Webteknologi

Modul 8

d3.js - Animationer

Dagens plan

- 08.30 Opfølgning på datavisualisering
- 09.00 Introduktion til animationer
- 09.15 Animationer på data
- 09.30 Øvelser i animationer
- 10.00 Pause
- 10.30 Animationer hvor data tilføjes / slettes
- 10.45 Animationer hvor data sorteres
- 11.00 Fortsætte med øvelser
- 12.00 Slut

Animationer i d3.js

Introduktion

Animationer er en generel teknik hvor man lave ting der bevæger sig ved at ændre i deres datagrundlag over tid. Helt simpelt er det sådan noget med at ændre på figureres position, størrelse, farve, eller anden synlig egenskab.

De gør visualiseringer mere interessante at se på. Og de kan kobles sammen med interaktion til at skabe flotte og interessante datavisualiseringer, hvor brugeren selv kan dykke ned og undersøge data på forskellige måder.

d3.js har funktioner indbygget der gør det nemt at skabe animationer ud fra data.

Animationer i d3.js

En animation i d3.js er en eller flere **transitioner** der **interpolere** imellem nogle start-værdier og slut-værdier. Transitioner kører sekventielt, men flere transition på forskellige elementer kan godt køre parallelt.

Begreberne er generelle og findes også i mange andre værktøjer og libraries, fx videoredigering (Adobe Premiere) og spilprogrammering (Unity, Flash, mm).

Mange værktøjer har en visuel tilgang til animationer, men med d3.js gør vi det igennem JavaScript-kode.

Animationer i d3.js

En simpel animation kunne fx være en bold der flytter sig fra venstre mod højre.

En bold kunne være en SVG **circle** med attributterne **cx**, **cy** og **r**.

En transition kan så tage værdien for **cx**, og øge den hen over tid. Resultatet er så at bolden flytte sig langs x-aksen.

En transition består af følgende:

- **Værdier** der skal animeres - fx **cx** på bolden.
- En **duration** (varighed) - fx 2 sekunder.
- En funktion der **interpolere** imellem start-værdien og slut-værdien.

Interpolering vil sige, at man har en funktion i stil med følgende:

```
// Input: 'tid' - Millisekunder siden start
// Return: en værdi imellem 'start' og 'slut'
funktion ease(tid) {
  // TODO: return en værdi imellem start og slut baseret på
  // hvor langt vi er i transitionen (tid)
}
```

I d3.js kalder man interpolering for **ease**. Og man behøver ikke lave dem selv. Man vælger bare en af de mange indbyggede. Og vælger man ikke, bruger den bare easeCubicInOut. Nogle af de andre indbyggede hedder:

- easeLinear
- easeQuad
- easeCubic
- easeExp
- easeSin

Og mange flere. I kan se dem alle her: https://github.com/d3/d3-ease

Der ligger flere eksempler i GitHub-repositoriet.

Start med at kigge i **circle-animation.html** og **kasser-animation.html**.

Det er de to simpleste.

Animationer på data

Introduktion

Der hvor det for alvor begynder at blive interessant, er når man animere med baggrund i konkret data.

Med JavaScript kan man vælge at ændre data løbende, fx på baggrund af brugerinput, og så kan man få d3.js til at animere efterfølgende.

På de følgende slides introduceres nogle eksempler.

Animationer på data

I dette eksempel er der tale om et simpelt scatter plot, men der er tre forskellige sæt data, alle med det samme antal punkter.

Der kan skiftes imellem hvilket datasæt der vises via knapper på siden, og når der skiftes, animeres punkterne fra de gamle koordinater til de nye.

