Responsive Voronoi Treemap with D3v5

Bechtold Oskar, Bobic Aleksandar, Jente Daniel, Schiffer Verena

Institute of Interactive Systems and Data Science (ISDS),
Graz University of Technology
A-8010 Graz, Austria

706.057 Information Visualisation SS 2018
Graz University of Technology

01 Jul 2018

Abstract

Treemaps are used in information visualization to display hierarchical data in a nested figure. Usually, the different layers of the nested data are visualized using rectangles, but it is possible to use more complex figure, such as Voronoi diagrams. Voronoi treemaps use Voronoi maps recursively to create the bottom layers of the hierarchy in each Voronoi cell. In recent years, D3 has become one of the most popular libraries for visualizations. In this paper, a responsive implementation of Voronoi treemaps using the latest version, v5 of D3 is described.

Contents

List of Figures

List of Listings

Chapter 1

Introduction

In the field of information visualization, there are many different possibilities of how to visualize hierarchical data. One way to display these nested diagrams is to use treemaps. Introduced in the early 1990s, treemaps have become popular, as it is possible to display multiple layers of nested data at once. However, treemaps are usually using rectangles for displaying each layer, and in some cases, more advanced figures are needed. Voronoi diagrams are usually used to partition a plane into regions, based on different seeds. Combining treemaps and Voronoi diagrams makes it possible to display nested data, where each cell of the Voronoi diagram contains further Voronoi diagrams. To combine the advantages of treemaps as well as Voronoi diagrams, a responsive implementation of Voronoi treemaps has been created. In this paper the details of the implementation are described and structured in the following way:

- In Chapter 2 Voronoi diagrams, Treemaps, and Voronoi treemaps are explained.
- A detailed description of the project, including the folder structure as well as the installation procedure, user and development guides are given in Chapter 3
- A list of the features implemented in this project and the used libraries can be found in Chapter 4
- An overview of the project, a list of implemented features and future work can be found in Chapter 5

Chapter 2

Background and Related Work

2.1 Voronoi Diagram

The usage of Voronoi diagrams dates back to René Descartes in the 17. century, but became popular in the 20. century when Georgy Voronoy extended the study of the Voronoi diagrams to higher dimensions. In general Voronoi diagrams describe a partition of a plane (in the 2D case) into n many subspaces given n points, named seeds. Each subspace is a convex polygon that contains exactly one seed. Every point in such Voronoi cell is closer to the generating seed than to any other, see Figure 2.1 [Wolfram MathWorld 2018].

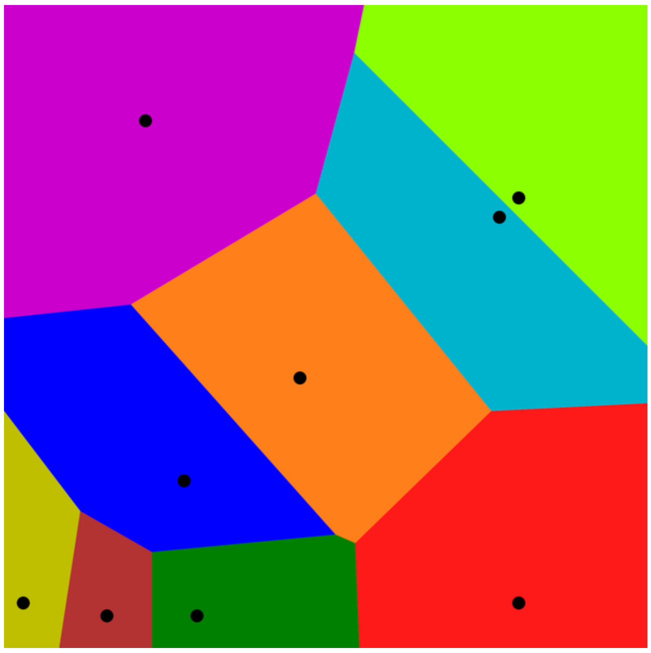

Figure 2.1: A simple example of a Voronoi diagram using the Euclidean distance. A plane is divided into smaller subplanes, called Voronoi cells, using the pre-defined seeds (black dots). The result of the algorithm is a partitioned plane, where every point in each cell is closer to its seed than any other. Image taken from [Jahobr 2018].

