TopoJSON

A smaller GeoJSON with some neat tricks

State of the Map US • June 2013
Nelson Minar • @nelson • nelson@monkey.org

many thanks to Mike Bostock @mbostock
Quick introduction
TopoJSON is...

- Text data format for geographic data
- Extension of GeoJSON
- Encodes topology, not just geometry
  Identification of shared arcs
- Space efficient
- Enables topology-aware visualization
- Particularly good for browser presentation
GeoJSON v. TopoJSON
Let’s compare

• Nearly visually identical
• GeoJSON: 67,859 bytes
• TopoJSON: 29,456 bytes
• 43% the size
• After gzip: 20k v. 9k, 46%
Luminary mentions

• Mike Bostock @mbostock
• Jason Davies @jasonondavies
• Shan Carter @shancarter
TopoJSON definition
GeoJSON schema

FeatureCollection
Feature
  properties
GeometryCollection
  Point, MultiPoint
  LineString, MultiLineString
  Polygon, MultiPolygon

Shapes: sequence of points
Reflecting Pool
GeoJSON example

```json
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [[
          [-77.0482, 38.8891],
          [-77.0482, 38.8895],
          ...]]
      },
      "properties": {
        "kind": "water",
        "name": "Reflecting Pool",
        "area": 49918.195312
      }
    }
  ]
}
```
“GeoJSON is spectacularly wrong, yet somehow right enough”

– Sean Gillies
GeoJSON in context

- Simple text format
- Easy export from GIS systems
- Excellent web support
  - Leaflet, D3, OpenLayers, Polymaps, ...
  - Vector tiles (OpenStreetMap, etc)
- Not very space efficient
  [ -59.572094692611529, -80.040178725096297 ]
“GeoJSON is spectacularly wrong, yet somehow right enough”
– Sean Gillies

- Simple text format
- Shapes: sequence of points
- Easy export from GIS systems
- Excellent web support
- Not very space efficient

[ -59.572094692611529, -80.040178725096297 ]
TopoJSON schema

Type: “topology”
Objects
Type: LineString, Polygon, ...
Arcs: included by reference
properties
Arcs: LineStrings
Transform: Scale, Translate

Shapes: sequence of arcs
Arcs: sequence of points
Null Island

Like no place on earth
Two rectangles
Two rects: GeoJSON

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [[0,0], [0,2], [1,2], [1,0], [0,0]]
        ]
      },
      "properties": { "name": "left" } },
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [[1,0], [1,2], [2,2], [2,0], [1,0]]
        ]
      },
      "properties": { "name": "right" } }
  ]
}
```
Two rects: TopoJSON

```json
{
  "type": "Topology",
  "transform": {
    "scale": [1,1],
    "translate": [0,0]
  },
  "objects": {
    "two-squares": {
      "type": "GeometryCollection",
      "geometries": [
        {
          "type": "Polygon",
          "arcs": [[0,1]],
          "properties": {"name": "left"}
        },
        {
          "type": "Polygon",
          "arcs": [[2,-1]],
          "properties": {"name": "right"}
        }
      ]
    }
  },
  "arcs": [
    [[1,2],[0,-2]],
    [[1,0],[-1,0],[0,2],[1,0]],
    [[1,2],[1,0],[0,-2],[-1,0]]
  ]
}
```
Arcs

• Geometry defined by referencing arcs
  "arcs": [[0,1]]   "arcs": [[2,-1]]

• Encoding of shared arcs

• Integer delta encoding of arc shape
  [[1,0], [-1,0], [0,2], [1,0]]

• Scale and translate
**Left**

"arcs": [[0, 1]]

**Right**

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

"arcs": [
    [[1, 2], [0, -2]],
    [[1, 0], [-1, 0], [0, 2], [1, 0]],
    [[1, 2], [1, 0], [0, -2], [-1, 0]]
]
Reflecting Pool

