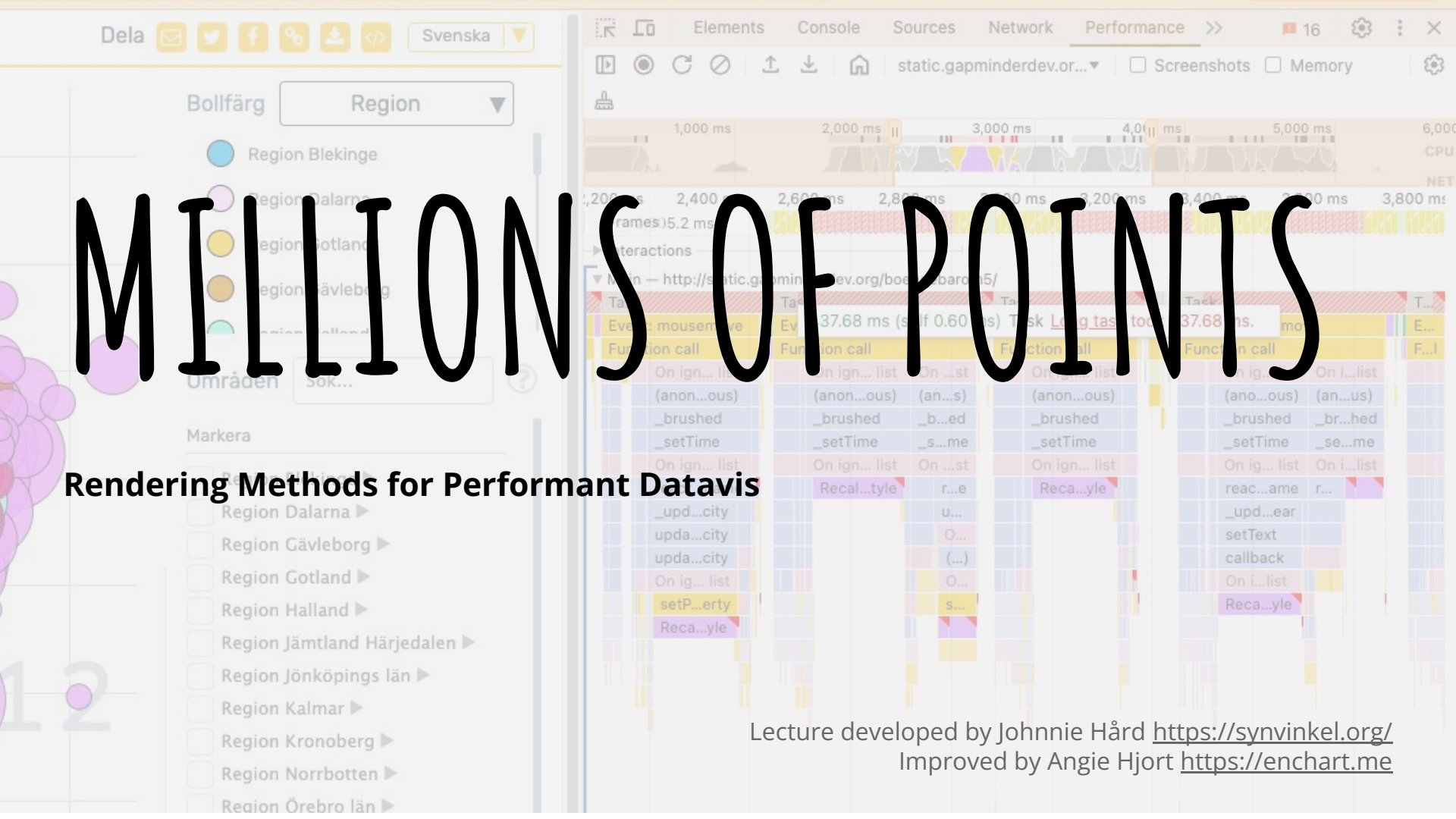
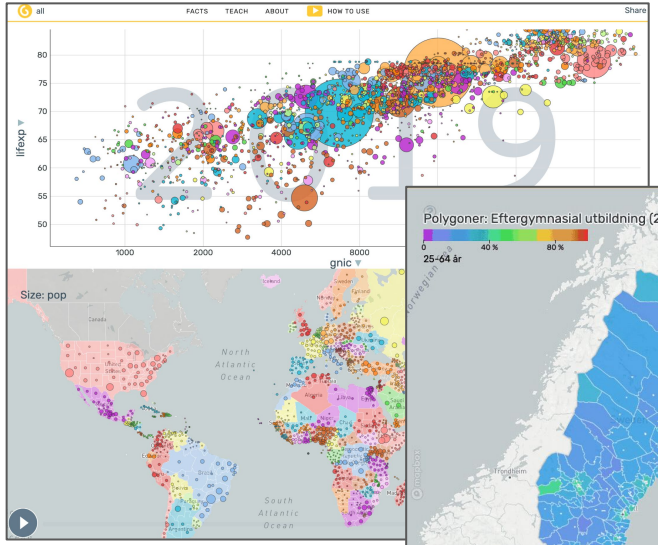


