ulm university universität ulm

Visual Computing Group
Ulm University

Projektionen in D3.js

Lukas Pellot, 14.02.2018

Agenda

1. Bestandteile einer Projektion in D3.js
2. Codebeispiel – Landkarte in D3.js
3. Choreoplethen-Karten

1. Bestandteile einer Projektion in D3.js

1. Bestandteile einer Projektion in D3.js

a) Geographische Daten (in Form von GeoJSON oder TopoJSON)

b) Projektionsfunktion

c) Pfadfunktion

1.a) GeoJSON

- Offenes Format, um geografische Daten zu repräsentieren
- Verwendet die JavaScript Object Notation (kurz JSON)

1.a) GeoJSON – Aufbau einer GeoJSON-Datei

- „type“:
  - „Feature“ für einzelnes Element
    - Zweiter Schlüssel: „geometry“, Wert: Einzelnes Element
  - „FeatureCollection“ für mehrere Elemente
    - Zweiter Schlüssel: „features“, Wert: Array von Features
- Optional: „bbox“, Wert: Array mit vier Elementen
  - Die ersten zwei Zahlen stellen den südwestlichen, die letzten zwei den nordöstlichen Eckpunkt des Rechtecks darstellen, das das Element/die Elemente exakt umfasst

1.a) GeoJSON – Aufbau eines einzelnen Elements

- Notwendige Schlüssel
  - „type“: <Name des Elements>
  - „coordinates“:
    - Einfaches Element
      - Einzelnes Koordinaten-Array (bei Point) oder Array von Koordinaten-Arrays (für alle anderen Elemente)
    - Mehrteiliges Element
      - Array an Koordinaten (entsprechend dem vervielfachten einfachen Element)

1.a) GeoJSON – Aufbau eines einzelnen Elements

- Optional: "properties":
  - Wert: JSON-Objekt mit beliebigen Schlüssel-Wert-Paaren

1.a) GeoJSON – Koordinaten

- Punktkoordinaten werden als Array in der Form [Längengrad, Breitengrad] angegeben
- Unterstützte Elemente
  - Einfach: Point (Punkt), LineString (Linie), Polygon
  - Mehrteilig: „Multi“ + entsprechendes einfaches Element (MultiPoint etc.)

1.a) GeoJSON - Beispiel

```
{
    "type": "FeatureCollection",
    "features":
    [
        {
            "type": "Feature",
            "geometry":
            {
                "type": "LineString",
                "coordinates": [[102.0, 0.0], [104.0, 0.0], [106.5, 2.1]]
            }
        },
        {
            "type": "Feature",
            "geometry":
            {
                "type": "Point",
                "coordinates": [13.24, 52.31]
            }
        }
    ]
}
```

1.a) GeoJSON vs. TopoJSON

- TopoJSON: Alternativer Weg, um geografische Daten zu speichern

	GeoJSON	TopoJSON
Speicherung von Strukturen	• Explizite Speicherung des Pfades für jede Struktur • Pfade werden bei aneinandergrenzenden Strukturen mehrfach gespeichert	• Alle genutzten Pfade werden genau einmal gespeichert • Zusätzlich Information, welche Struktur welche Pfade nutzt
Vorteil	• Einfachere Dateistruktur	• Kleinere Dateigröße
Nachteil	• Ggf. wesentlich größere Datei	• Komplexere Dateistruktur • Ggf. weitere Bibliotheken zur Verarbeitung erforderlich

1.b) Projektionsfunktion – Aufgabe

- Problem
  - Die Erde ist eine Kugel (dreidimensional), Projektionsflächen sind Ebenen (zweidimensional) => „Verlust“ einer Dimension
  - Darstellung daher nur verzerrt (unter Verlust von Längen-/Flächen-/Winkeltreue) möglich
- Lösung
  - Aufstellen einer Projektionsfunktion, die Breiten- und Längengrad in X- und Y-Koordinaten „übersetzt“ und so die Verzerrung vereinheitlicht

