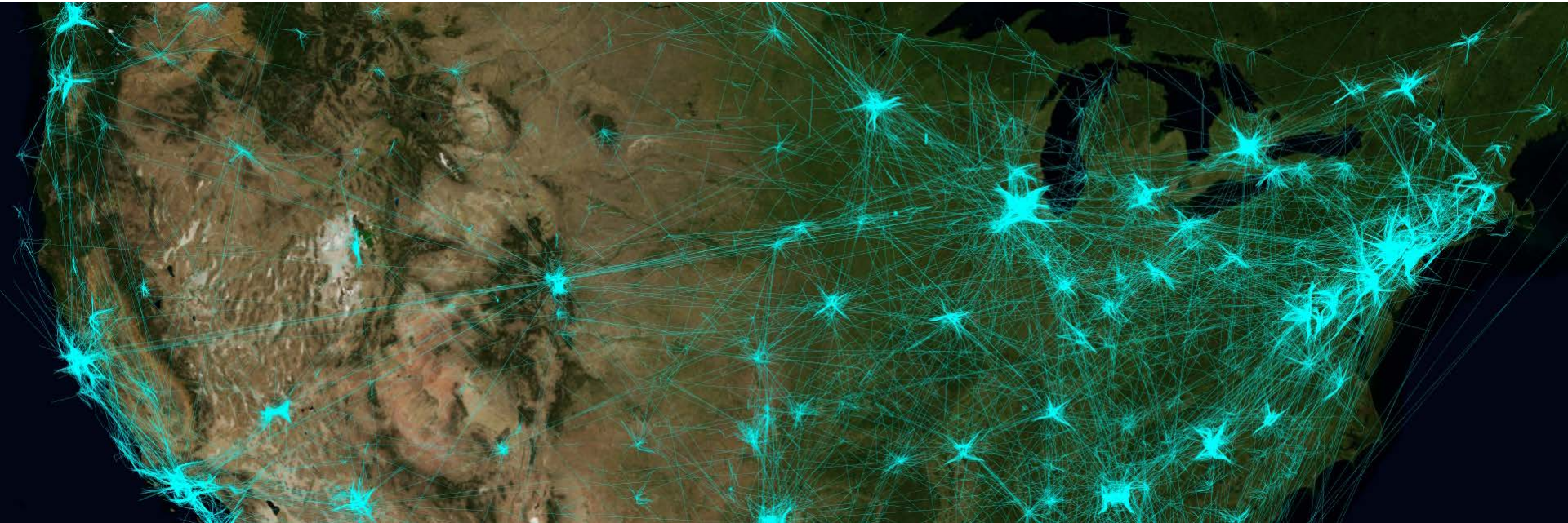
```
Origine_Point
   "coordinates": [0,0]

Under_Point
   "coordinates": [0,-1]

Under_LineString
   "arcs": [3]

Left_Polygon
   "arcs": [[0,1]]

Right_Polygon
   "arcs": [[2,-1]]

"arcs":[
   [[1,2],[0,-2]],
   [[1,0],[-1,0],[0,2][1,0]],
   [[1,2],[1,0],[0,-2],[-1,0]],
   [[0,-1],[2,0]]
]
```