MILLIONS OF POINTS

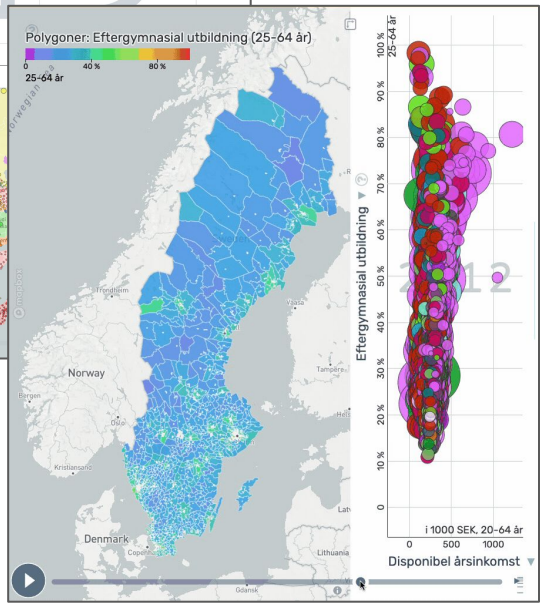
Rendering Methods for Performant Datavis

Lecture developed by Johnnie Hård https://synvinkel.org/
Improved by Angie Hjort https://enchart.me

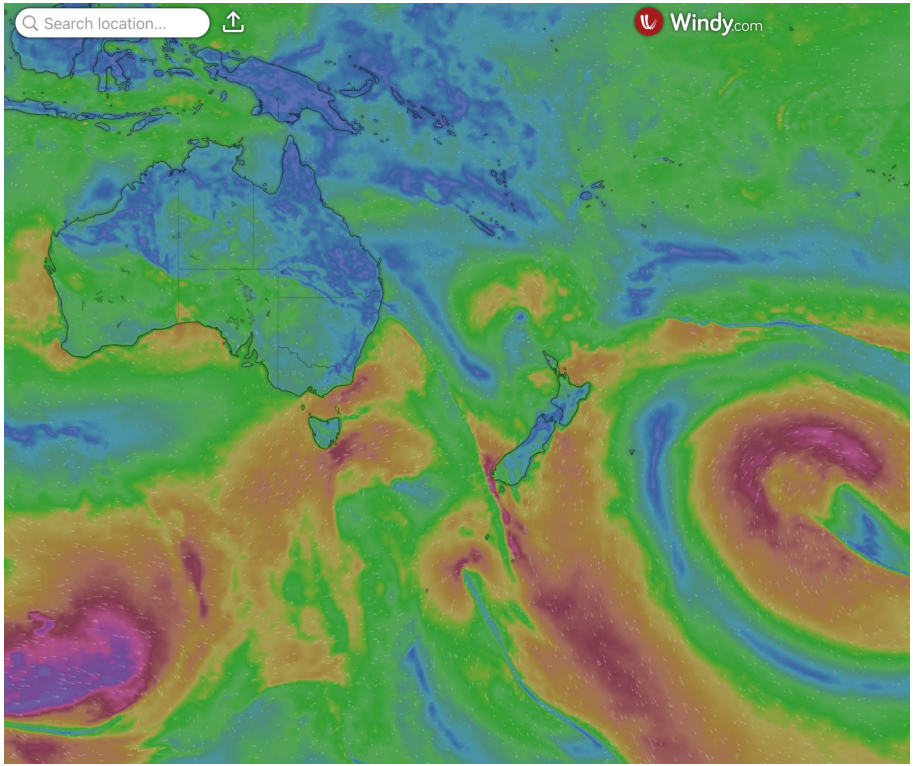
Compare performance of these tools. Think why the difference?



map panning [here]



time slider [here]



Particle animation here:
https://www.windy.com/

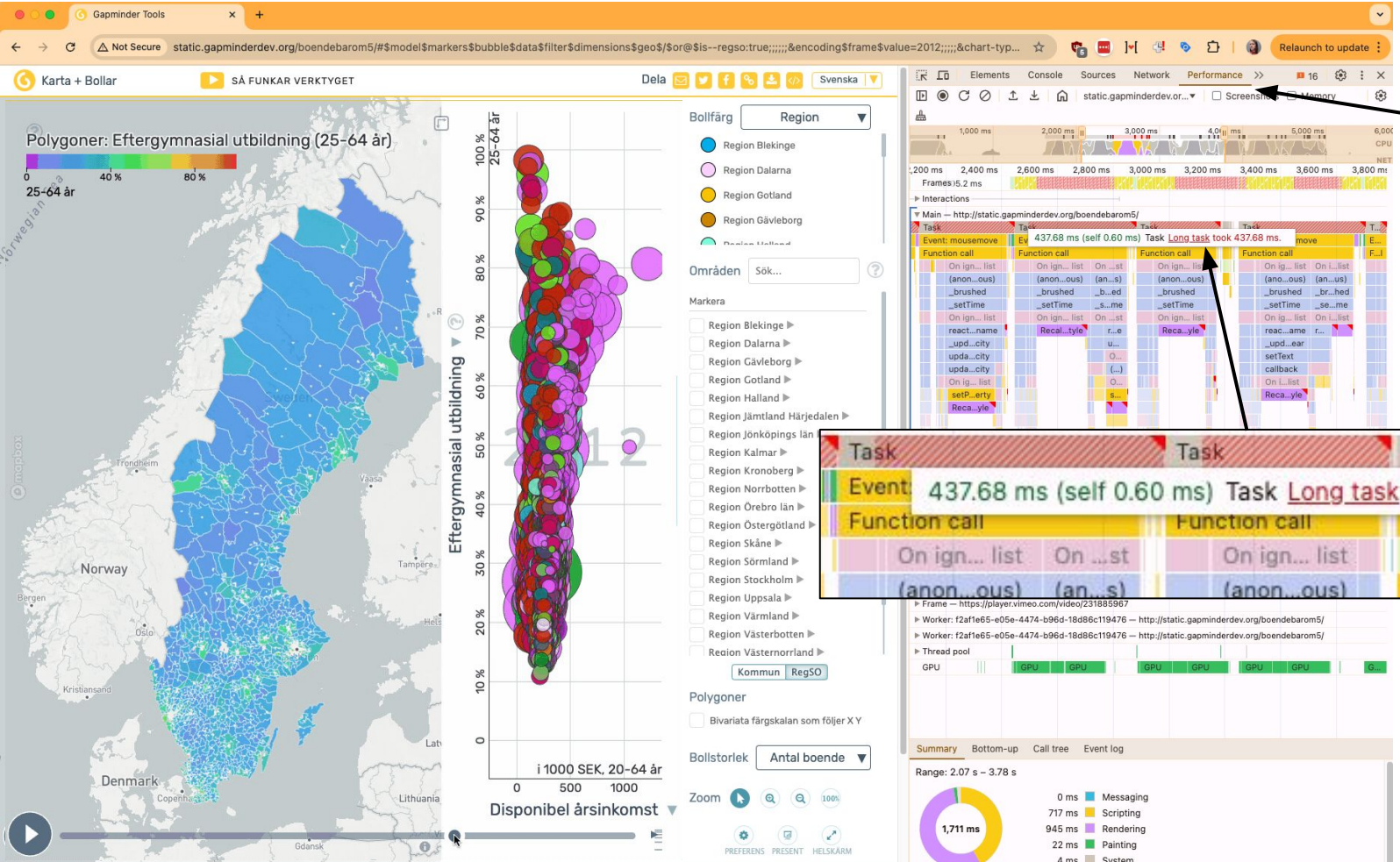
What do I mean by performance?

