Motivation and Basic Idea
- Web-based visualization interfaces are getting increasingly popular
- For a high usability, an interface must always run smoothly
- Especially challenging with high number of displayed elements, e.g., in large graph visualizations or big multiple coordinated views apps

Comparing SVG, Canvas, and WebGL
- Measuring the interface performance as frames per second (FPS) during user interactions

How to Compare Web-Visualizations
Measuring Performance through FPS:
- FPS can more accurately represent the perceived performance
- Longer loading time may be acceptable
- But: slow-acting or laggy interface is not

Using Tree Visualizations as Example:
- Consist of a large number of nodes and can easily be scaled
- Here: tree visualization similar to Value Driver Trees

Findings:
- WebGL faster than SVG and Canvas
- SVG on par with Canvas

Performance loss above 400 nodes:
- Performance losses started above 400 nodes for all technologies
- Corresponds to ca. 8,000 graphical elements

SVG and Canvas are on par, WebGL faster:
- Surprisingly, SVG and Canvas perform almost equally
- Contradicting general assumption that Canvas is faster than SVG
- Drop for WebGL (with text) less extreme

WebGL is optimal without text:
- WebGL performance drop caused by bitmap-based text handling
- Almost constant FPS without text (50 FPS measured for extreme setup of 400,000 nodes; ca. 8 million graphical elements)

Browser Dependency:
- No notable differences regarding FPS between browsers
- Exception: Initial lag in Firefox for SVGs caused by applying CSS

Strategies for Performance Improvements
Flexible Level of Detail:
- Not all elements are of interest to the user
  - Idea: remove elements (e.g., details, labels) when zoomed out
  - Effect: fewer graphical elements; speeds up rendering performance

Combined Approaches:
- Idea: Combine different technologies, e.g., WebGL for graphic elements and a separate Canvas for text elements
  - Challenge: keep both scenes synchronized

Asynchronous Tile Loading:
- Idea: Asynchronous tile loading similar to map applications
- Rendering efforts are split up across multiple threads
- Improved web standards allow for a client-side implementation
- Implementation: client starts multiple threads (Webworker API) running headless browser rendering instances (Offscreen Canvas API)
- Effect: Interface runs constantly at 60 FPS
- Latency for loading tiles becomes main performance indicator