Source: https://en.wikipedia.org/wiki/GeoJSON#TopoJSON

Agenda

1. GeoJSON and TopoJSON
2. **Paths and Projections**
3. Choropleth
4. Adding Points



Path generation

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

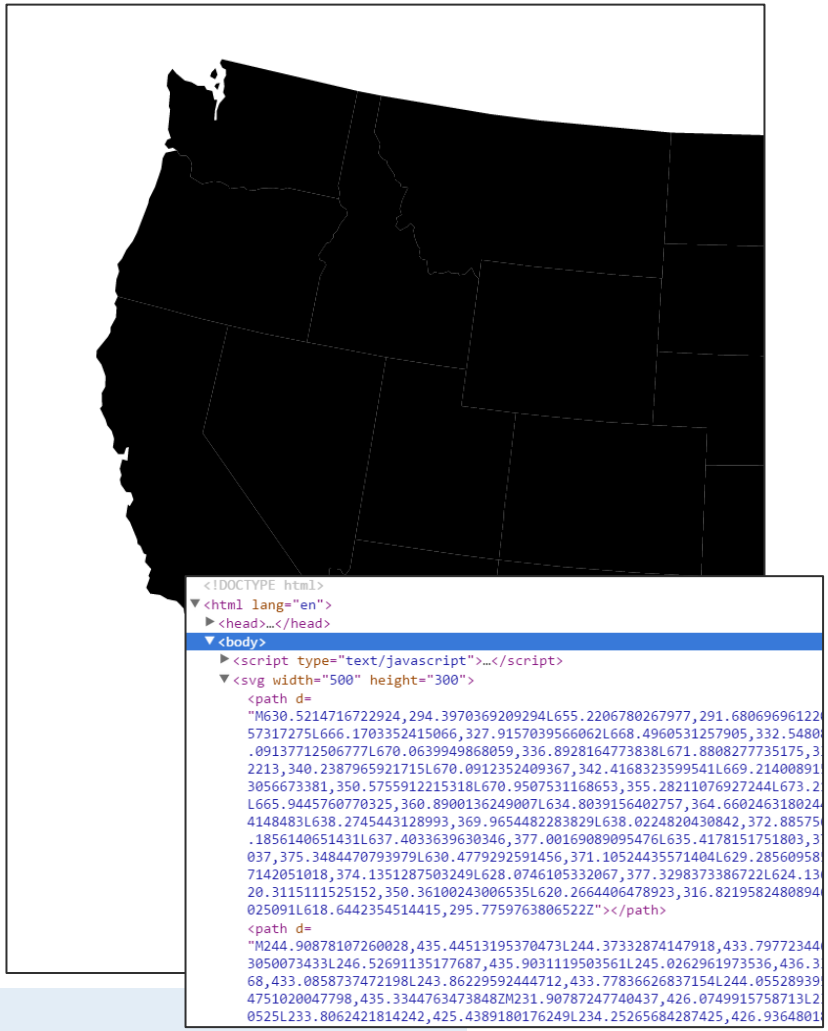
```
var path = d3.geo.path();
```

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

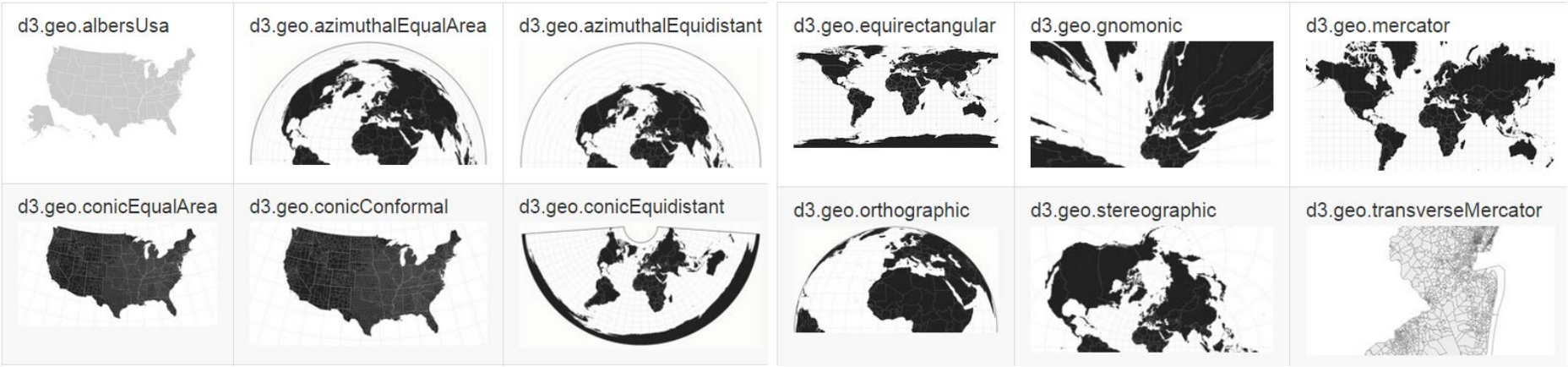
```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>D3: Loading GeoJSON data and generating SVG paths</title>
    <script type="text/javascript" src="d3.v3.js"></script>
    <style type="text/css">
      /* No style rules here yet */
    </style>
  </head>
  <body>
  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