2.2 Treemap

Treemaps are used to display hierarchical data in a space-filling visualization by slicing the available space vertically and horizontally. As the name "Treemap" suggests, only tree-like structures can be visualized. Usually, nested rectangles are used to display different levels of a structure, as seen in Figure 2.2. Each rectangle represents one layer of the data, where the size determines the weight of the layer. Different colors are used to ease the distinction between different branches. For each child of one layer, it is possible to sort them for example alphabetically. In general, Treemaps are used in information visualization as it aids to investigate correlations in a dataset [Ao.Univ.-Prof. Dr. Keith Andrews 2018].

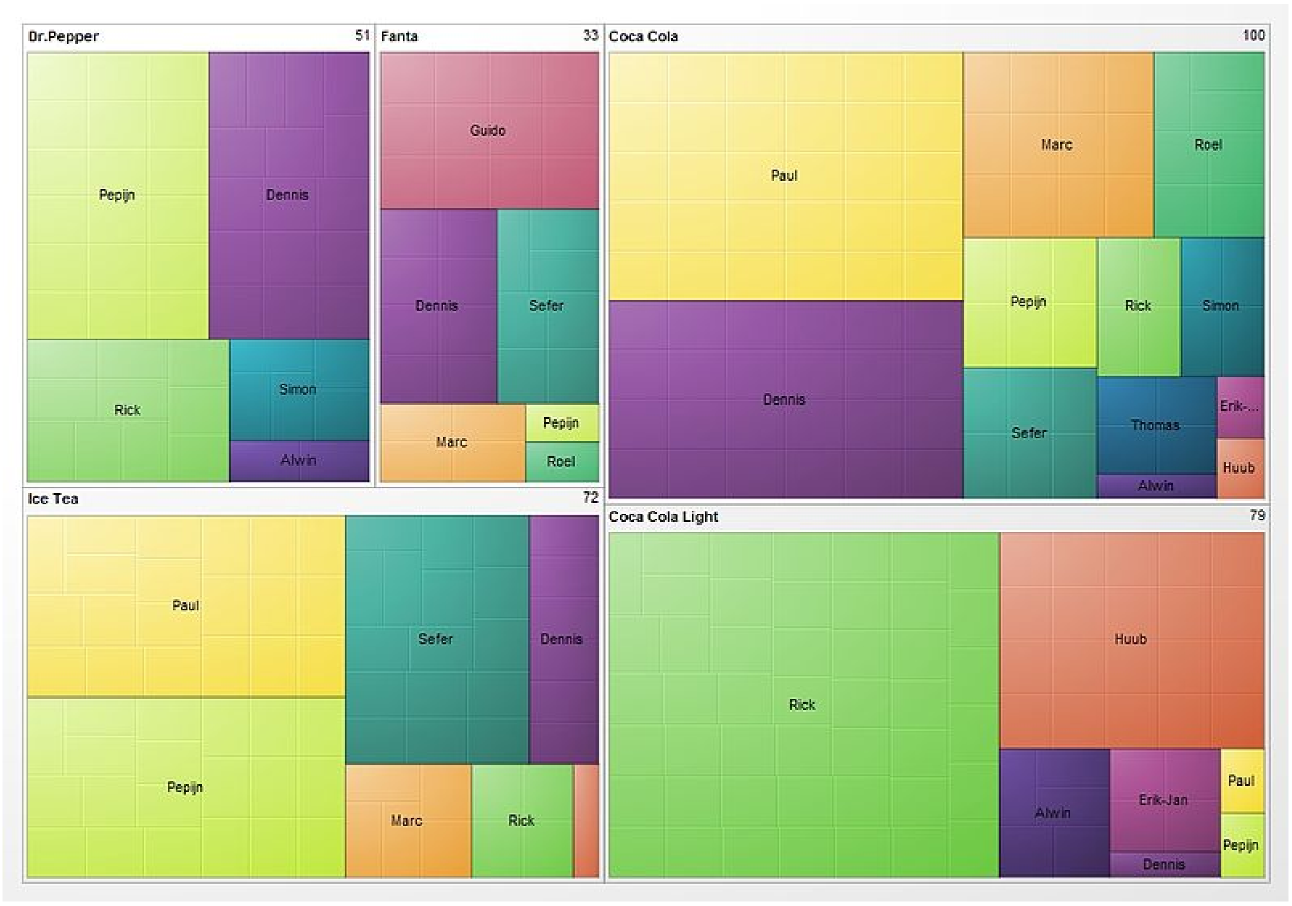

Figure 2.2: A Treemap visualizing the consumption of different soft drinks in a small group. Each node, representing a group member is individually colored to make it easier to identify them. The size of each rectangle is determined by the liters consumed by a person. Image taken from [GBoshouwers 2018].

