Multi-Dimensional Visualisation with Three.js and svelte

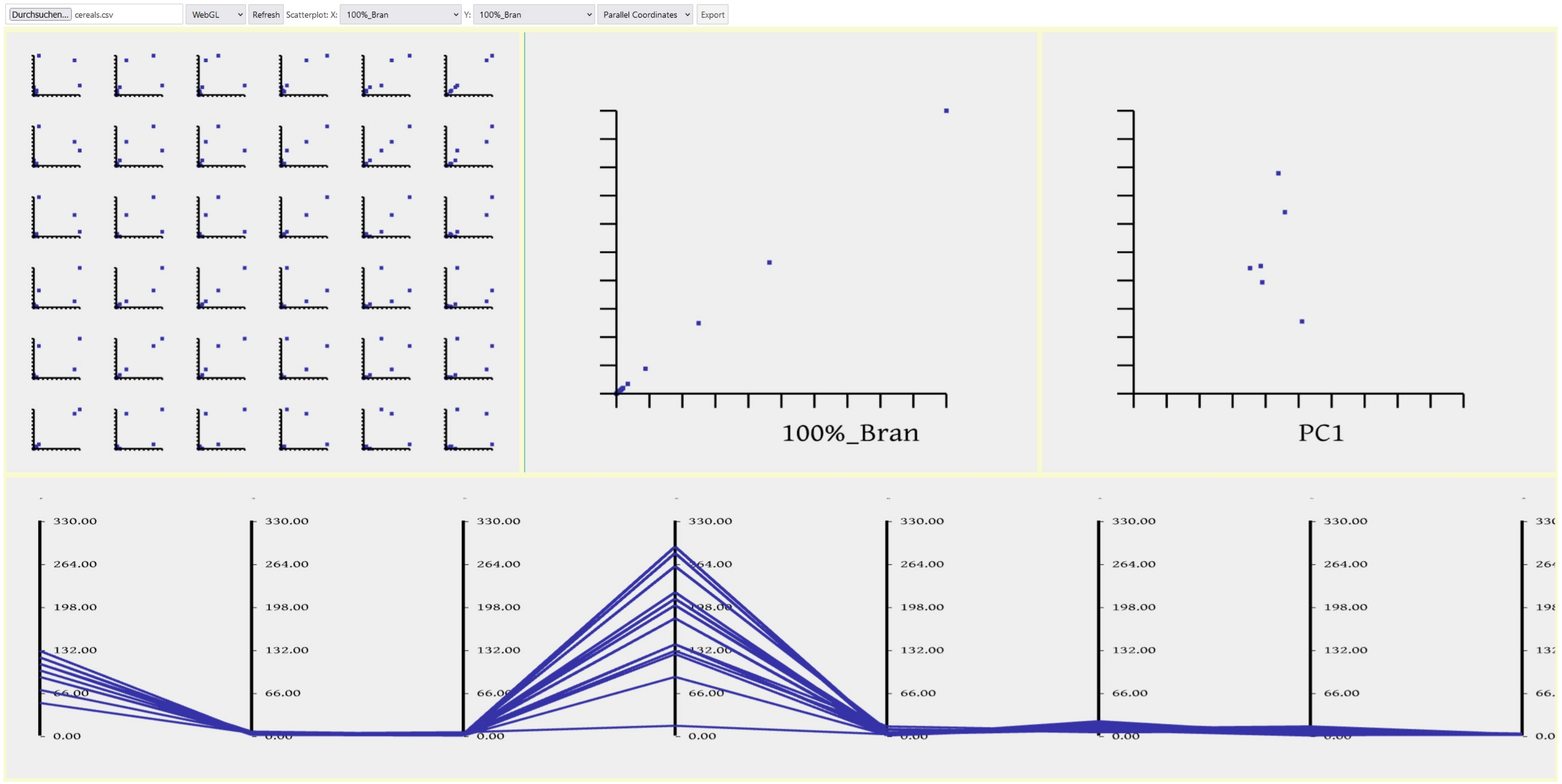
Group G3
Christoph Söls, Valerio Mariani, Sebastian Überreiter

29 June 2022

Introduction

- Create a (Web-) application to visualize data in 4 ways:
  - Scatterplot
  - Scatterplot Matrix
  - Similarity Map
  - Parallel Coordinates
- Frontend framework: Svelte
- Graphics library: Three.js
- Library used for CSV import: D3

Introduction



Svelte

- Free JavaScript frontend framework
- Builds upon HTML, CSS and JavaScript
- Compiles to vanilla JavaScript
- Makes JavaScript itself reactive

Getting Started with Svelte

- If not already installed, add node.js.
- Create Svelte app
- Add Three.js
- Base file: App.svelte

```
npx degit sveltejs/template my-
svelte-project
cd my-svelte-project
npm install
npm run dev
```

Getting Started with Three.js

- Import Three.js:

```
import * as THREE from "./three";
<script src="js/three.js"></script>
```

- 3 main components: scene, camera, renderer

```
const scene = new THREE.Scene()
const camera = new THREE.PerspectiveCamera()
const renderer = new THREE.WebGLRenderer()
```

  - 3 main components setup → free to use all functionality
  - Three.BoxGeometry(), Three.Mesh(), etc...