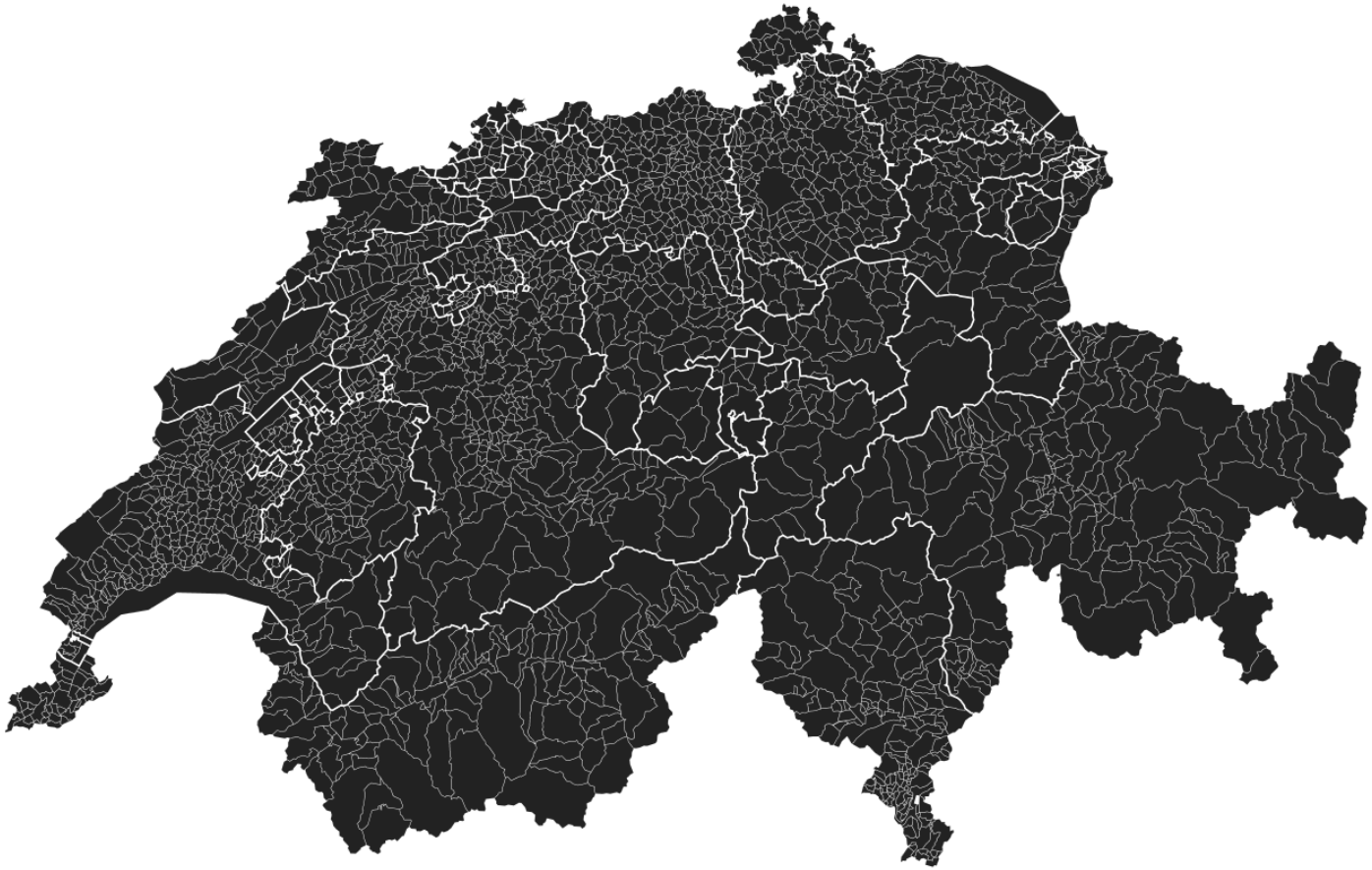


Source: https://github.com/mbostock/d3/wiki/Geo-Projections

Swiss Cantons

Source: http://bl.ocks.org/mbostock/4207744

Swiss Cantons and Municipalities



Source: http://bl.ocks.org/herrstucki/4327678

Swiss Maps

Branch: master | New pull request | New file | Find file | HTTPS | https://github.com/inter | Download ZIP

herrstucki Use Git LFS for shapefiles — Latest commit ebe8cd5 on 13 Oct 2015

src	Add 2015	a year ago
.gitattributes	Use Git LFS for shapefiles	7 months ago
.gitignore	Add SVG output	8 months ago
LICENSE	Update license for 2014 data	2 years ago
LICENSE-GEODATA	Update license for 2014 data	2 years ago