1.b) Projektionsfunktion – In D3.js

- Es wird stets ein geographischer Punkt auf eine (Pixel-)Ebene projiziert (Default-Größe: 960x500 Pixel)
  - [Breitengrad, Längengrad] → [x-Koordinate, y-Koordinate]
- D3 liefert diverse allgemeine Projektionsfunktionen ‚ab Werk' in `g3-geo` mit
  - Weitere Projektionen können über `d3-geo-projections` nachgeladen werden

1.b) Funktionen auf Projektionen – Auszug (1)

- Projektion eines einzelnen Punktes
  - *`projection`*`(point)`
    - Gibt ein Array mit den Koordinaten des projizierten Punktes (in Pixeln) zurück
    - `point`: Array [Breitengrad, Längengrad]
  - *`projection`*`.invert(point)`
    - „Umkehrfunktion“ zu *`projection`*`(point)`

1.b) Funktionen auf Projektionen – Auszug (2)

- Verschieben des projizierten Punkts
  - *projection*`.translate([tx, ty])`
    - Versetzt die Projektion um die Koordinaten tx und ty entlang der entsprechenden Achsen
    - Falls keine Werte übergeben werden, wird der aktuelle Versatz (Standardwert: [480, 250]) zurückgegeben
    - Versatz bezieht sich auf Mittelpunkt der Projektion

1.b) Funktionen auf Projektionen – Auszug (3)

- Umsetzen des Mittelpunkts der Projektion
  - *projection*`.center([longitude, latitude])`
  - Setzt den Mittelpunkt der Projektion auf den übergebenen Punkt
  - Falls keine Werte übergeben werden, wird der momentane Mittelpunkt zurückgegeben (Standardwert: <0°, 0°>)

1.b) Funktionen auf Projektionen – Auszug (4)

- Skalieren des projizierten Punktes
  - *projection*`.scale(scale)`
    - Skaliert die Koordinaten des projizierten Punktes um den angegebenen Faktor
    - Falls kein Wert übergeben wird, wird der aktuelle Skalierungsfaktor zurückgegeben (Standardwert: Von gewählter Projektion abhängig)

1.c) Pfadfunktion – Aufgabe

- Wird genutzt, um eine (in GeoJSON oder TopoJSON dargestellte) geometrische Struktur einer bestimmten Projektionsfunktion entsprechend darzustellen

1.c) Pfadfunktion – Erzeugung

- `d3.geoPath(projection)` erzeugt eine neue Pfadfunktion, die gemäß der übergebenen Projektionsfunktion Elemente abbildet
  - Wird kein Wert übergeben, wird standardmäßig `d3.geoAlbersUSA()` genutzt (zusammengesetzte Projektion für die USA)

1.c) Pfadfunktion – Manipulation

- *path*`.projection(projection)` ändert die von path genutzte Projektionsfunktion auf die übergebene
  - Wird kein Wert übergeben, wird die aktuell genutzte Projektionsfunktion zurückgegeben

1.c) Pfadfunktion – Nutzung

- *path*(object) zeichnet das übergebene object gemäß der gesetzten Projektionsfunktion als SVG-Pfad

2) Codebeispiel – Landkarte in D3.js

2) Codebeispiel – Landkarte in D3.js (1)

- Allgemeines

```
var width = 1440;
var height = 750;

var svg = d3.select("body")
        .append("svg")
        .classed("map", true);

var background = svg.append("rect")
        .classed("background", true)
        .attr("width", width)
        .attr("height", height);

// Fortsetzung auf nächster Folie
```

2) Codebeispiel – Landkarte in D3.js (2)

- Anlegen von Projektions- und Pfadfunktion

```
// Fortsetzung von letzter Folie

var projection = d3.geoEquirectangular()
        .scale(width / 2 / Math.PI)
        .translate([width / 2, height / 2]);

var path = d3.geoPath()
        .projection(projection);

// Fortsetzung auf nächster Folie
```

