
Team 12

Frédéric Blondeel
Dirk Vanbeveren
Bert Van den Bosch

Inhoudsopgave

1	Introductie	2
2	Lokaal testen	2
2.1	Animatie met timers	2
2.2	Problemen met timers	2
2.3	RequestAnimationFrame	3
2.4	Performance	4
2.5	HTTP request frames	4
3	Synchronisatie testen	6
4	Besluit	6

1 Introductie

Animatie neemt veel problemen met zich mee en een goede kennis van enerzijds event loops, tasks, HTTP requests, hardware en software is van groot belang. Anderzijds is de synchronisatie tussen de clients ondereen essentieel om een zo vloeiend mogelijk beeld te krijgen. De testen die werden uitgevoerd zouden een goede basis moeten vormen voor de verdere implementatie methodes om zo op een correcte manier de animatie te laten lopen.

2 Lokaal testen

2.1 Animatie met timers

Een animatie wordt smooth bevonden als er geen frame ontbreekt tijdens het afspelen. Hoe meer frames er per seconde afgebeeld kunnen worden, hoe smoother de animatie oogt. Het gemiddelde consumer-device heeft een refresh rate van 60Hz wat overeenstemt met 60 frames per seconde (fps). Om een animatie zo smooth mogelijk te laten afspelen, wordt er dus over het algemeen gemikt naar een afspeelsnelheid van 60fps.

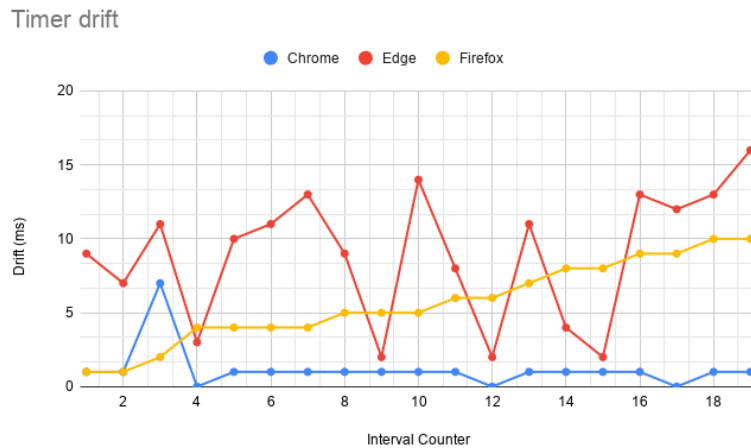
Om elke seconde 60 frames af te beelden moeten de frames volgens een correct getimed interval de volgende frame afbeelden. In javascript zijn hier twee build-in functies voor beschikbaar, `setTimeout()` en `setInterval()`. Op deze manier kan er met een timer om de 16.7ms een nieuwe frame afgebeeld worden om de 60fps playback snelheid te behalen.

2.2 Problemen met timers

Bij het gebruik maken van timers wordt er verondersteld van een playback van 60fps omdat de meeste schermen dit hanteren, maar als er een trager scherm gebruikt wordt om deze animatie af te beelden zullen er overvloedige frames ingeladen worden niet gerenderd zullen worden door de browser. Een laadtijd van 16.7ms of minder voor elke frame is ook niet gegarandeerd en wordt verder besproken in [2.4](#).

Drift in timers Deze timers zouden er voor zorgen dat er elke gerenderde frame door de browser, de juiste frame van de animatie bevat. Dit kan niet aangenomen worden aangezien er altijd een kleine fout zal bestaan op de wachttijd tot de executie van de opgeroepen functie van de timer. Bij het uitvoeren van een test gebaseerd op [\[2\]](#) valt te constateren dat veel populaire browsers de kleine fout niet compenseren en daardoor begint op te tellen en als gevolg de globale timing van de animatie volledig zijn synchronisatie verliest.

MAYBE MEER BROWSERS?

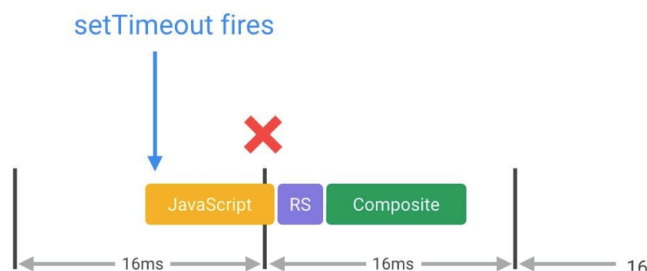


drift.png

Figuur 1: Timer drift van verschillende populaire browsers

Door het continu opschuiven van frame updates kan er een framedrop voorkomen door redundante stappen van de animatie in te laden tijdens de foute frame waardoor de animatie stappen verliest en haperend ervaren wordt.