Makefile	Deprecate overly specific clean tasks	7 months ago
README.md	Add note about cleaning before re-making with different years	7 months ago
package.json	3.3.1	7 months ago

README.md

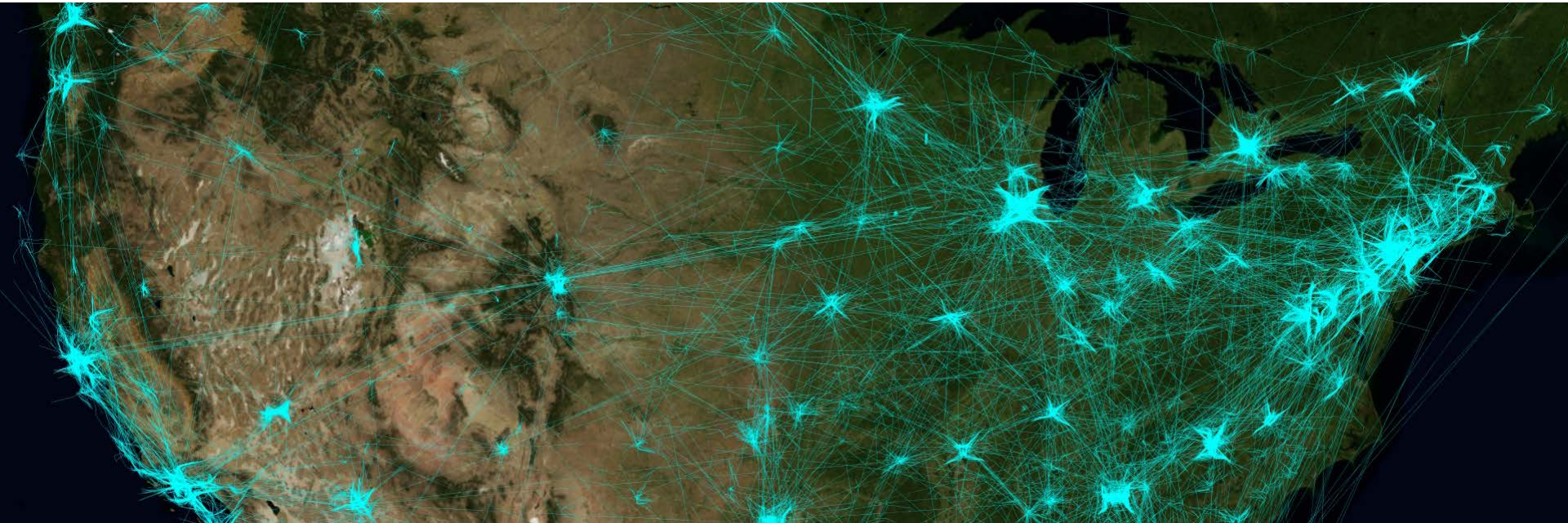
Swiss Maps

This repository provides a mechanism to generate TopoJSON from publicly available (but difficult to access) swisstopo geodata.

Source: https://github.com/interactivethings/swiss-maps

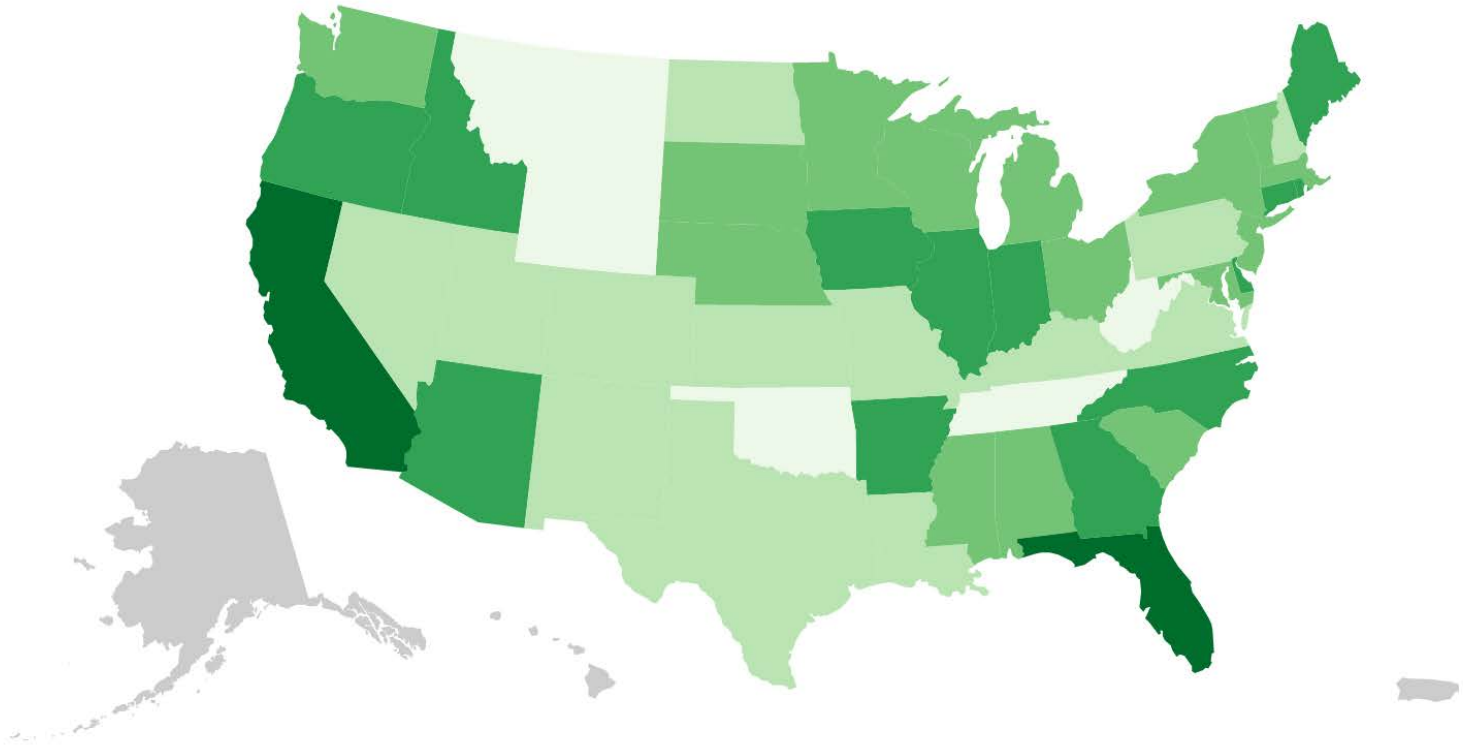
Agenda

1. GeoJSON and TopoJSON
2. Paths and Projections
3. **Choropleth**
4. Adding Points



Choropleth

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



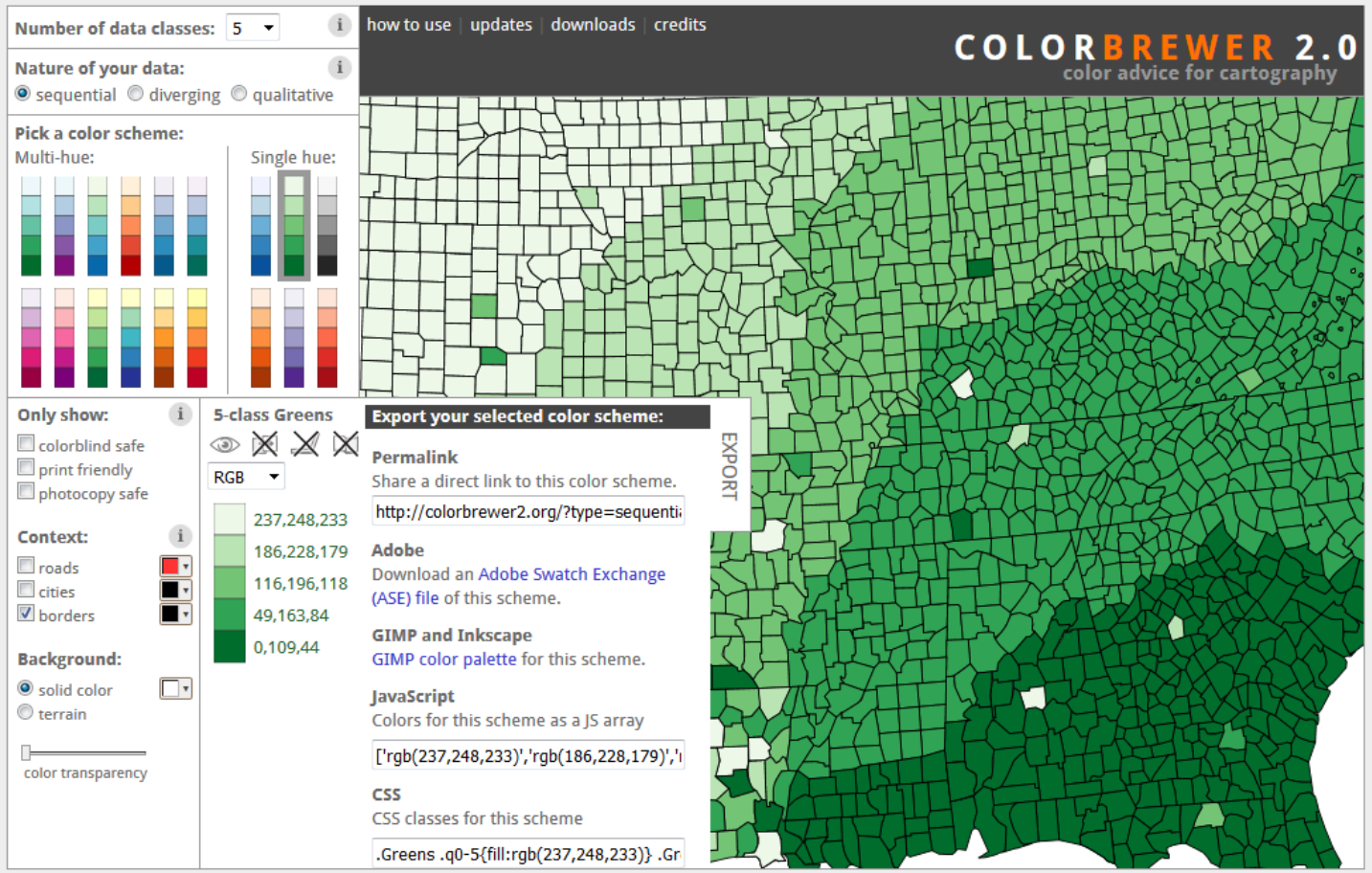
Color scale

