09. D3JS Geographies.

Geospatial coordinates are often used for weather or population data. D3.js gives us three tools for geographic data −

- **Paths** − They produce the final pixels.
- **Projections** − They turn sphere coordinates into Cartesian coordinates and
- **Streams** − They speed things up.

Before learning what geographies in *D3.js* are, we should understand the following two terms −

- **D3 Geo Path** and
- **Projections**

Let us discuss these two terms in detail.

9.1. D3 Geo Path

It is a geographic path generator. GeoJSON generates SVG path data string or renders the path to a Canvas. A Canvas is recommended for dynamic or interactive projections to improve performance. To generate a D3 Geo Path Data Generator, you can call the following function.

Example:

```
d3.geo.path()
```

Here, the d3.geo.path() path generator function allows us to select which Map Projection we want to use for the translation from Geo Coordinates to Cartesian Coordinates.

For example, if we want to show the map details of India, we can define a path as shown below.

Example:

```
var path = d3.geo.path()
svg.append("path")
   .attr("d", path(states))
```

9.2. Projections

Projections transform spherical polygonal geometry to planar polygonal geometry. D3 provides the following projection implementations.

- **Azimuthal** − Azimuthal projections project the sphere directly onto a plane.
- **Composite** − Composite consists of several projections that are composed into a single display.
- **Conic** − Projects the sphere onto a cone and then unroll the cone onto the plane.
- **Cylindrical** − Cylindrical projections project the sphere onto a containing cylinder, and then unroll the cylinder onto the plane.

To create a new projection, you can use the following function.

Example:

```
d3.geoProjection(project)
```

It constructs a new projection from the specified raw projection project. The project function takes the longitude and latitude of a given point in radians. You can apply the following projection in your code.

Example:

```
var width = 400
var height = 400
var projection = d3.geo.orthographic()
var projections = d3.geo.equirectangular()
var project = d3.geo.gnomonic()
var p = d3.geo.mercator()
var pro = d3.geo.transverseMercator()
   .scale(100)
   .rotate([100,0,0])
   .translate([width/2, height/2])
   .clipAngle(45);
```

Here, we can apply any one of the above projections. Let us discuss each of these projections in brief.

- **d3.geo.orthographic()** − The orthographic projection is an azimuthal projection suitable for displaying a single hemisphere; the point of perspective is at infinity.

- **d3.geo.gnomonic()** − The gnomonic projection is an azimuthal projection that projects great circles as straight lines.

- **d3.geo.equirectangular()** − The equirectangular is the simplest possible geographic projection. The identity function. It is neither equal-area nor conformal, but is sometimes used for raster data.

- **d3.geo.mercator()** − The Spherical Mercator projection is commonly used by tiled mapping libraries.

- **d3.geo.transverseMercator()** − The Transverse Mercator projection.

9.3. Working Example

Let us create the map of India in this example. To do this, we should adhere to the following steps.

Step 1: Apply styles − Let us add styles in map using the code below.

```
<style>
   path {
      stroke: white;
      stroke-width: 0.5px;
      fill: grey;
   }

   .stateTN { fill: red; }
   .stateAP { fill: blue; }
   .stateMP{ fill: green; }
</style>
```

Here, we have applied particular colors for state TN, AP and MP.

Step 2: Include topojson script − TopoJSON is an extension of GeoJSON that encodes topology, which is defined below.

```
<script src = "http://d3js.org/topojson.v0.min.js"></script>
```

We can include this script in our coding.

Step 3: Define variables − Add variables in your script, using the code below.

```
var width = 600;
var height = 400;
var projection = d3.geo.mercator()
   .center([78, 22])
   .scale(680)
   .translate([width / 2, height / 2]);
```

Here, SVG width is 600 and height is 400. The screen is a two-dimensional space and we are trying to present a three-dimensional object. So, we can grievously distort the land size / shape using the ***d3.geo.mercator()*** function.

The center is specified [78, 22], this sets the projection's center to the specified location as a two-element array of longitude and latitude in degrees and returns the projection. Here, the map has been centered on 78 degrees West and 22 degrees North.

The Scale is specified as 680, this sets the projection's scale factor to the specified value. If the scale is not specified, it returns the current scale factor, which defaults to 150. It is important to note that scale factors are not consistent across projections.

Step 4: Append SVG − Now, append the SVG attributes.

```
var svg = d3.select("body").append("svg")
   .attr("width", width)
   .attr("height", height);
```

Step 5: Create path − The following portion of code creates a new geographic path generator.

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

Here, the path generator (d3.geo.path()) is used to specify a projection type (.projection), which was defined earlier as a Mercator projection using the variable projection.

Step 6: Generate data − indiatopo.json

```
{"type":"Topology","transform":{"scale":
[0.002923182318231823,0.0027427542754275428],
"translate":[68.1862,8.0765]},"objects":
{"states":{"type":"GeometryCollection",
"geometries":[{"type":"MultiPolygon","id":"AP","arcs":
[[[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,
25,26,27,28,29,30,31,32,33,34]],[[35,36,37,38,39,40,41]],[[42]],
[[43,44,45]],[[46]],[[47]],[[48]],[[49]],[[50]],[[51]],[[52,53]],
[[54]],[[55]],[[56]],[[57,58]],[[59]],[[60]],[[61,62,63]],[[64]],
[[65]],[[66]],[[67]],[[68]],[[69]],[[-41,70]],
[[71]],[[72]],[[73]],[[74]],[[75]]],
"properties":{"name":"Andhra Pradesh"}},{"type":"MultiPolygon",
"id":"AR","arcs":[[[76,77,78,79,80,81,82]]],
"properties":{"name":"Arunachal Pradesh"}},{"type":"MultiPolygon",
"id":"AS","arcs":[[[83,84,85,86,87,88,89,90,
91,92,93,94,95,96,97,98,99,100,101,102,103]],
[[104,105,106,107]],[[108,109]]], ...
```

Step 7: Draw map − Now, read the data from the indiatopo.json file and draw the map.

```
d3.json("indiatopo.json", function(error, topology) {
   g.selectAll("path")
   .data(topojson.object(topology, topology.objects.states)
   .geometries)
   .enter()
   .append("path")
   .attr("class", function(d) { return "state" + d.id; })
   .attr("d", path)
});
```

Here, we will load the TopoJSON file with the coordinates for the India map (indiatopo.json). Then we declare that we are going to act on all the path elements in the graphic. It is defined as, ***g.selectAll("path")***. We will then pull the data that defines the countries from the TopoJSON file.

Example:

```
.data(topojson.object(topology, topology.objects.states)
   .geometries)
```

Finally, we will add it to the data that we are going to display using the *.enter()* method and then we append that data as path elements using the *.append("path")* method.