

# Developing a DSL for visualizing geospatial data

---

Christopher Stelzmüller  
Sebastian Tanzer

Präsentations- und Arbeitstechnik, 343.306  
Johannes Kepler Universität Linz

# TABLE OF CONTENTS

2

**01**

Motivation

**03**

Alternative? DSL!

**02**

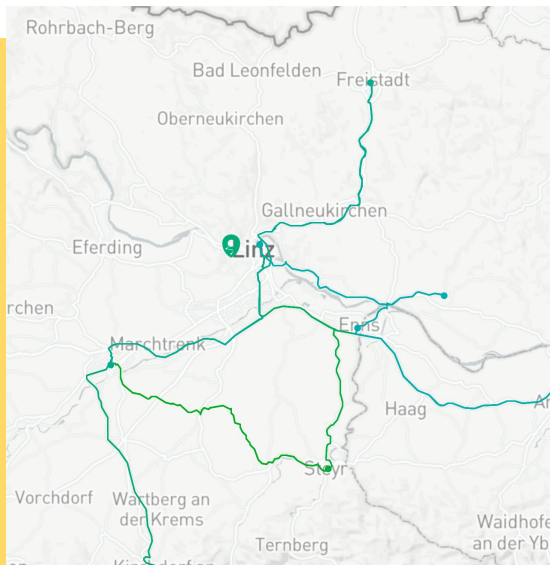
Current Situation

**04**

Conclusion

# MOTIVATION - why do we need Visualisations?

3



01

## Intuitivity

Visualizing geospatial data can make complex topics and data easier to understand

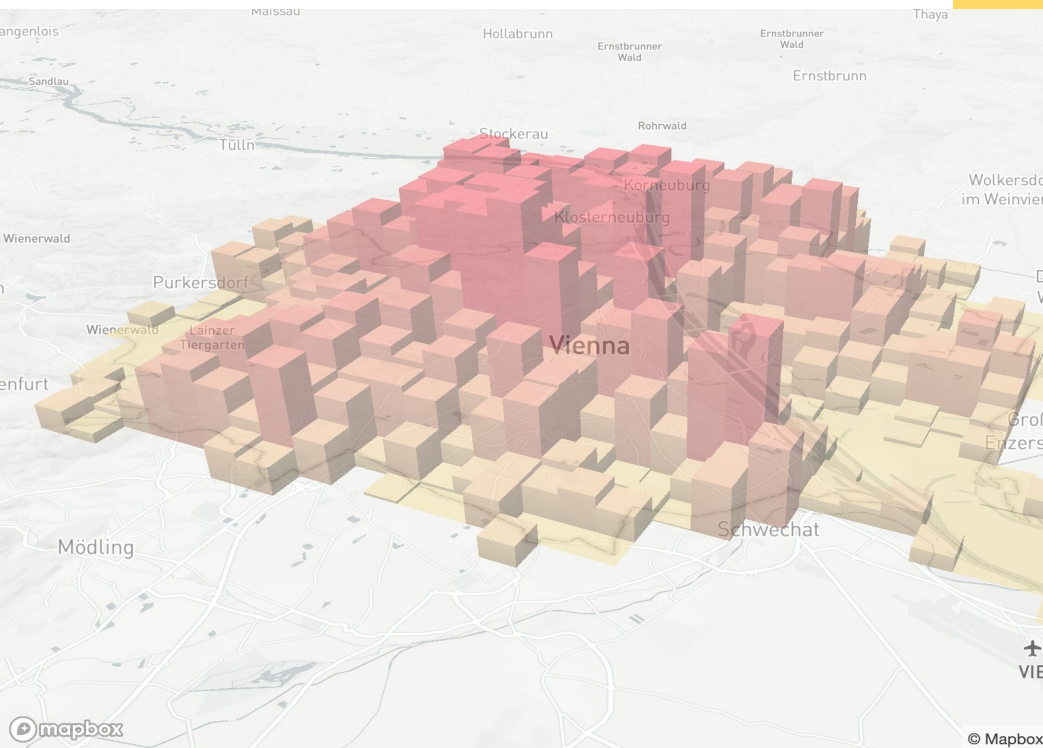
02

## Problems with existing solution

Creating visualizations is often repetitive and error-prone

# AN EXAMPLE

4



**Visualizing Vienna's  
Population distribution**

# CURRENT SITUATION

```
1 {  
2   "id": "population",  
3   "type": "fill-extrusion",  
4   "source": {  
5     "type": "geojson",  
6     "data": "./data/vienna_population.geojson"  
7   },  
8   "paint": {  
9     "fill-extrusion-color": [  
10      "interpolate",  
11      ["linear"],  
12      ["*", 40, ["sqrt", ["get", "tot_p"]]],  
13      0,  
14      "#ecda9a",  
15      5600,  
16      "#ee4d5a"  
17    ],  
18    "fill-extrusion-height": [  
19      "*",  
20      40,  
21      ["sqrt", ["get", "tot_p"]]  
22    ],  
23    "fill-extrusion-opacity": 0.5  
24  }  
25 }
```