Fil: **scatter_plot_skift_datasæt.html**

Animationer på data

I flere eksempler bruges en **"on click"** event handler til at reagere når brugeren trykker på en knap. I HTML ser knappen sådan her ud:

```
<button id="klik1">Datasæt 1</button>
```

Efterfølgende skrives en callback-funktion der kører, når knappen trykkes:

```
// Knappen vælges via dens id 'klik1'
d3.selectAll("#klik1")
  // Koden herunder køres kun ved tryk på knappen
  .on("click", function (e) {
    console.log("Du har nu trykket på knappen!");
  });
```

Animationer på data

Man kan se hvilken knap der er trykket på sådan her:

```
// Knappen vælges via dens id 'klik1'
d3.selectAll("#klik1")
  // Koden herunder køres kun ved tryk på knappen
  .on("click", function (e) {
    // Find hvilken knap der blev trykket på
    const id = e.target.id;
    console.log(`Der blev trykket på ${id}`);
  });
```

Når data tilføjes eller fjernes

Baggrund

Vi har en ekstra udfordring hvis datasættet modtager nye punkter, eller hvis gamle punkter slettes.

For hvordan animere man et nyt punkt, når der ikke er nogle start-værdier af animere fra?

Og hvordan animere man et punkt der forsvinder?

Når data tilføjes eller fjernes

Baggrund

enter og exit

Det er når data tilføjes eller fjernes at begreberne **enter()** og **exit()** bliver vigtige.

enter() bruges som i tidligere har set, når der skal vises ny data. Vi bruger den hver gang vi binder **ny data**, da den skaber **placeholders** hvor vi kan **appende** ny svg-elementer.

exit() bruges tilsvarende når data forlader datasættet.

Når data tilføjes eller fjernes

Det som rent faktisk sker herunder er, at man **selecter** alle **circles** i **svg**-tagget.

Man **binder** herefter data på **selektionen** med **data()**.

Og med **enter()** vælger man så alle de punkter, der ikke allerede har en **circle**.

```
svg.selectAll("circle")
.data(dataset1)
.enter()
```

Eksemplerne indtil nu har ikke haft data på forhånd, så i praksis har vi altid, med ovenstående kode, valgt alle punkter hver gang.

Når data tilføjes eller fjernes

Hvis man bruger **exit()** fungerer det lidt på samme måde, men her vælger man alle de punkter der er *forsvundet* med den seneste **data()**.

På den måde er der mulighed for at lave en **transition** på de punkter der forsvinder, og kun de punkter.

Se eksemplet i de følgende slides.

Når data tilføjes eller fjernes

I filen **scatter_plot_skift_datasæt_enter_exit.html** er der et eksempel, hvor man kan skifte imellem tre datasæt.

Der er ikke lige mange punkter i hvert datasæt, så nogle gange kommer der nye punkter til, og andre gange forsvinder nogle af punkterne igen, samtidigt med at de resterende punkter skifter position.

Når data tilføjes eller fjernes

Som en del af eksemplet, er der behov for at lave en transition, hvor man både animere:

- Alle de nye punkter
- Alle punkter der var der i forvejen.

Derfor skal man bruge **merge()** til at **flette** de nye punkter sammen med de eksisterende punkter i en fælles **selection**.

Se koden på de efterfølgende slides:

Når data tilføjes eller fjernes

Brug svg-tagget til at finde alle cirkler der findes i forvejen, og bind det nye data til dem.

```
// Vælg cirkler
const updateSelection = svg.selectAll("circle")
  .data(nyData);
```

Hvis vi bruger **enter()** efterfølgende, får vi en **selection** med placeholders til alle de nye punkter der endnu ikke har fået deres egen "circle".

Når data tilføjes eller fjernes

Brug **enter()** og append **circle**. Sæt attributterne på hvert af de nye cirkler.

Bemærk, at ved at bruge **enter()** vælges de nye punkter, og kun de nye.

```
updateSelection.enter()
  .append("circle")
  .attr("cx", function (d) {
    return w / 2;
  })
  .attr("cy", function (d) {
    return h;
  })
  .attr("r", 1)
```

Når data tilføjes eller fjernes

Alle cirkler animeres. Bemærk, at her bruges **merge()** til at flette de nye punkter sammen med de gamle i en fælles **selection**.

```
// Her flettes de nye punkter sammen med de gamle
.merge(updateSelection)
// Og nu animeres alle punkter
.transition()
.duration(1500)
.attr("cx", function (d) {
  return d[0];
})
.attr("cy", function (d) {
  return d[1];
})
.attr("r", 5);
```

Når data tilføjes eller fjernes

exit() bruges til at vælge de punkter der fjernes igen.

