Implementation of Complex Data Visualizations in Browser

Author: Kamil Rafikov
Email: mailbox@kamil-rafik.com
WhatsApp: +996 755 439 777
Telegram: kamilrafikov

Updated at November 8th, 2025.
Edition: 2 (final fact-checking not yet performed).

Table of Contents

Introduction

This document is written to demonstrate my theoretical knowledge of the topic of implementation of complex data visualizations in browser. It is not related anyhow to the projects performed by me at previous jobs. That's why you should not and cannot use the information provided further as for development of MVP as for development of production stage projects. However, I suppose it may be helpful for you when you will plan to implement such functionality.

Further information is provided on visualizations for business and technological environments only, gaming topic is not covered.

Basic technologies

The set of basic modern technologies used for in-browser data visualizations include the following:

a) Canvas API (including OffscreenCanvas);

b) WebGL API;

c) WebGPU API (successor of WebGL API, with limited availability in browsers);

d) Web Workers API for running heavy computational tasks in separate threads;

e) SharedArrayBuffer object to exchange data between workers and parallelize complex computations;

f) WebAssembly API to load WebAssembly code;

g) WebAssembly code format to implement blocks of functionality with performance that is close to native machine code (this technology is useful only in cases of extremely large amounts of mathematical calculations and/or necessity to use for these purposes ported versions of C++ libraries, for example);

h) AssemblyScript, a TypeScript-like language optimized to be compiled into WebAssembly format;

i) possibly, some custom libraries that allow to interact with DOM from WebAssembly code (for exotic cases only).

Exact numbers of calculations and rendered objects where each technology is applicable the most may be found in the academic literature and popular blog articles. Such benchmarks are calculated and published every several years.

Security issues

The main security issues that must be considered upon production deployment of complex data visualizations include the following:

a) standard malicious data injection issues and cross-origin request issues that are common for all web applications;

b) dependencies on malicious Wasm code modules;

c) caching of secure data or rendered images on client-side;

d) denial of service on heavy computations, large amounts of data, and multiple parallelized jobs in workers.

Tools and technologies

Canvas-based frameworks: Konva, Paper.js, ZIM.

General purpose charting: D3.js; Observable Plot; Vega and Vega-Lite visualization grammars; Plotly.js; Apache ECharts; Chart.js; Highcharts.

For handling large datasets: Deck.gl, Luma.gl.

Graphs and networks: Cytoscape.js, Sigma.js, Graphology, ElkJS.

GIS: CesiumJS, Kepler.gl, Mapbox GL JS.

Summary

I hope this document demonstrates my general knowledge of the topic, I hope you enjoyed reading, and I expect long-term productive work with you. Thank you for spending your time!