Interactive Visualizations (IVI) Bericht

Luca Gisler

HS2023

Repository:

Data: https://earthquake.usgs.gov/earthquakes/feed/v1.0/csv.php

Contents

[2. LO1: Performance 2](#_Toc154135115)

[WebGL 2](#_Toc154135116)

[Tiles 2](#_Toc154135117)

# LO1: Performance

Performance in interactive visualizations is vital in the big data era to ensure a positive user experience, provide real-time data analysis, handle complex data effectively, ensure accessibility and scalability, facilitate quick insight discovery, and maintain a competitive edge. In this LO it will be looked at how the performance of Visualizations behaves.

## WebGL

In the context of performance visualizations, WebGL's functionality as a rasterization engine is crucial. It allows for efficient rendering of complex visuals like points, lines, and triangles, essential for detailed and dynamic graphical representations. The use of GPU for processing and the requirement for specific code through vertex and fragment shaders, written in GLSL, enable high-performance visualizations. These shaders play a key role: the vertex shader computes vertex positions for rendering primitives, and the fragment shader assigns colors to each pixel, directly influencing the visual output's quality and performance. Since WebGL’s API primarily focuses on setting up states for these shaders, it's integral in optimizing the performance of visualizations. Efficient use of WebGL in performance visualizations hinges on how well one can manage and execute these shader functions and state setups, ensuring that the necessary data is accessible and processed effectively by the GPU (*WebGL Fundamentals*, o. J.).

The benefit of the GPU over the CPU is that is specifically made for Visual tasks. It contains thousands of kernels, processes are running parallel to each other, more efficiently in in processing lots of small tasks.

To investigate the performance differences, I created a visualization that shows 500000 random points in a 2D coordinate system. Points in a 2D coordinate system. Once the visualization was rendered with the SVG mode and once with WebGL.

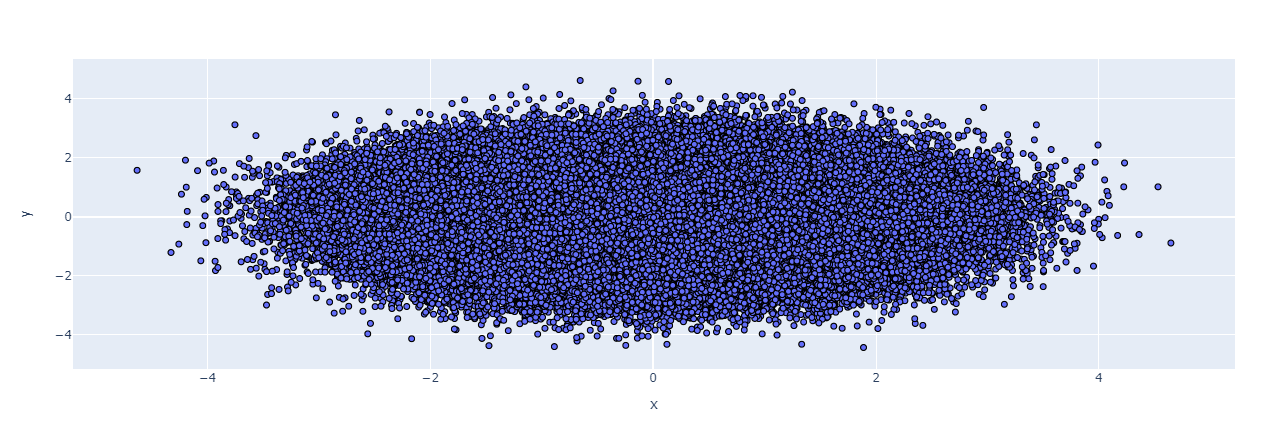


Figure : Plot WebGL

The plot created with WebGL was shown much faster than the one created with SVG. The runtime difference was not that huge. For the WebGL [1.31s] and for SVG [1.54s]. The plot created with WebGL was much smother to handle than the SVG. Handling the SVG plot let VScode to crash or stop working. This shows that the WebGL plot is much more resource efficient than SVG.

Ein Bild, das Screenshot, Majorelle Blue, Electric Blue (Farbe) enthält.

Automatisch generierte Beschreibung

Figure : Plot SVG

In addition the look of the SVG plot is different from the WebGL. The SVG points are much more detailed and have very clear borderlines.

## Tiling