Reduce gulf of evaluation
Improve the satisfaction of using technology

- animations look smooth
- interactions are immediate
- quick to load (not always as important)

How to measure performance?

Example of painfully slow tool using D3.js + SVG to draw & animate ~3300 bubbles + ~3300 map polygons



Tips

Use performance graph in dev tools!

What options are there?

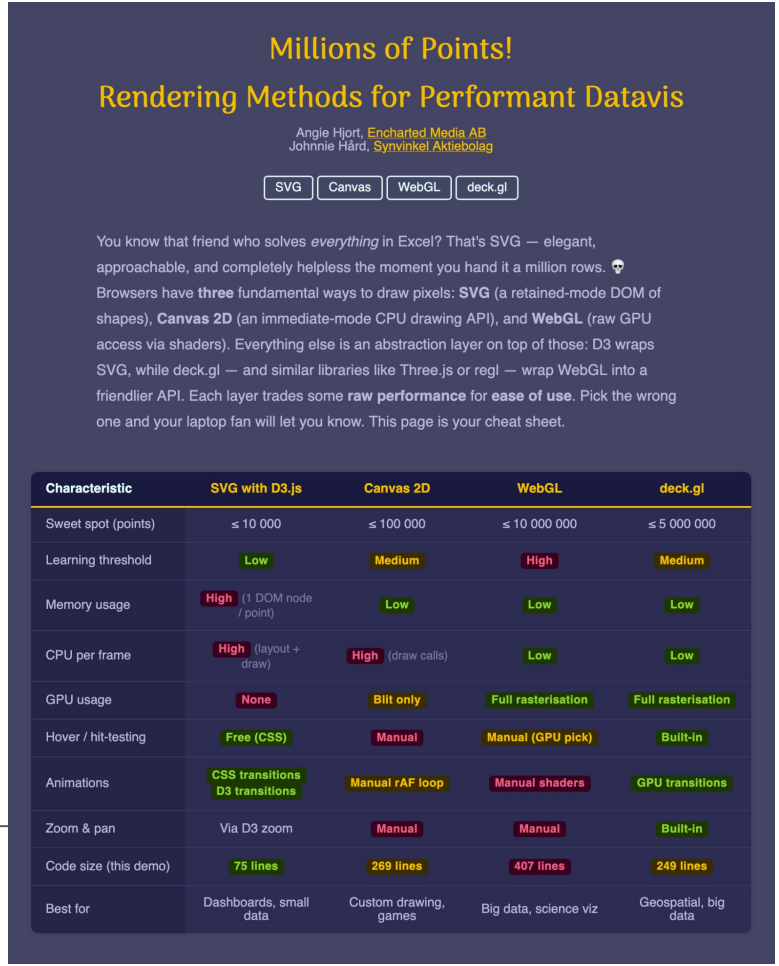
scope: web based, interactive

- SVG
- Canvas
- WebGL

- Abstraction libraries made for more ergonomic use of any of the above
  - SVG -> d3.js -> Observable plot
  - DeckGL, three.js, Regl for WebGL

Interactive comparison tool

https://rendering-methods.enchart.me/



Millions of Points!