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



Source: http://colorbrewer2.org

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; })
    ]);
```

```
state,value
Alabama,1.1791
Arkansas,1.3705
Arizona,1.3847
California,1.7979
Colorado,1.0325
Connecticut,1.3209
Delaware,1.4345
Florida,1.6304
Georgia,1.3891
Iowa,1.5297
Idaho,1.4285
Illinois,1.5297
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.

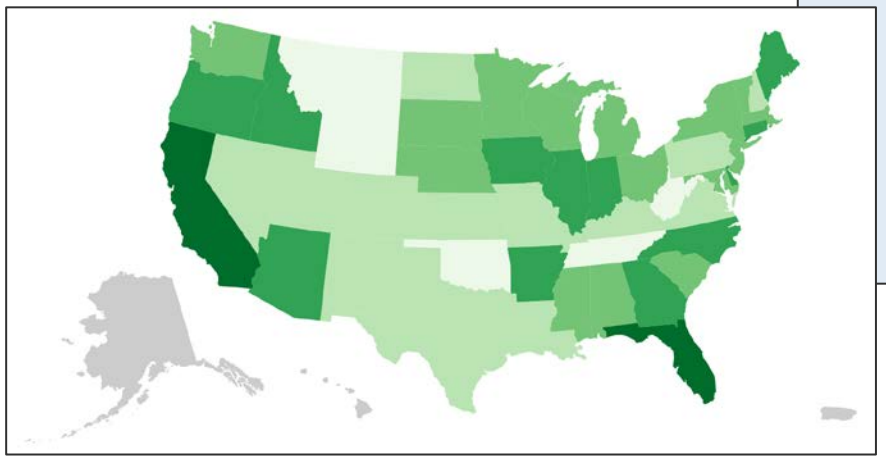
We definitely need the GeoJSON, from which the paths are generated, but we also need the **new agricultural data.** So if we can smush them into a single, monster **array,** then we can **bind** them to the new path elements all at the same time.