- Lack of static validation
- No IDE-support
- Repetitive
- Error-prone

# ALTERNATIVE: A DSL

6



**Improved  
Productivity**



**Readability &  
Communication**



**Editor Support &  
Validation**



**Reusability**

# Our DSL Implementation

```
1 fillExtrusionLayer {  
2     id = "population"  
3     geoJsonSource {  
4         dataUrl = "/data/vienna_routes.geojson"  
5     }  
6     paint {  
7         color = interpolateLinear(  
8             40 * sqrt(get("tot_p")),  
9             0 to "#ecda9a",  
10            5600 to "#ee4d5a"  
11        )  
12        height = 40 * sqrt(get("tot_p"))  
13        opacity = l(.5)  
14    }  
15 }
```

- Embedded DSL
- Written in Kotlin
- Similar Syntax to Java & C#
- Can be used in JVM-backend & Web frontend applications

# Advantages of the DSL

8

```
layers { this: Layers
  fillExtrusionLayer { this: Layer.FillExtrusionLayer
    id = "population"
    geoJsonSource { this: Source.GeoJsonSource
      dataUrl = "/data/vienna_population.geojson"
    }
    paint { this: FillExtrusionPaint
      color = interpolateLinear(
        value: 40
        ...stops: {
          value: Expression<Number>,
          vararg stops: Pair<Number, T>
        }
      )
      height = 40
      opacity = 1
    }
  }
}
```

at.triply.mapboxdsl.DslRoot.kt  
public fun <T : Any> interpolateLinear(  
 value: Expression<Number>,  
 vararg stops: Pair<Number, T>  
) : InterpolateExpression<T>  
  
Produces continuous, smooth results by  
linearly interpolating between pairs of input  
and output values ("stops"). The input may be  
any numeric expression (e.g., ). Stop inputs  
must be numeric literals in strictly ascending  
order. The output type must be number, array,  
or color.  
More information on The mapbox style docs  
mapbox\_dsl.main

```
id = "population"
geoJsonSource { this: Source.GeoJsonSource
  dataUrl = "/data/vienna_population.geojson"
}
(x) dataUrl String?
Press ^, to choose the selected (or first) suggestion and insert a dot afterwards Next Tip
```

```
fillExtrusionLayer { this: Layer.FillExtrusionLayer
  id = "population"
  geoJsonSource { this: Source.GeoJsonSource
    dataUrl = "/data/vienna_population.geojson"
  }
}
```

```
fillExtrusionLayer { this: Layer.FillExtrusionLayer
  id = "population"
  source
  (x) source Source?
  f geoJsonSource {...} (block: Source.GeoJsonSou... Unit
  f vectorSource {...} (block: Source.VectorSourc... Unit
  ^↓ and ^↑ will move caret down and up in the editor Next Tip
```



# COMPARISON - before & after

9

```
1 {
2   "id": "population",
3   "type": "fill-extrusion",
4   "source": {
5     "type": "geojson",
6     "data": "./data/vienna_population.geojson"
7   },
8   "paint": {
9     "fill-extrusion-color": [
10      "interpolate",
11      ["linear"],
12      ["*", 40, ["sqrt", ["get", "tot_p"]]],
13      0,
14      "#ecda9a",
15      5600,
16      "#ee4d5a"
17    ],
18    "fill-extrusion-height": [
19      "*",
20      40,
21      ["sqrt", ["get", "tot_p"]]
22    ],
23    "fill-extrusion-opacity": 0.5
24  }
25 }
```

```
1 fillExtrusionLayer {
2   id = "population"
3   geoJsonSource {
4     dataUrl = "/data/vienna_routes.geojson"
5   }
6   paint {
7     color = interpolateLinear(
8       40 * sqrt(get("tot_p")),
9       0 to "#ecda9a",
10      5600 to "#ee4d5a"
11    )
12     height = 40 * sqrt(get("tot_p"))
13     opacity = 1(.5)
14   }
15 }
```



**First draft of DSL implemented**



**Usable for creating web-visualizations**



**Future work:**

- Implement all functionality in mapbox style language
- Enable usage in Android application

# THANKS!

---

Do you have any questions?

CREDITS: This presentation template was created  
by **Slidesgo**, including icons by **Flaticon**, and  
infographics & images by **Freepik**

Please keep this slide for attribution

[github.com/tuesd4y/mapbox-dsl](https://github.com/tuesd4y/mapbox-dsl)