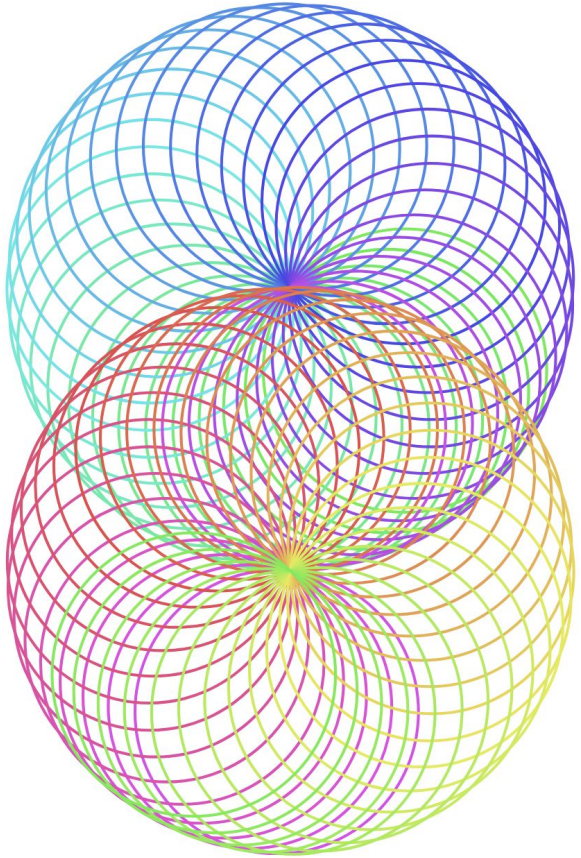
Rendering Methods for Performant Datavis

Angie Hjort, Encharted Media AB
Johnnie Hård, Synvinkel Aktiebolag

SVG | Canvas | WebGL | deck.gl

You know that friend who solves *everything* in Excel? That's SVG — elegant, approachable, and completely helpless the moment you hand it a million rows. 💀 Browsers have **three** fundamental ways to draw pixels: **SVG** (a retained-mode DOM of shapes), **Canvas 2D** (an immediate-mode CPU drawing API), and **WebGL** (raw GPU access via shaders). Everything else is an abstraction layer on top of those: D3 wraps SVG, while deck.gl — and similar libraries like Three.js or regl — wrap WebGL into a friendlier API. Each layer trades some **raw performance** for **ease of use**. Pick the wrong one and your laptop fan will let you know. This page is your cheat sheet.

Characteristic	SVG with D3.js	Canvas 2D	WebGL	deck.gl
Sweet spot (points)	≤ 10 000	≤ 100 000	≤ 10 000 000	≤ 5 000 000
Learning threshold	Low	Medium	High	Medium
Memory usage	High (1 DOM node / point)	Low	Low	Low
CPU per frame	High (layout + draw)	High (draw calls)	Low	Low
GPU usage	None	Blit only	Full rasterisation	Full rasterisation
Hover / hit-testing	Free (CSS)	Manual	Manual (GPU pick)	Built-in
Animations	CSS transitions D3 transitions	Manual rAF loop	Manual shaders	GPU transitions
Zoom & pan	Via D3 zoom	Manual	Manual	Built-in
Code size (this demo)	75 lines	269 lines	407 lines	249 lines
Best for	Dashboards, small data	Custom drawing, games	Big data, science viz	Geospatial, big data