Single threaded Javascript wordt single-threaded in de browser uitgevoerd. Door deze limitatie kan een taak die getimed staat over x aantal milliseconden pas uitgevoerd worden op de main thread als de event loop op dat moment beschikbaar is om die functie dan uit te voeren. Als er tussen het initialiseren van de timer en het uitvoeren van deze geplande taak een andere taak tussenbeide komt die langere executietijd heeft dan de geplande tijd zal de uitvoering uitgesteld worden tot de event loop terug vrij komt. Ook dit is een deuk in de betrouwbaarheid van timers en kan voor framedrops zorgen als de executie te lang wordt uitgesteld tot voor de nieuwe repaint.



Figuur 2: Framedrop door gebruik van setTimeout. (Afbeelding van Google Web Fundamentals [4])

2.3 RequestAnimationFrame

Een methode die moderne browsers bieden om animaties te updaten met javascript is `requestAnimationFrame()`. Het wordt in dezelfde context gebruikt als `setTimeout()`, maar lost enkele cruciale problemen op. Ten eerste moet er geen tijdsinterval meer gespecificeerd worden als parameter, maar syncd de browser de executie van deze taak met de refreshrate van de monitor. De opgegeven callback functie wordt door de event loop net voor de repaint van de pagina uitgevoerd. Hierdoor worden enkel frames geüpdatet voor

een repaint en zullen er geen redundante taken uitgevoerd worden per frame of framedrops voorkomen. [3]

2.4 Performance

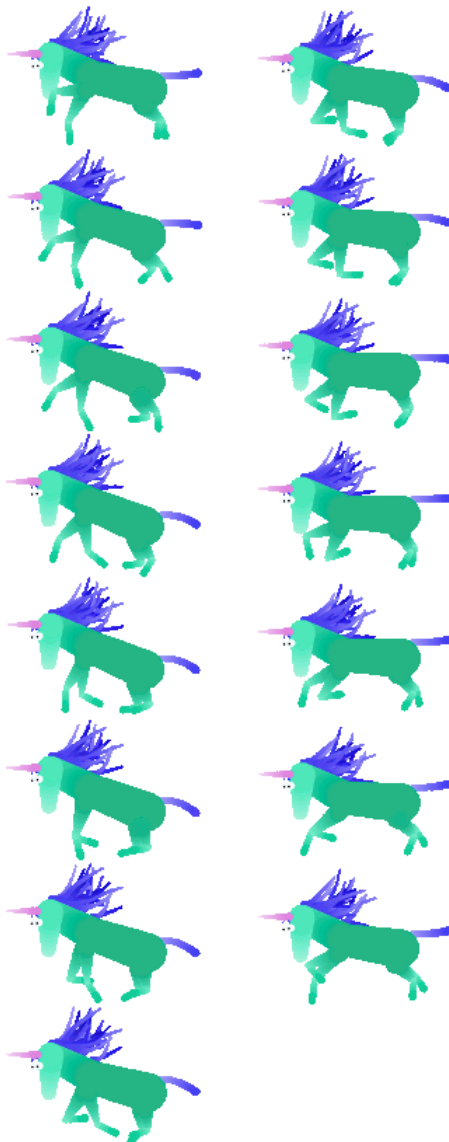
Deze methode lost een groot deel van globale timing op, maar garandeert daarom geen smooth animation playback. We gaan er verder vanuit dat er geen devices een hogere refresh rate dan 60Hz verwachten en mikken dus op een playback snelheid van 60fps. Als de methode om de frame te updaten langer duurt dan 16.7ms zal de frame niet gedropt worden, maar uitgetrokken. Alle frames worden wel weergegeven, maar niet aan een snelheid van 60fps wat dus ook oogt als een niet vloeiende animatie.

TODO: performance bekijken met artif. workload (busy wait) en fps.

Om een pagina zo responsief mogelijk te houden, moet het aantal reflows geminimaliseerd worden. Er zijn nog verscheidene elementen die de reflow snelheid beïnvloeden zoals de diepte en afhankelijkheden van nodes in de DOM-tree van de webpagina. Aangezien er in dit geval enkel gewerkt wordt met een enkelvoudige animatie op een webpagina gaan we hier niet verder op in. [1]

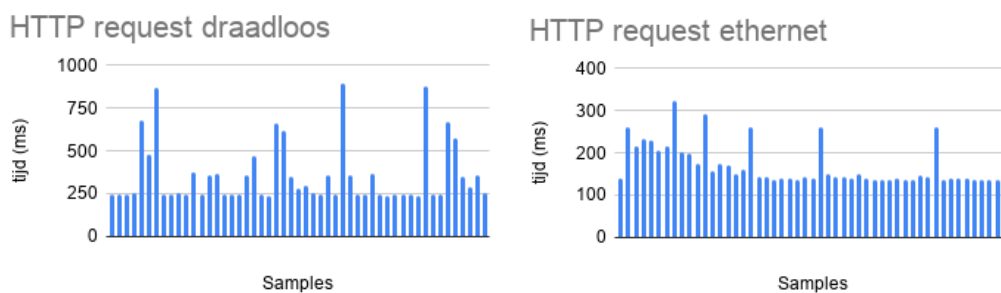
2.5 HTTP request frames

Voor deze test werd gebruik gemaakt van een sprite sheet met 15 frames op (figuur 3) en een animatie snelheid van 30 fps.