```
//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

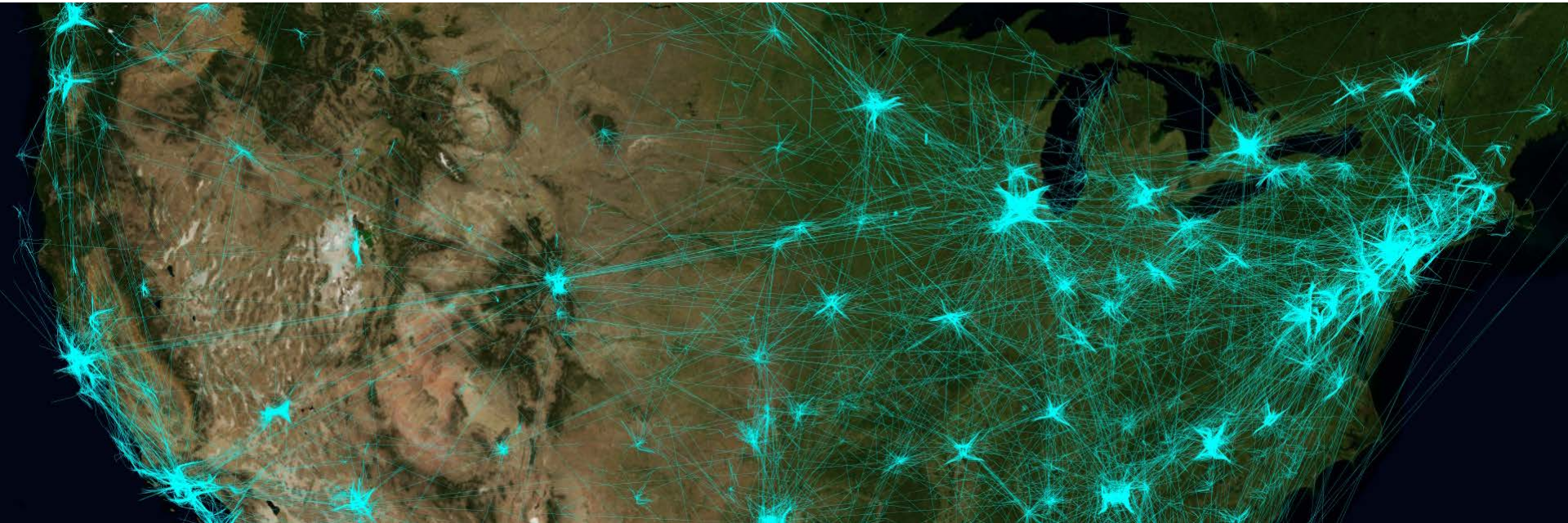
Lastly, we create the **`paths`** just as before, only we make our **`style()`** value dynamic:

```
//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";
         }
     });
```



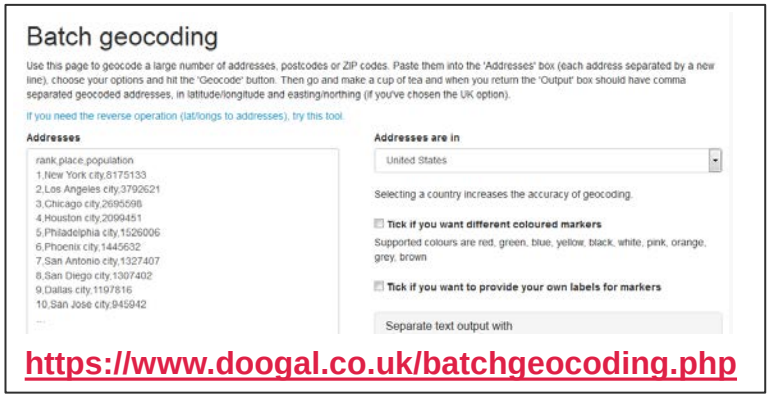
Agenda

1. GeoJSON and TopoJSON
2. Paths and Projections
3. Choropleth
4. **Adding Points**



Batch Geocoding

```
rank,place,population
1,New York city,8175133
2,Los Angeles city,3792621
3,Chicago city,2695598
4,Houston city,2099451
5,Philadelphia city,1526006
6,Phoenix city,1445632
7,San Antonio city,1327407
8,San Diego city,1307402
9,Dallas city,1197816
10,San Jose city,945942
…
```