svg.svg

```
<svg viewBox="0,0,1152,600" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
    <circle fill="none" stroke="rgb(85, 235, 71)" stroke-width="2" r="100" cx="656.9016994374947" cy="258.7785252292474"/>
    <circle fill="none" stroke="rgb(72, 235, 71)" stroke-width="2" r="100" cx="644.1487612701519" cy="273.18296480290917"/>
    <circle fill="none" stroke="rgb(71, 235, 84)" stroke-width="2" r="100" cx="628.8734649546132" cy="284.8787176133882"/>
    <circle fill="none" stroke="rgb(71, 235, 97)" stroke-width="2" r="100" cx="611.641187871325" cy="293.43289424566126"/>
    <circle fill="none" stroke="rgb(71, 235, 110)" stroke-width="2" r="100" cx="593.0897402123277" cy="298.52888297080784"/>
    <circle fill="none" stroke="rgb(71, 235, 123)" stroke-width="2" r="100" cx="573.9057580116643" cy="299.97806834748457"/>
    <circle fill="none" stroke="rgb(71, 235, 136)" stroke-width="2" r="100" cx="554.7992890077944" cy="297.7268123568193"/>
    <circle fill="none" stroke="rgb(71, 235, 149)" stroke-width="2" r="100" cx="536.4775119795682" cy="291.8584396812554"/>
    <circle fill="none" stroke="rgb(71, 235, 162)" stroke-width="2" r="100" cx="519.6185622696572" cy="282.59015364714776"/>
    <circle fill="none" stroke="rgb(71, 235, 175)" stroke-width="2" r="100" cx="504.84643227907145" cy="270.2649969798849"/>
    <circle fill="none" stroke="rgb(71, 235, 188)" stroke-width="2" r="100" cx="492.70787592899" cy="255.3391549243344"/>
    <circle fill="none" stroke="rgb(71, 235, 201)" stroke-width="2" r="100" cx="483.65217191423176" cy="238.36507067426555"/>
    <circle fill="none" stroke="rgb(71, 235, 214)" stroke-width="2" r="100" cx="478.0144947615753" cy="219.97099805144077"/>
    <circle fill="none" stroke="rgb(71, 235, 228)" stroke-width="2" r="100" cx="476.00350917215195" cy="200.837748241477"/>
    <circle fill="none" stroke="rgb(71, 229, 235)" stroke-width="2" r="100" cx="477.6936468126985" cy="181.6734912486439"/>
    <circle fill="none" stroke="rgb(71, 216, 235)" stroke-width="2" r="100" cx="483.0223514111749" cy="163.18754473153214"/>
    <circle fill="none" stroke="rgb(71, 203, 235)" stroke-width="2" r="100" cx="491.7923941227308" cy="146.06412036113164"/>
    <circle fill="none" stroke="rgb(71, 189, 235)" stroke-width="2" r="100" cx="503.6791734684658" cy="130.93699941505773"/>
    <circle fill="none" stroke="rgb(71, 176, 235)" stroke-width="2" r="100" cx="518.2427296577732" cy="118.36607492828159"/>
    <circle fill="none" stroke="rgb(71, 163, 235)" stroke-width="2" r="100" cx="534.9440286137873" cy="108.8166286347424"/>
    <circle fill="none" stroke="rgb(71, 150, 235)" stroke-width="2" r="100" cx="553.1649129889345" cy="102.64210971268398"/>
    <circle fill="none" stroke="rgb(71, 137, 235)" stroke-width="2" r="100" cx="572.2309817330065" cy="100.07105273594108"/>
    <circle fill="none" stroke="rgb(71, 124, 235)" stroke-width="2" r="100" cx="591.4365513830271" cy="101.19861903091049"/>
    <circle fill="none" stroke="rgb(71, 111, 235)" stroke-width="2" r="100" cx="610.0707751943951" cy="105.98307451499497"/>
    <circle fill="none" stroke="rgb(71, 98, 235)" stroke-width="2" r="100" cx="627.4439533781506" cy="114.24733438063477"/>
    <circle fill="none" stroke="rgb(71, 85, 235)" stroke-width="2" r="100" cx="642.9130606358858" cy="125.6855174522606"/>
    <circle fill="none" stroke="rgb(71, 72, 235)" stroke-width="2" r="100" cx="655.905546153622" cy="139.87426762282735"/>
    <circle fill="none" stroke="rgb(84, 71, 235)" stroke-width="2" r="100" cx="665.9405251566371" cy="156.2884233349067"/>
    <circle fill="none" stroke="rgb(97, 71, 235)" stroke-width="2" r="100" cx="672.6465776721618" cy="174.32045513918126"/>
    <circle fill="none" stroke="rgb(110, 71, 235)" stroke-width="2" r="100" cx="675.7754957226847" cy="193.30295189710148"/>
    <circle fill="none" stroke="rgb(123, 71, 235)" stroke-width="2" r="100" cx="675.2114701314478" cy="212.53332335643046"/>
    <circle fill="none" stroke="rgb(136, 71, 235)" stroke-width="2" r="100" cx="670.9753769132067" cy="231.29980479482776"/>
    <circle fill="none" stroke="rgb(149, 71, 235)" stroke-width="2" r="100" cx="663.2240046000844" cy="248.90780123379562"/>
    <circle fill="none" stroke="rgb(162, 71, 235)" stroke-width="2" r="100" cx="652.2442511011448" cy="264.70559615694447"/>
    <circle fill="none" stroke="rgb(175, 71, 235)" stroke-width="2" r="100" cx="638.4425048846016" cy="278.1084731878463"/>
    <circle fill="none" stroke="rgb(188, 71, 235)" stroke-width="2" r="100" cx="622.3296035119862" cy="288.62035792312145"/>
    <circle fill="none" stroke="rgb(201, 71, 235)" stroke-width="2" r="100" cx="604.5019262469976" cy="295.8521789017376"/>
    <circle fill="none" stroke="rgb(214, 71, 235)" stroke-width="2" r="100" cx="585.6193205494378" cy="299.5362681245744"/>
    <circle fill="none" stroke="rgb(228, 71, 235)" stroke-width="2" r="100" cx="566.3806794505622" cy="499.5362681245744"/>
    <circle fill="none" stroke="rgb(235, 71, 229)" stroke-width="2" r="100" cx="547.4980737530025" cy="495.8521789017376"/>
    <circle fill="none" stroke="rgb(235, 71, 216)" stroke-width="2" r="100" cx="529.6703964880138" cy="488.62035792312145"/>
    <circle fill="none" stroke="rgb(235, 71, 203)" stroke-width="2" r="100" cx="513.5574951153985" cy="478.10847318784636"/>
    <circle fill="none" stroke="rgb(235, 71, 189)" stroke-width="2" r="100" cx="499.75574889885524" cy="464.70559615694447"/>
    <circle fill="none" stroke="rgb(235, 71, 176)" stroke-width="2" r="100" cx="488.77599539991564" cy="448.9078012337956"/>
```

What options are there?

scope: web based, interactive

- **SVG**
- Canvas
- WebGL

- Abstraction libraries made for more ergonomic use of any of the above
  - SVG -> d3.js -> Observable plot
  - DeckGL, three.js, Regl for WebGL

+ Tool interoperability
+ Free interactivity
+ CSS animations
+ Responsivity handled by browser
+ Accessibility
+ Server side rendering

– Slow because CPU rendered :(
– Slow because uses DOM

Interactive comparison tool

https://rendering-methods.enchart.me/

What options are there?

scope: web based, interactive

- SVG
- **Canvas**
- WebGL

- Abstraction libraries made for more ergonomic use of any of the above
  - SVG -> d3.js -> Observable plot
  - DeckGL, three.js, Regl for WebGL

~~+ Tool interoperability~~
~~+ Free interactivity~~
~~+ CSS animations~~
~~+ Responsivity handled by browser~~
~~+ Accessibility~~
+ Server side rendering

– Still using CPU to calculate pixels :(

+ Bypasses DOM → faster!
+ Full control
+ Simple and understandable API

Interactive comparison tool

https://rendering-methods.enchart.me/

What options are there?

scope: web based, interactive

- SVG
- Canvas
- **WebGL**

- Abstraction libraries made for more ergonomic use of any of the above
  - SVG -> d3.js -> Observable plot
  - DeckGL, three.js, Regl for WebGL

~~+ Tool interoperability~~
~~+ Free interactivity~~
~~+ CSS animations~~
~~+ Responsivity handled by browser~~
~~+ Accessibility~~
~~+ Server side rendering~~

+ Uses GPU → really really fast
+ Full control
– so what's the downside?