2) Codebeispiel – Landkarte in D3.js (3)

- Einlesen und Verarbeiten der GeoJSON-Datei

```
// Fortsetzung von letzter Folie

var geojsonURL = <URL zur darzustellenden GeoJSON-Datei>;

//Weltkarte, Auflösung von 110m

d3.json(geojsonURL, function(geojson){
    svg.append("path")
            .attr("d", path(geojson));
});
```

2) Codebeispiel – Landkarte in D3.js (4)

- Resultierende SVG-Datei → mapSinglePath.html öffnen
  - Styling durch CSS

3) Choreoplethenkarten

3. Choreoplethenkarten

a) Allgemeines

b) Codebeispiel – Choreoplethenkarte in D3.js

3.a) Choreoplethenkarten - Allgemeines

- Alternative Bezeichnung: Thematische Karte
- Abschnitte der Karte werden einer statistischen Variablen entsprechend eingefärbt
- Nutzen: Veranschaulichen, wie Messungen/Werte in bestimmten Regionen variieren
- Beispiel: Arbeitslosenanteil pro Landkreis/Bundesland

3.a) Choreoplethenkarten – In D3.js

- Aufgabe:

  Jedes Land soll entsprechend einer Kennzahl eingefärbt werden

- Problem mit bisheriger Implementierung:

  Erzeugung eines einzelnen Pfades für alle Länder zusammen →

  Nur Füllung mit einzelner Farbe möglich

- Lösungsansatz:

  Ein Pfad für jedes einzelne Land

3.a) Choreoplethenkarten – In D3.js

- Statt dem "features"-Array mit allen Feldern als Ganzes wird jedes Feld einzeln an die Pfad-Funktion übergeben
- Wird durch die Nutzung des enter/update/exit-Patterns ermöglicht

3.b) Codebeispiel – Choreoplethenkarte in D3.js (1)

```
/**
 * Allgemeines, Projektions-/Pfadfunktion wie in vorigem Beispiel
**/

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

3.b) Codebeispiel – Choreoplethenkarte in D3.js (2)

- Resultierende SVG-Datei → mapMultiplePaths.html öffnen
  - Styling durch CSS
  - Beobachtung: Für jedes Land wurde ein eigener path-Tag angelegt

3.b) Codebeispiel – Choreoplethenkarte in D3.js (3)

- Um die path-Tags der Länder ihrer Kennzahl entsprechend einfärben zu können, muss eine Farbskala erstellt werden
- Beim Zeichnen der Pfade wird die Farbe des Landes mithilfe der entsprechenden Kennzahl und der Farbskala errechnet

3.b) Codebeispiel – Choreoplethenkarte in D3.js (4)

```
/**
 * Allgemeines, Projektions-/Pfadfunktion wie in vorigem Beispiel
**/

d3.json(geojsonURL, function(geojson){
    var features = geojson.features;

    var min = d3.min(features, function(d){
        return d.properties.gdp_md_est;
    });

    var max = d3.max(features, function(d){
        return d.properties.gdp_md_est;
    });

    var color = d3.scaleLinear()
            .domain([min, max])
            .range(["white", "crimson"]);

// Fortsetzung auf nächster Folie
```

3.b) Codebeispiel – Choreoplethenkarte in D3.js (5)

```
// Fortsetzung von letzter Folie

    svg.selectAll("path")
            .data(features)
            .enter()
            .append("path")
            .attr("d", path)
            .style("fill", function(d){
                return color(d.properties.gdp_md_est);
            });
)};
```

3.b) Codebeispiel – Choreoplethenkarte in D3.js (6)

- Resultierende SVG-Datei → mapChoreopleth.html öffnen
  - Styling durch CSS

Fragen?

Visual Computing Research Group
Ulm University
http://www.uni-ulm.de/in/mi/mi-forschung/viscom.html

Danke für eure Aufmerksamkeit!