2.3 Voronoi Treemap

Voronoi Treemaps combine Voronoi Diagrams (Section 2.1) and Treemaps (Section 2.2). Instead of using rectangles, Voronoi Treemaps use Voronoi diagrams to visualize each level of the hierarchy, see Figure 2.3. However odering the children accoding to one criteria is not possible in Voronoi Treemaps.

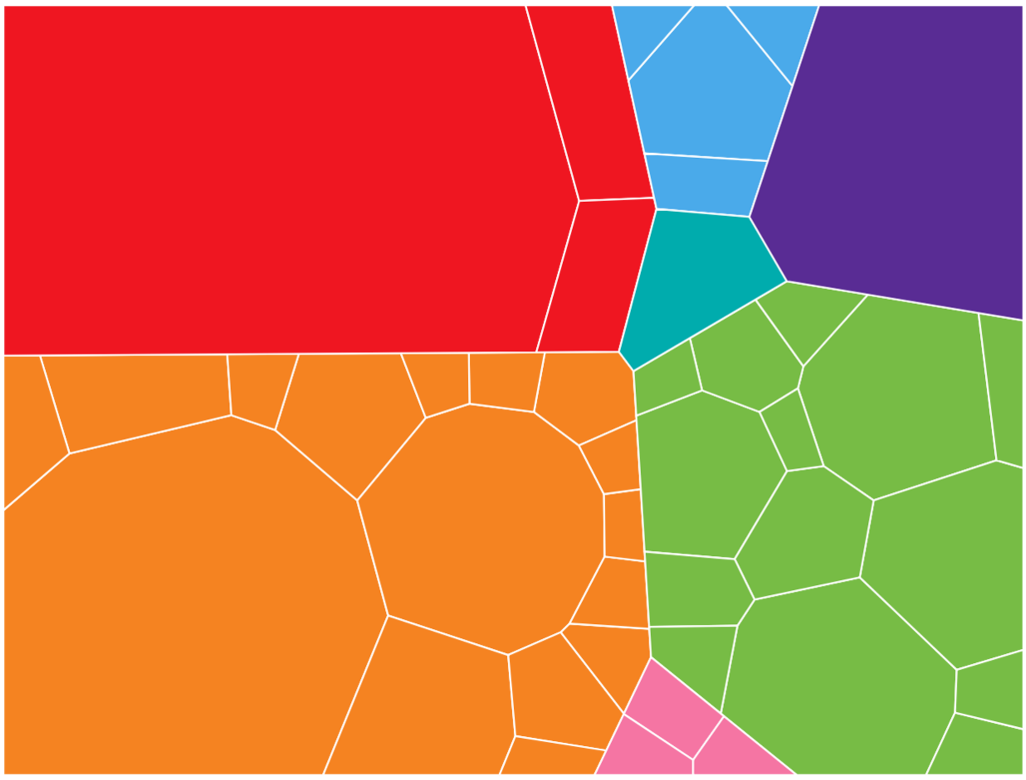

Figure 2.3: A Treemap using a Voronoi Diagram in each level of the hierarchy. This visualization is called a Voronoi Treemap. Different colors are used to make the different cells easily distinguishable.

Chapter 3

Project Setup

3.1 Structure and Dependencies

This project contains four different npm projects:

- **D3-voronoi-map** forked from the original npm-project, see d3-voronoi-map npm website
- **D3-voronoi-treemap** forked from the original npm-project, see d3-voronoi-treemap npm website
- **Library** contains the code of the project, which can be included in arbitrary projects
- **Example** folder containing various example projects, that utilize the library

3.2 Installation Guide

Before using the library, the following steps need to be executed:

- Build *d3-voronoi-map* and *d3-voronoi-treemap* using *yarn build* in case changes have been made in their source files.
- Build and start the library from the first terminal, see 3.1

```
cd library
npm install
npm start
```

Listing 3.1: Steps for building the library

  npm install, installs the needed node_modules and *npm start* builds the project to the *dist* directory and starts serving the files on a localhost and listening to file changes.