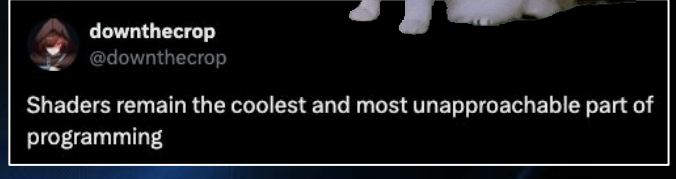
Interactive comparison tool

https://rendering-methods.enchart.me/

Shaders?

```
vec2
p=(FC.xy*2.-r)/r.y,l,i,v=p*(l+=4.-4.*abs(.7-dot(p,p)));for(;i.y++<8.;
o+=(sin(v.xyyx)+1.)*abs(v.x-v.y))v+=cos(v.yx*i.y+i+t)/i.y+.7;
o=tanh(5.*exp(l.x-4.-p.y*vec4(-1,1,2,0))/o);
```

HUH?



https://twigl.app/?ol=true&ss=-OJyw73KFafCZX9RndXH

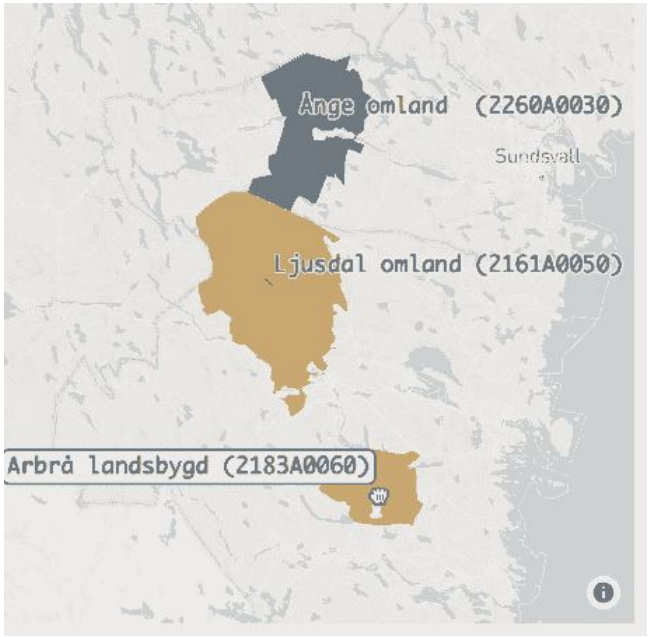
I started WebGL learning here

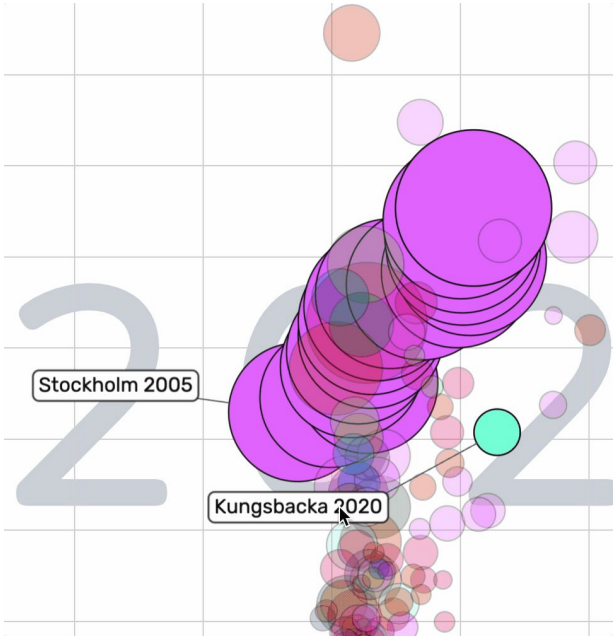
https://observablehq.com/@angiehjort/learning-webgl

WebGL in 100 seconds

Drawbacks of working with pixels directly (canvas, webgl)

When your pixels don’t do the thing, you must do the thing yourself





What options are there?

scope: web based, interactive

- SVG
- Canvas
- WebGL

- Abstraction libraries made for more ergonomic use of any of the above
  - SVG -> d3.js -> Observable plot
  - DeckGL, three.js, Regl for WebGL