Herefter laves en transition på kun de punkter der fjernes.

```
updateSelection.exit()
  .attr("fill", "darkred") // Farven starter som 'darkred'
  .transition()
  .duration(400)
  .ease(d3.easeQuadOut)
  .attr("r", 20) // Cirklerne gøres større
  .remove(); // Hver 'circle' slettes til sidst
```

Når data sorteres

Baggrund

En ting er når data tilføjes eller fjernes.

Men hvad hvis fx søjlerne i en barchart skifter rækkefølge? Her er det jo præcist det samme data, men med en anden sortering. Hvordan kan det vises, og ikke mindst, animeres?

Når data sorteres

Baggrund

Eksempel

Eksemplet der tages udgangspunkt i her er **bar_chart_sortere_højde.html**.

Løsningen er at hvert datapunkt nu skal have en **key**.

```
const dataset =
[
  // key, value, color
  [0, 3.7876024608683223, "#0ee6b5"],
  [1, 95.57905286956687, "#576480"],
  [2, 11.18730761924719, "#015c7e"]
]
```

Key er en slags id, eller index, på hvert datapunkt.
Den skal være unik.

Når data sorteres

Key i datasættet spiller en vigtig rolle. Med en key kan d3.js se hvis et datapunkt har skiftet position, da key er unik og konstant.

Man skal fortælle d3.js hvordan den finder key når man bruger **data()**.

```
svg.selectAll("rect")
  .data(dataset, function (d) {
    // Denne funktion udpeger 'key' i hvert punkt,
    // som er første element i arrayet.
    return d[0];
  })
```

Når data sorteres

I eksemplet laves data tilfældigt med et for loop. I kan derfor ikke finde det præcise data i koden. Og data er forskelligt fra gang til gang.

```
const maxValue = 100;

function randomData(key) {
  return [key, Math.random() * maxValue, randomColor()];
}

// Nu laves et array med 20 tilfældige datapunkter
let dataset = [];
for (let i = 0; i < 20; i++) {
  dataset.push(randomData(i));
}
```

Når data sorteres

Selve sortering af data sker når der trykkes på en knap.

Sorteringen ses her:

```
// Data sorteres med "compare" callback function
dataset.sort(function (a, b) {
  // En positivt return svarer til at 'a' er størst
  // En negativt return svarer til at 'b' er størst
  // Hvis return er 0, er 'a' og 'b' lige store.
  return a[1] - b[1]; // Index 1 er 'value'
});
```

Se mere her: http://mdn.io/Array.prototype.sort

Når data sorteres

Når data er sorteret, laves en ny skaleringsfunktion.

I stedet for en **scaleLinear** bruges der her i eksemplet altid en **scaleBand** til **x-aksen**.

Med en scaleBand har man et bestemt sæt af værdier som domæne, og de forekommer i en bestemt rækkefølge.

Det er den rækkefølge vi kan ændre på med sortering og en ny skaleringsfunktion.

Når data sorteres

Herunder bruger man en ny **xScale** som sikrer at data nu sættes ind i en ny sorteret rækkefølge, og en transition bruges til at animere søjlerne til deres nye positioner.

```
const updateSelection = svg.selectAll("rect")
  .data(dataset, function (d) {
    return d[0]; // Vælge key på hvert punkt
  });
updateSelection
  .transition()
  .duration(2000)
  .attr("x", function (d) {
    // Nu bruges 'xScaleNy' i stedet for den gamle 'xScale'
    return xScaleNy(d[1]);
  });
```

De sidste øvelser

I de sidste opgaver skal I hive fat i tidligere barcharts og scatter plots i har lavet, og så skal I indføre animationer og flere datasæt i disse.

Næste modul i Webtek bliver efter de første 14 dages projektperiode, og vil komme til at handle om at bruge Git i jeres projektarbejde. Modulet efter kommer nok til at handle om nogle mere avancerede teknikker i d3.js, som er relevante for jeres projekt.