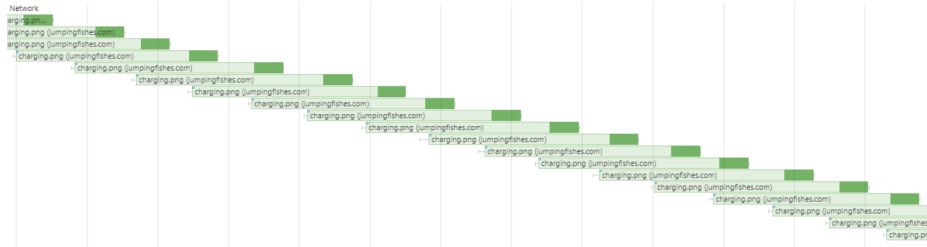
Figuur 3: Sprite sheet met 15 frames (Figuur van [5])

Bij de eerste test werd de hele sprite sheet per frame ingeladen. Dit is om het effect te creëren dat de server elke frame genereert en doorstuurt naar de cliënt. Dit bleek zeer inefficiënt te zijn doordat de tijd om de frame op te halen met ethernet rond de 140 ms en met draadloos internet rond de 250 ms ligt (figuur 4a en figuur 4b).

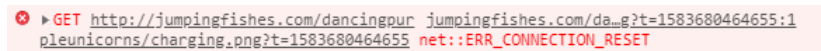


(a) HTTP request met draadloze verbinding (b) HTTP request met ethernet verbinding

Hoewel de frames asynchroon worden ingeladen (figuur 5) is de animatie alles behalve ideaal. Door lange wachttijden zijn er veel frame skips en door de vele http requests wordt de verbinding gereset om de server niet te overbelasten met 30 requests per seconde (figuur 6).

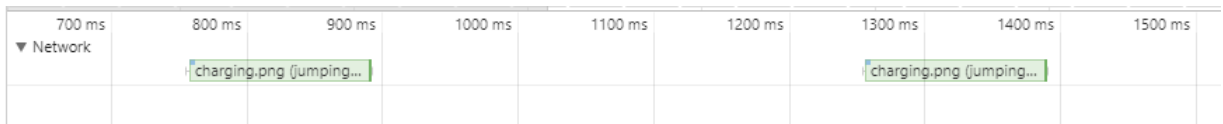


Figuur 5: Inladen van elke frame



Figuur 6: Connection reset door teveel requests

Bij de tweede test werd gebruik gemaakt van een buffer. De sprite sheet werd om de 0.5 seconden ingeladen zodat elke frame tijd had om te displayen ($30 \frac{\text{frames}}{\text{sec}} \frac{1}{15 \text{frames}} = \frac{0.5}{\text{sec}}$). Dit creëert het effect dat de server 15 frames genereert en doorstuurt in 1 keer i.p.v. frame per frame. Dit is duidelijk een verbetering want de server krijgt 15 keer minder requests (figuur 7) en er zijn geen frame skips meer. Na het inladen van de sheet zal de cliënt de 15 frames achter elkaar drawen zonder te hoeven wachten op de server. Daarna zal de volgende sheet (of buffer) gebruikt worden.



Figuur 7: Inladen van de buffer

3 Synchronisatie testen

4 Besluit

Referenties

- [1] Craig Buckler. 10 ways to minimize reflows and improve performance, 2015. <https://www.sitepoint.com/10-ways-minimize-reflows-improve-performance/>, Last accessed on 09-02-2020.
- [2] cTn-dev. setInterval keeps drifting over time, 2018. <https://github.com/nodejs/node/issues/21822>, Last accessed on 08-02-2020.
- [3] Mozilla. Window.requestAnimationFrame(), 2020. <https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame>, Last accessed on 08-02-2020.
- [4] Paul Lewis. Optimize javascript execution, 2019. <https://developers.google.com/web/fundamentals/performance/rendering/optimize-javascript-execution>, Last accessed on 08-02-2020.
- [5] StackOverflow. Sprite sheet van stackoverflow, 2019. <https://stackoverflow.com/questions/9486961/animated-image-with-javascript>, Last accessed on 10-03-2020.