Rendering Large Visualizations with WebGL

Before switching to WebGL:

- Doubling SVG Performance at Khan Academy: http://www.crmarsh.com/svg-performance/

- Speeding up D3.js: A Checklist https://blog.safaribooksonline.com/2014/02/20/speeding-d3-js-checklist/

- Think of other ways to represent your data that does not involve tens of thousands of DOM elements

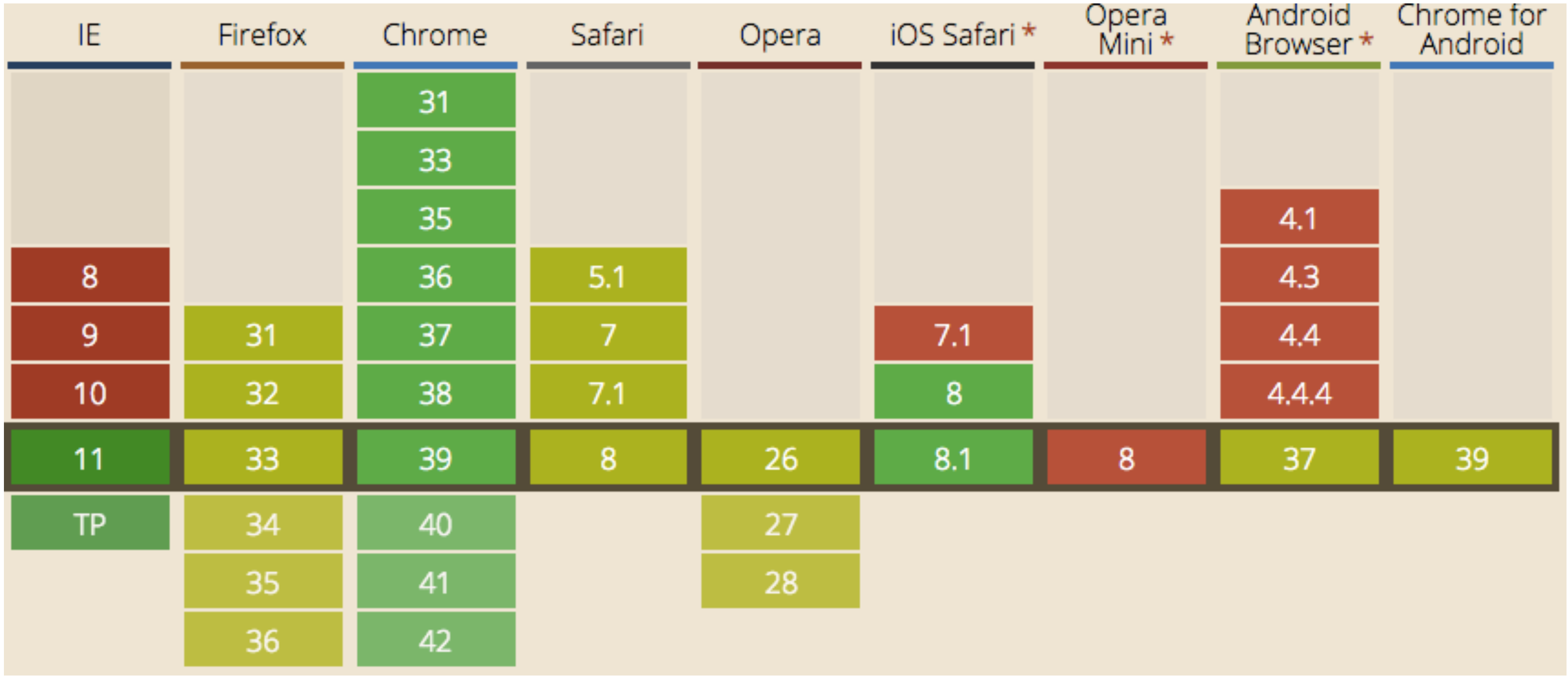
WebGl goodness

- better performance for large visualizations

- better performance for the rest of your DOM
(transitions render faster because the CPU is less bogged down)

WebGl badness

- browser support (getting better...)



IE	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Chrome for Android
		31						
		33						
		35					4.1	
8		36	5.1				4.3	
9	31	37	7		7.1		4.4	
10	32	38	7.1		8		4.4.4	
11	33	39	8	26	8.1	8	37	39
TP	34	40		27				
	35	41		28				
	36	42						

WebGl badness

- code overhead

- with a d3 stack, we can create a network that zooms, pans, makes node selections, and changes the node colors in about 130 lines of code

- with pixi.js (a 2D WebGL API), we needed about 400… though it is a work in progress

- your css has no power in canvas land

What can be salvaged from D3?

D3	pixi.js v2 Made by goodboy
• philosophy	• create API for enter / exit / update
• zoom & pan	• API exposes transform function that takes a transform object
• brush	• with a vacant SVG DOM element, let D3 brush calculate extent, API exposes selection FN (slightly hacky)

- using webGL does not guarantee faster results - manually drawing circles and lines through PIXI's graphics API results in performance similar if not worse than SVG

- the key to better performance is generating your sprites and putting them in sprite batch containers

Compare & Contrast

4363 nodes, 32039 links	**SVG**	**WebGl**
Transform (translate & zoom)	4-5 FPS	18-22fps
Changing all node & link colorings	1.26 seconds to complete	0.01s to complete
Selecting all nodes and links via brush	0.89 seconds to complete	0.01s to complete