- Build and start the Example from the second terminal, see 3.2

```
cd example
npm install
npm start
```

Listing 3.2: Steps for building the examples

3.3 User Guide

In Section 3.2 the steps for correctly installing the library are given. After building both folders, access the URL given by the command line after the example project is built. When accessing this URL, some implemented examples are shown in combination with feature descriptions. If custom examples need to be added, either the files need to be replaced (same filename), or further examples need to be added to the *index.ts* file.

3.4 Developement Guide

The library and the example project are using webpack to bundle the project which can be seen under *webpack.config.js*. Additionally, *TypeScript (2.8.3)* is used to support type checking in JavaScript and compiling to older versions of JavaScript.

The Typescript compilation options can be seen under *tsconfig.json*. TypeScript Linter options are given at *tslint.json*. Styles are written in Sass and later compiled to css.

Chapter 4

Features of the Implementation

4.1 JSON Data

An example of a dataset for the Voronoi Treemap can be seen in Listing 4.1. Each level of hierarchy can be added via the "children" map. "label" describes the label of each level and "color" the color of each level of hierarchy. "weight" and "short_label" are needed to determine the size and short label of each cell.

```json
{
    "label": "world",
    "children": [
      {
        "label": "South America",
        "color": "#4aaaea",
        "children": [
          {"label": "Brazil", "weight": 2.39, "short_label": "BR"},
          {"label": "Argentina", "weight": 0.79, "short_label": "AR"},
          {"label": "Venezuela", "weight": 0.5, "short_label": "VE"},
          {"label": "Colombia", "weight": 0.39, "short_label": "CO"}
        ]
      },
      {
        "label": "North America",
        "color": "#ef1621",
        "children": [
          {"label": "United States", "weight": 24.32, "short_label": "US"},
          {"label": "Canada", "weight": 2.09, "short_label": "CA"},
          {"label": "Mexico", "weight": 1.54, "short_label": "MX"}
        ]
      }
    ]
}
```

Listing 4.1: Example of a JSON file for the Voronoi Treemap

4.2 TreeML data

As seen in Listing 4.2 not only JSON but also TreeML datasets can be used.