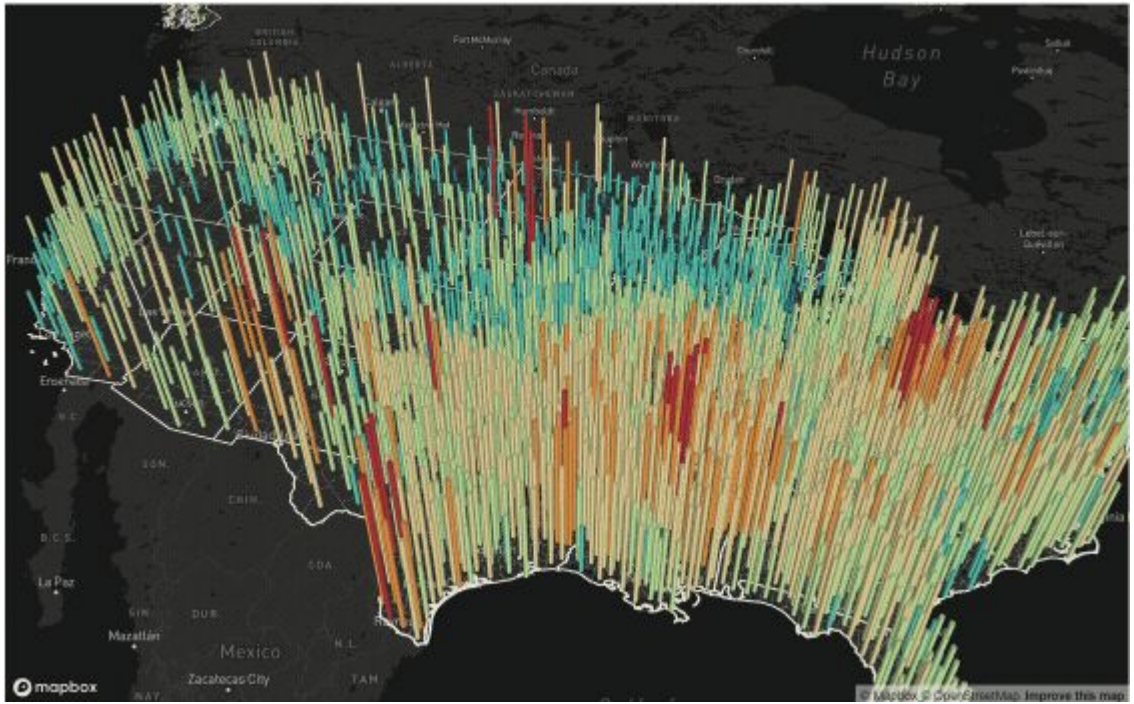
+ Uses GPU → really really fast
+ Approachable
+ Free, open source

...

Interactive comparison tool

https://rendering-methods.enchart.me/

Deck.gl example
from previous students + how to



Three.js example

https://github.com/vasturiano/3d-force-graph

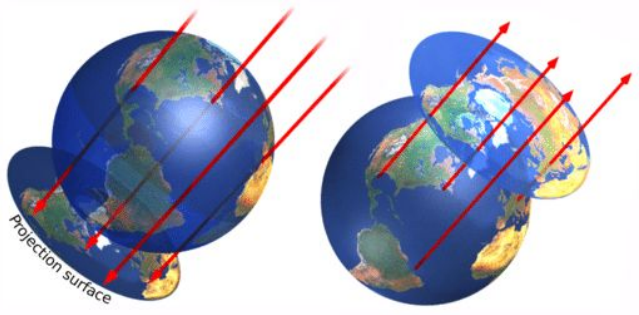
https://200.zona.media/

One more deck.gl example

Drawbacks of abstractions

- Nothing is "one size fits all" Certain tools are easier to customize than others

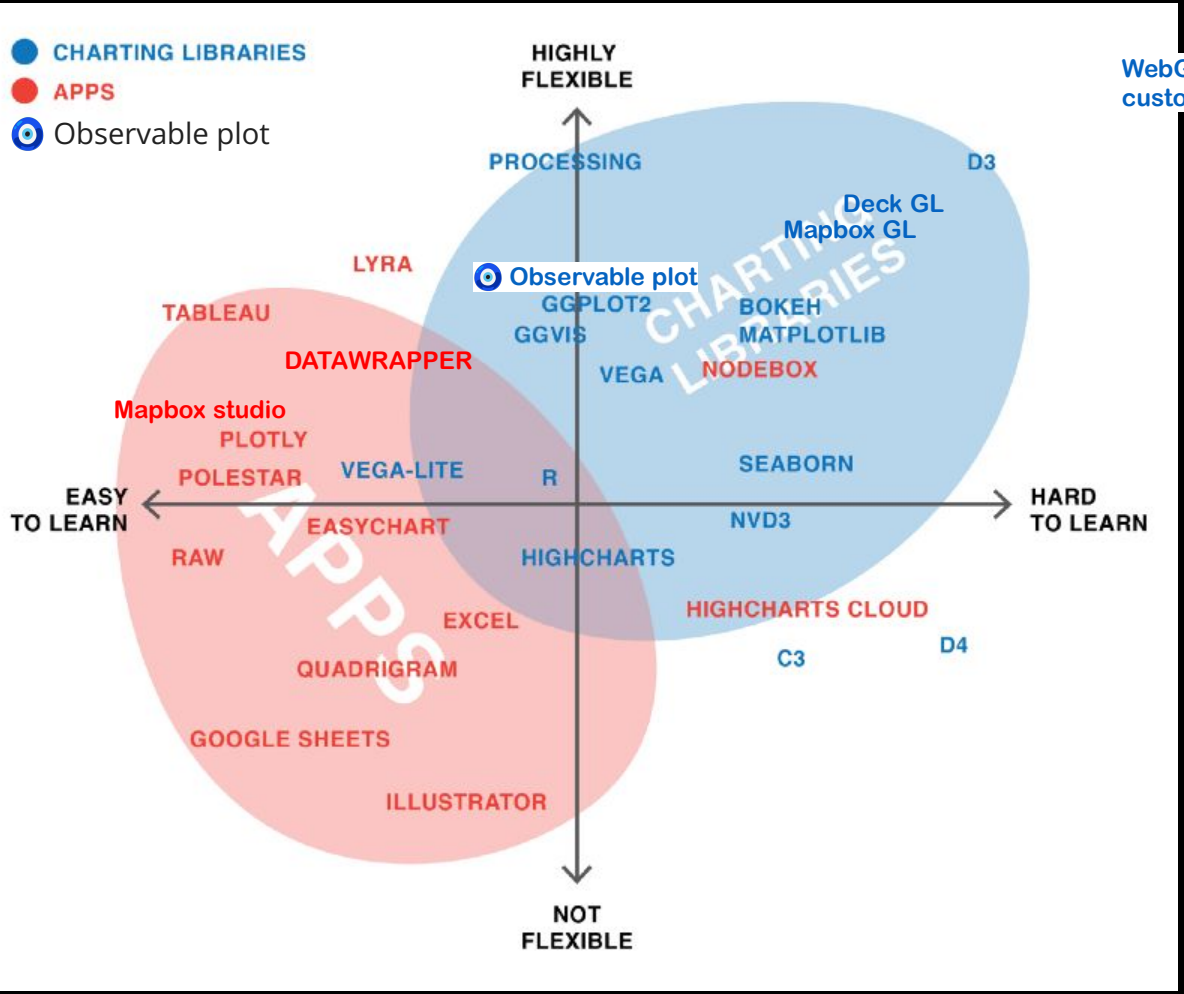
- Stuck in the abstraction Abstractions shield you from the hard stuff, something that's not always helpful. when you want to build custom stuff you discover it’s not made for you




```
function updateDeckView() {
  if (!deckInstance) return;
  deckInstance.setProps({
    width:  cssW,
    height: cssH,
    views: new OrthographicView({
      id: 'ortho'
    }),
    viewState: {
      target: [cssW / 2, cssH / 2, 0],
      zoom:   0,
      minZoom: -Infinity,
      maxZoom:  Infinity,
    },
    layers: [buildLayer()],
  });
}
```