Handling CSV Import

- Import D3:

```
import * as d3 from "d3";
```

- Get file and parse it via d3.csvParse():

```
const input = file;
const reader = new FileReader();
reader.onload = function (e) {
  const text = e.target.result;
  var data_csv = d3.csvParse(text);
};
reader.readAsText(input);
```

- The output of d3.csvParse() function is a formatted JSON.

Handling SVG Export

- Define SVGRenderer:

```
let rendererSVG = new SVGRenderer();
```

- Export the rendered SVG:

```
var XMLS = new XMLSerializer();
var svgfile = XMLS.serializeToString(rendererSVG.domElement);

var svgData = svgfile;
  var preface = '<?xml version="1.0" standalone="no"?>\r\n';
  var svgBlob = new Blob([preface, svgData], {
    type: "image/svg+xml;charset=utf-8",
  });
var svgUrl = URL.createObjectURL(svgBlob);
```

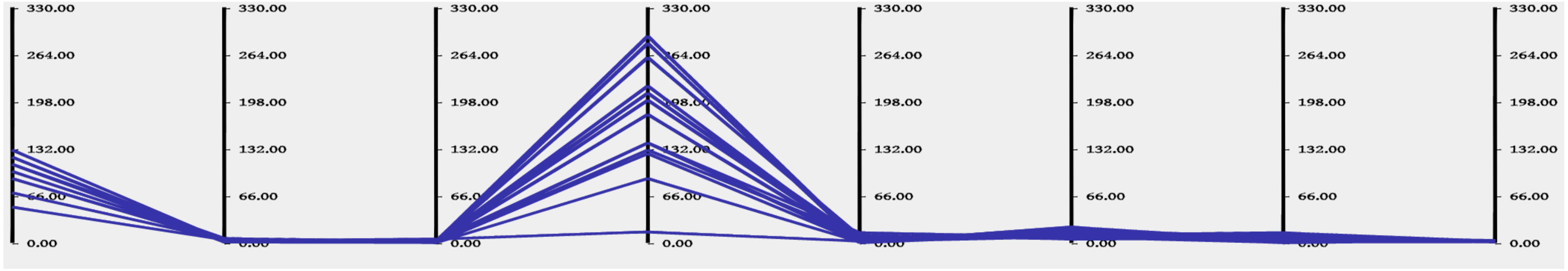
Resizable Grid Layout for Plots

- Every plot has a canvas.
- Canvases are placed into the CSS grid layout.

```
grid-template-areas:
  "area1 area1 area1 area2 area2 area2 area3 area3 area3"
  "area1 area1 area1 area2 area2 area2 area3 area3 area3"
  "area1 area1 area1 area2 area2 area2 area3 area3 area3"
  "area4 area4 area4 area4 area4 area4 area4 area4 area4"
  "area4 area4 area4 area4 area4 area4 area4 area4 area4";
```

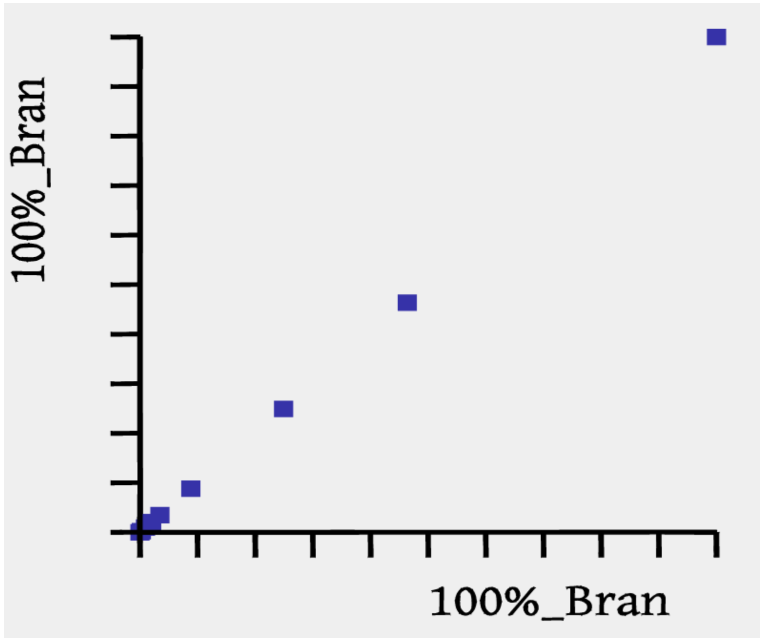
- Each Plot is then rendered into its corresponding <canvas> elements in the grid.