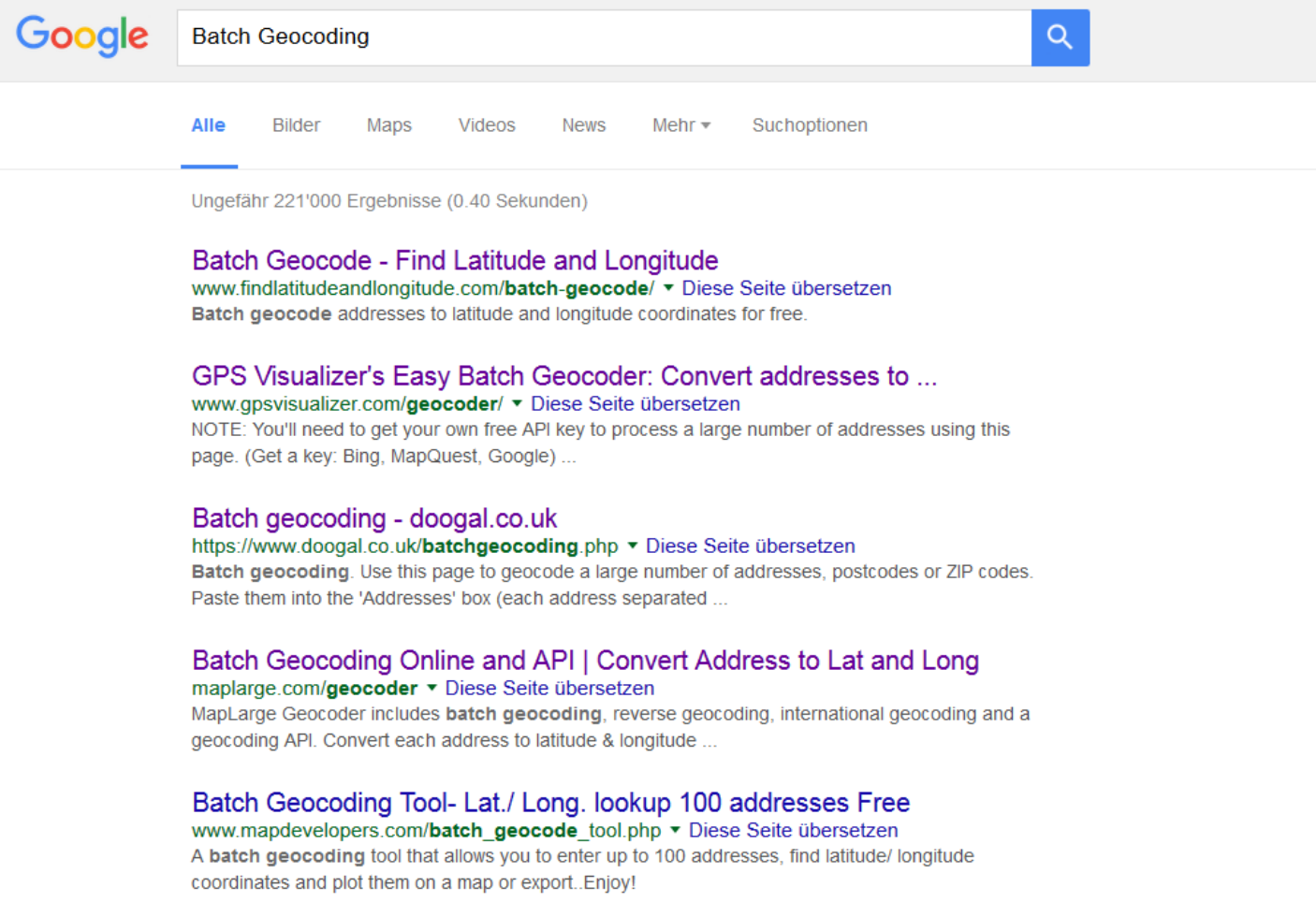


https://www.doogal.co.uk/batchgeocoding.php

```
rank,place,population,lat,lon
1,New York city,8175133,40.630586,-74.105678
2,Los Angeles city,3792621,34.057035,-118.34714
3,Chicago city,2695598,41.8788,-87.629
4,Houston city,2099451,29.623363,-95.395127
5,Philadelphia city,1526006,40.000965,-75.227046
6,Phoenix city,1445632,33.640014,-112.004334
7,San Antonio city,1327407,29.415944,-98.495048
8,San Diego city,1307402,32.733801,-117.193304
9,Dallas city,1197816,32.783797,-96.798247
10,San Jose city,945942,37.31471,-121.977034
…
```

Geocoding is the process of taking place names, looking them up on a map (or in a database, really), and returning precise lat/lon coordinates.