Comparison of different methods

Characteristic	SVG with D3.js	Canvas 2D	WebGL	deck.gl
Sweet spot (points)	≤ 10 000	≤ 100 000	≤ 10 000 000	≤ 5 000 000
Learning threshold	Low	Medium	High	Medium
Memory usage	High (1 DOM node / point)	Low	Low	Low
CPU per frame	High (layout + draw)	High (draw calls)	Low	Low
GPU usage	None	Blit only	Full rasterisation	Full rasterisation
Hover / hit-testing	Free (CSS)	Manual	Manual (GPU pick)	Built-in
Animations	CSS transitions D3 transitions	Manual rAF loop	Manual shaders	GPU transitions
Zoom & pan	Via D3 zoom	Manual	Manual	Built-in
Code size (this demo)	75 lines	269 lines	407 lines	249 lines
Best for	Dashboards, small data	Custom drawing, games	Big data, science viz	Geospatial, big data



WebGL with custom shaders

Flexible tools allow very creative, complex and unique projects but the effort of using them is very high and you may get lost in the options

Video:

D3.js in 100 seconds

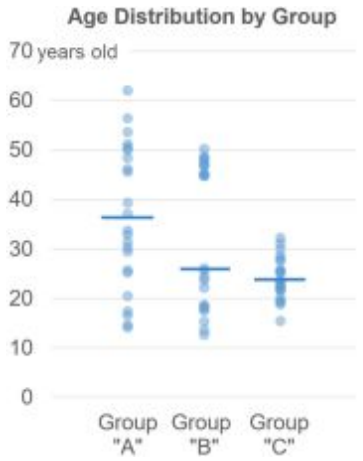
WebGL in 100 seconds

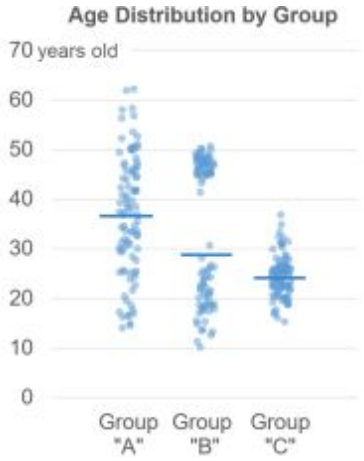
Source: Lisa Charlotte Rost, What i learned recreating one chart using 24 tools

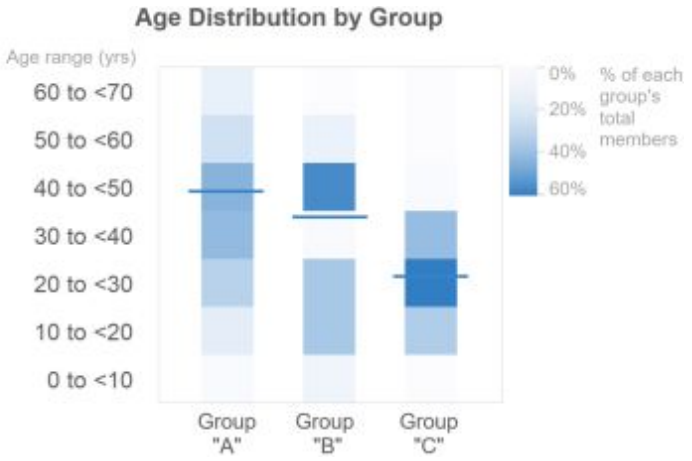
So what did we gain?

More datapoints = more understanding?

More datapoints = more trust?







source