Parallel Coordinates



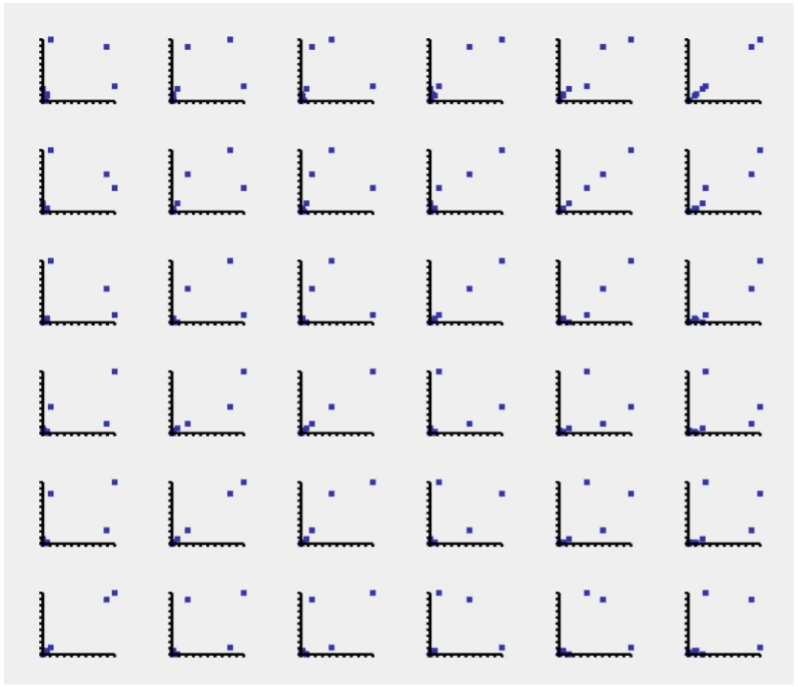
- Detailed code in the live demo.

Scatterplot



- Detailed code in the live demo.

Scatterplot Matrix



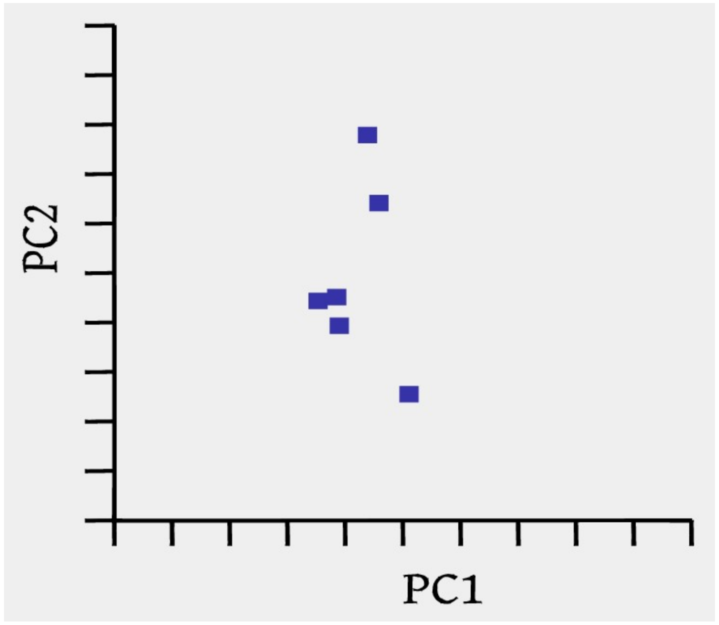
- Detailed code in the live demo.

Similarity Map

- Based on Principal Component Analysis (PCA).
- PCA code taken from github.com/bitanath[1].
- Create eigenvectors and eigenvalues and plot top two eigenvectors based on highest eigenvalues.

[1] https://github.com/bitanath/pca

Similarity Map



- Detailed code in the live demo.

Creating Text with Three.js

- CSS2D renderer can create a HTML which can be edited with CSS:
  - Easy
  - Cannot be exported to SVG
- TextGeometry renders the text as a Three.js object:
  - More complicated to edit
  - Can be exported to SVG

Creating Text with Three.js

- Render function for TextGeometry:

```
const geometry = new TextGeometry(t, {
    font: font,
    size: font_size,
    height: 0,
  });

const material = new THREE.MeshBasicMaterial({ color: 0x000000 });
const text = new THREE.Mesh(geometry, material);
text.position.x = x;
text.position.y = y;
if (rot) {
  text.rotation.z = rot;
}

scene.add(text);
```

Interactivity: Hovering

- Define Raycaster and Pointer object:

```
const raycaster = new THREE.Raycaster();
const pointer = new THREE.Vector2();
```

- Define Raycaster and Pointer object:

```
// update the picking ray with the camera and pointer position
raycaster.setFromCamera(pointer, camera);

// calculate objects intersecting the picking ray
const intersects = raycaster.intersectObjects(scene.children);
```

Discussion

- It is possible to create a (Web-) application with svelte + Three.js with multiple charts.
- D3 CSV import is possible.
- SVG export is possible:
  - However, the files are currently really large.
- Interactivity is possible, but complicated, because:
  - There is no built-in interactivity.
  - A raycaster has to be managed manually.
- Scaling the canvas is possible, but complicated.