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE tree SYSTEM "treeml.dtd">
<tree>
  <declarations>
   <attributeDecl name="label" type="String"/>
   <attributeDecl name="weight" type="Real"/>
   <attributeDecl name="short_label" type="String"/>
  </declarations>
  <branch>
    <attribute name="label" value="World in TreeML"/>
    <attribute name="short_label" value="WWW"/>
    <branch>
      <attribute name="label" value="Europe-Sample"/>
      <leaf>
        <attribute name="label" value="Austria"/>
        <attribute name="weight" value="20"/>
        <attribute name="short_label" value="A"/>
      </leaf>
      <leaf>
        <attribute name="label" value="Germany"/>
        <attribute name="weight" value="60"/>
        <attribute name="short_label" value="G"/>
      </leaf>
      <leaf>
        <attribute name="label" value="France"/>
        <attribute name="weight" value="10"/>
        <attribute name="short_label" value="F"/>
      </leaf>
    </branch>
  </branch>
</tree>
```

Listing 4.2: Example of a TreeML file for the Voronoi Treemap

4.3 Custom Colors

The implemented algorithm enables users to define custom colors in their dataset. Each level of the hierarchy can be assigned one color as HEX codes. If no color is assigned, all remaining cells are drawn according to a color scheme, see Listing 4.1. Furthermore, it is possible to add color to one parent node of the data, and the color is assigned to every child node, as seen in Figure 4.1.

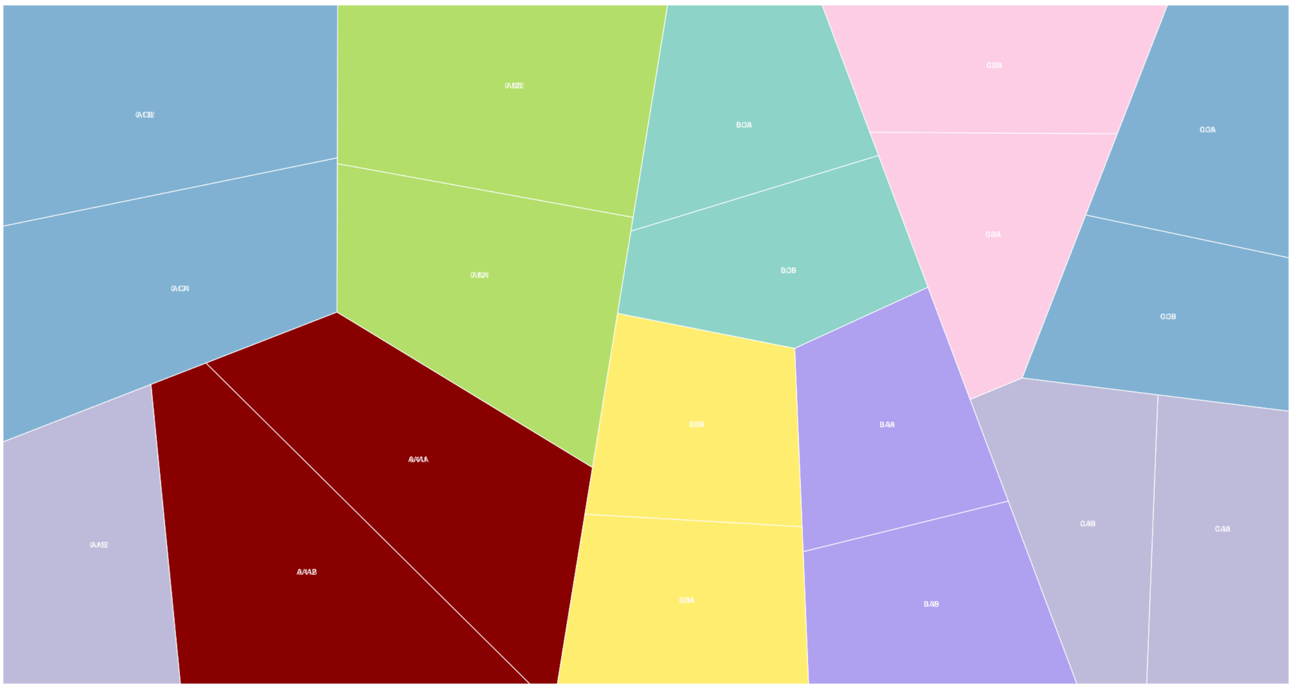

Figure 4.1: A multilevel treemap visualization. Each node of the hierarchy can be assigned a custom color, or a color scheme can be assigned. Furthermore, if colors are added to a parent node, the color gets assigned to each child node. In this example this has been assigned to the *AC* node, therefore *ACA* and *ACB* are colored in the same way.

4.4 Vega-like specification

To simplify the modification of the appearance of the Voronoi Treemap as well as the used dataset, a specification similar to Vega [2018] has been used to configure the graph and can be seen in Listing 4.3. The data has to be provided in the same structure as is provided in the example. In case there is not enough room in the Voronoi treemap for the entire label the alternative short_label is used. The weight, as well as the color, can be defined by the user manually for each cell using the weight or color label respectively. Finally, each node in the dataset can have multiple child nodes which can be provided under the children attribute as an array. The second parameter of the responsive Voronoi treemap library initialization function is the configuration object. In this object, the user has to provide the id of the SVG element where they want the Voronoi treemap to be rendered using the dom_id parameter. They can also specify if they would like to use a color_schema and a custom schema. In addition, they can add a clipping_path to render the voronoi_treemap in a specific shape. The user can also specify the id of a button that will be used for downloading the treemap as an SVG.