```json
{
  "type": "Topology",
  "transform": {
    "scale": [0.00007125, 0.00000554],
    "translate": [-77.048238, 38.889085]
  },
  "objects": {
    "pool": {
      "type": "GeometryCollection",
      "geometries": [{
        "type": "Polygon",
        "arcs": [[0]],
        "properties": {"name": "Reflecting Pool"}
      }]
    }
  },
  "arcs": [
    [0,0],[0,79],[92,16],[0,4],[7,1],[0,-4],[1,0],[0,-79],[0,-4],[-8,-1],[0,4],[-92,-16]]
}
```
Reflecting Pool
TopoJSON algorithm

- Quantize points to a grid
- Draw every line on the grid
- Pick out common arcs
- Simplify arcs
- Encode all arcs
- Encode all geometries referencing arcs
- 200MB inputs require subtlety
Downsampling

- Quantization: lower precision points
  - Default: 10,000 x 10,000
  - Similar to rounding GeoJSON

- Simplification: fewer points
  - Default: none

- Preserves topology
Quantization
Simplification
TopoJSON in practice
Applications

• Browser delivery
• Smaller files, less bandwidth
• Topology-aware visualizations
• Presentation, not archival
## Smaller sizes

<table>
<thead>
<tr>
<th>Source</th>
<th>GeoJSON</th>
<th>TopoJSON</th>
<th>Pct</th>
<th>gzip</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF (lines)</td>
<td>79M</td>
<td>84M</td>
<td>69M</td>
<td>83%</td>
</tr>
<tr>
<td>SF (polys)</td>
<td>68M</td>
<td>64M</td>
<td>49M</td>
<td>75%</td>
</tr>
<tr>
<td>Chongqing</td>
<td>22M</td>
<td>21M</td>
<td>13M</td>
<td>61%</td>
</tr>
<tr>
<td>CA Rivers</td>
<td>174M</td>
<td>258M</td>
<td>81M</td>
<td>31%</td>
</tr>
<tr>
<td>Zipcodes (no props)</td>
<td>839M</td>
<td>17M</td>
<td>6M</td>
<td>39%</td>
</tr>
</tbody>
</table>
OSM Vector tile map

• 30 tiles, 4 layers
  • OSM land usage, roads, water; NHD rivers
• GeoJSON: 9904k, 1311k gzip
• TopoJSON: 6562k, 815k gzip
• 66% the size (62% after gzip)
• No properties: 6011k vs 2725k, 45% the size
MapBox PBF vectiles

- $z=14$, 28 tiles in San Francisco
  - gzip GeoJSON: 851k
  - gzip TopoJSON: 364k
  - gzip PBF: 1040k
- *But:* PBF has many more layers in it!
  - $z=12$ misc: 815k Topo vs. 706k PBF
  - $z=14$ roads: 90k Topo vs. 71k PBF
- Thanks Dane Springmeyer!
Internal boundaries

• Polygons: land, states, counties
• Renderer extracts internal boundaries
• 2374k of GeoJSON data
• 642k of TopoJSON data
• 27% the size
Testing boundaries

topojson.mesh(
    topology,
    topology.objects.objects.counties,
    function(a, b) {
        return a !== b &&
            a.state === b.state;
    })
Polygon adjacency

- Dorling cartogram
- Replace geometry with scaled circle
- Force directed layout
- Preserve country adjacency
Demo
TopoJSON tools
TopoJSON project

https://github.com/mbostock/topojson

- Command line tools (NodeJS)
- Browser API (Javascript)
- TopoJSON Wiki
Encoding

$ topojson
   --id-property osm_id
   -p name
   -s 0.00001
   -q 10000
   -o sf.json
san-francisco.osm-line.shp

quantization: bounds -122.7368806 37.4490002
    -122.0110009 37.9549999 (spherical)
quantization: maximum error 4.26m (0.0000383°)
simplification: retained 334873 / 733786 points (46%)
prune: retained 167509 / 167509 arcs (100%)
Input files

- GeoJSON, Shapefiles, CSV, TopoJSON
- Inputs need to be topologically valid
- Giant files (> 100MB)
  - Shapefiles stream better than GeoJSON
  - `node --max_old_space_size=8192`
  - Rivers (132MB .shp): 45 seconds
  - Zip codes (836MB .shp): 150 seconds?
Properties

- Stripped by default
- `-p` flag; list which to include
- ISO-8859-1 by default
- Can join to CSV files
Quantization

• 10,000 x 10,000 by default
• Similar to rounding GeoJSON coords
  But more specific: 10,000 for bbox
• Think about pixels on screen

quantization: bounds
  -124.40958558399814 32.50005761622009
  -114.58848453257576 43.33627233273347 (spherical)
quantization: maximum error 75.5m (0.000679°)
Simplification

• --spherical
  • Simplify in geographic space
    • -s <steradians> (area)
    • --simplify-proportion (fraction)
  • --cartesian
    • Simplify in projected pixel space
    • --width --height
Demo
Serving via HTTP

- Treat it like GeoJSON
- MIME type: application/json
- compress, serve cache headers
- Access-Control-Allow-Origin: *
- Beware .topojson file extension
Javascript client API

- `topojson.feature(topology, object)`
  - converts object to GeoJSON
- `topojson.mesh(topology, object, filter)`
  - returns merged arcs as LineString
  - `filter(a, b);` either side of each arc
- `topojson.neighbors(objects)`
  - list of adjacent objects
Other tools

• Sean Gillies’ Python decoder
• Shan Carter’s Distillery
• Josh Livni’s ShpEscape
• Mike Bostock’s US-Atlas
• Wanted: Python encoder (TileStache)
• Wanted: GDAL/OGR support
Use TopoJSON!

- Efficient wire format
- Easy to use simplification, quantization
- Visualize topologies, not just geometries
- Open source, simple, lots of examples

https://github.com/mbostock/topojson