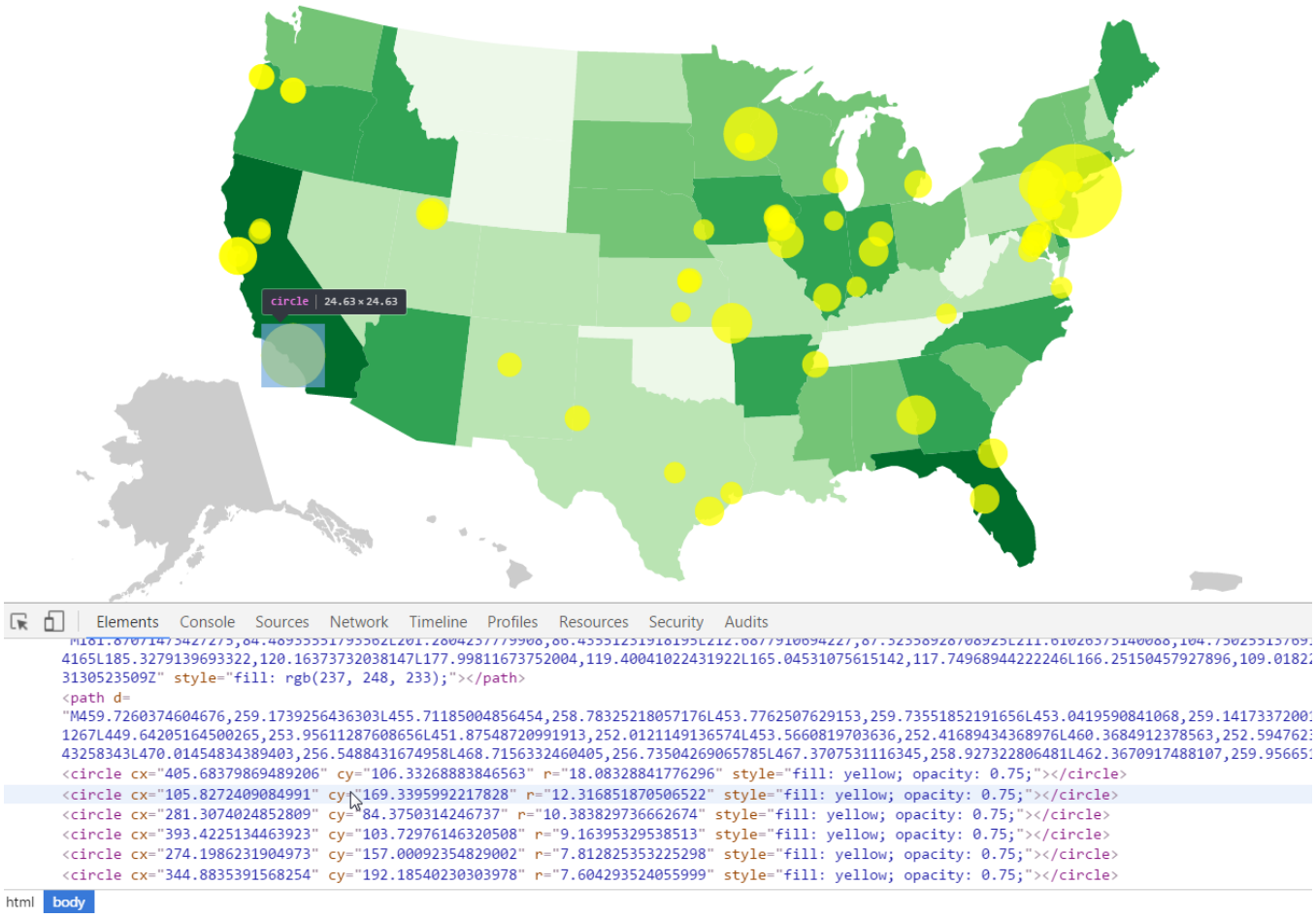
Batch Geocoding Tools



Adding points

In the callback function, we can specify how to create a **new circle element for each city,** and then position each circle according to the corresponding city's geo-coordinates:

```
//Load in cities data
d3.csv("us-cities.csv", function(data) {
  svg.selectAll("circle")
     .data(data)
     .enter()
     .append("circle")
     .attr("cx", function(d) {
        return projection([d.lon, d.lat])[0];
     })
     .attr("cy", function(d) {
        return projection([d.lon, d.lat])[1];
     })
     .attr("r", function(d) {
        return Math.sqrt(parseInt(d.population) * 0.00004);
     })
     .style("fill", "yellow")
     .style("opacity", 0.75);
});
```

The map projection takes a two-value array as input, with *longitude* first (remember, it's lon/lat, not lat/lon, in GeoJSON-ville). Then the projection returns a two-value array with x/y screen values. So, for cx, we use [0] to grab the *first* of those values, which is *x*. For cy, we use [1] to grab the *second* of those values, which is *y*.

Adding points