```
interface IVoronoiTreemapLayout
{
  dom_id: string;
  color_scheme?: ReadonlyArray<string>;
  use_color_scheme?: boolean,
  clipping_path?: Array<[number, number]>;
  maxIterationCount?: number;
  convergenceRatio?: number;
  minWeightRatio?: number;
  downloadButtonID?: string;
}

interface IDataLayout
{
  label: string;
  short_label: string;
  weight?: number;
  color?: string;
  children: Array<IDataLayout>;
}
```

Listing 4.3: An example of a config for speciifying the layout of the graph. The specifications are similar to the ones used in Vega [2018]

4.5 Responsiveness

The implementation of the Voronoi Treemap is responsive. The image resizes to different screens, which means that this implementation can be used on PC's as well as on mobile devices. The implementation of the Voronoi Treemap is not static and allows navigation through the different layers of the hierarchy. Different types of interactions have been added to support all types of devices. For the user to navigate to a deeper level of the hierarchy, they have to click or double tap on the cell they want to enter. To navigate one level higher in the hierarchy, they have to click or tap somewhere outside of the Voronoi cells. In addition, a user can see additional information about a cell by hovering over it with a mouse or by taping it once with a finger.

4.6 Export as SVG

The Voronoi Treemap that has been rendered via this library can also be exported to a device as SVG. To enable this feature the user has to create a button element on their webpage and set an id attribute on it. Then they have to provide the id of the created button to the function for initializing the Voronoi Treemap as the downloadButtonID parameter of the layout object (the second argument). The downloaded image will be identical to the image the user sees on the webpage. If they decide to navigate to any of the deeper levels of their data, only those deeper levels will be shown in the downloaded image.

4.7 Example Datasets

Some examples of how to use this library have been added to make life easier for users and developers. One example that has been added is taken from the website of Mike Bostock, the author of the *d3-voronoi-treemap* library, see Weighted Voronoi Treemap.

Chapter 5

Concluding Remarks

This paper described the implementation of a responsive and user configurable version of the Voronoi treemap visualization using D3v5. The fundamentals of Voronoi diagrams, treemaps, and Voronoi treemaps were explained. Next, the project structure and the setup process were described. Finally, the features implemented were introduced.

5.1 Feature overview

The library supports the usage of JSON and TreeML data. In addition, it enables the user to use custom colors or color schemas. All of the generated Voronoi treemaps are responsive and interactive. The user can zoom and navigate through the data hierarchy. Furthermore, a download functionality can be easily added to each Voronoi treemap. To make the library more straightforward to use, a Vega-like specification was introduced to the initial library function.

5.2 Future Work

Even though the library is ready for everyday usage, there is still room for improvement. First, the library could be improved by adding additional user configurable settings. For example, the library could enable the user to select the number of levels they want to see at once when viewing a certain Voronoi Treemap. If they choose three levels, they would see three levels deep. However, if they would like to see deeper levels of their hierarchy, they would still have to enter one of the cells. Second, the d3-Voronoi-map and d3-Voronoi-treemap could be rewritten with ES6 syntax by using a class structure that was introduced in ES6. In addition, types could be added to d3-Voronoi-map and d3-Voronoi-treemap to improve readability, speed up debugging and increase the understanding for new developers that would like to expand them. Third, the handling of the data could be improved by introducing an option for the definition of the data structure. With that, the library could handle any data structure and would not force the user to have a specific data labeling schema. Finally, a SKOS parser could be implemented which would expand the number of data formats the library supports. However, that could be a library for itself which would be used by the Voronoi treemap library.

Bibliography

Ao.Univ.-Prof. Dr. Keith Andrews [2018]. *Tree Maps*. Keith Andrews Website. 24th Jun 2018. `courses.isds.tugraz.at/ivis/ivis.pdf` (cited on page 4).

GBoshouwers [2018]. *Treemap*. Wikimedia Commons. 24th Jun 2018. `commons.wikimedia.org/wiki/File:Gradient_grouped_treemap.jpg` (cited on page 5).

Jahobr [2018]. *Voronoi Diagram*. Wikimedia Commons. 24th Jun 2018. `commons.wikimedia.org/wiki/File:Voronoi_static_euclidean.png` (cited on page 4).

Vega [2018]. *Vega Specifications*. Vega Website. 24th Jun 2018. `vega.github.io/vega/docs/specification/` (cited on pages 11–12).

Wolfram MathWorld [2018]. *Voronoi Diagrams*. MathWorld Website. 24th Jun 2018. `mathworld.wolfram.com/VoronoiDiagram.html` (cited on page 3).