— Verslag P&O CW 2019–2020 Taak ST1.1 —

Department Computerwetenschappen – KU Leuven **5 mei 2020**

Team 12

Frédéric Blondeel
Dirk Vanbeveren
Bert Van den Bosch

Inhoudsopgave

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 te implementeren methodes om zo op een correcte manier de animatie synchroon te laten lopen over verschillende devices. Eerst worden er algemene methodes getest en vergeleken om animaties af te beelden met JavaScript in de browser. Deze werden dan verder gebruikt en uitgebreid om een animatie synchroon te laten lopen over verschillende apparaten.

2 Frame testen

2.1 Animatie met timers

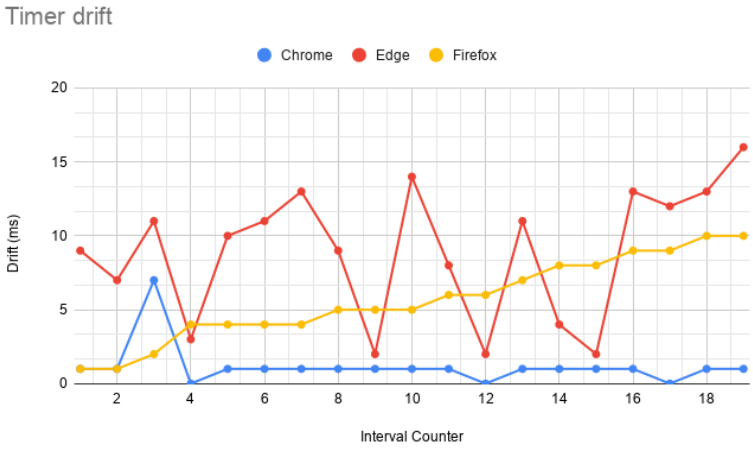
Een animatie wordt vloeiend bevonden als er geen frame ontbreekt tijdens het afspelen. Hoe meer frames er per seconde afgebeeld kunnen worden, hoe vloeiender de animatie oogt. Het gemiddelde consumer-device heeft een refresh rate van 60Hz wat overeenstemt met 60 frames per seconde (fps). Om een animatie zo vloeiend 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 twee build-in functies beschikbaar, `setTimeout()` en `setInterval()`. Aan de hand van deze functies 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

Assumptie 60Hz Bij timers moet het interval expliciet bepaald worden. De meeste devices hebben een display van 60Hz en wordt er een interval van 16.7ms gekozen. Als deze code gebruikt wordt op een scherm met een lagere of hogere refresh rate uitgevoerd worden, zou de animatie te veel of respectievelijk onvoldoende frames renderen.

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.



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 vrijkomt. 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.

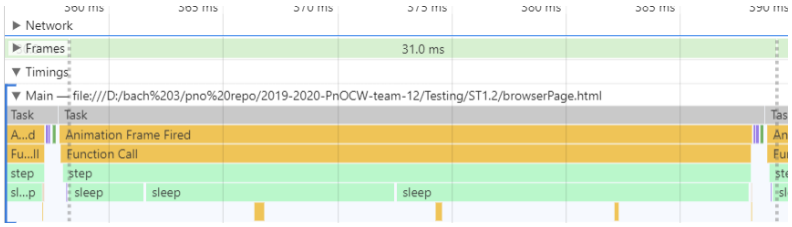
2.3 RequestAnimationFrame

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 gespecifieerd worden als parameter, maar synchroniseert 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. [4]

2.4 Invloeden FPS veranderingen

2.4.1 Rekentijd

JavaScript wordt single threaded uitgevoerd (zie 2.2). Als taken een lange rekentijd vragen, worden andere taken uitgesteld om uitgevoerd te worden. Ook de repaint van de volgende frame wordt uitgesteld als de main thread nog bezet is. Dit fenomeen is eenvoudig te reproduceren met een busy loop van bijvoorbeeld *30ms*. Zoals te zien op figuur 2 wordt de repaint uitgesteld tot de thread klaar is waardoor de frame uitgetrokken wordt en in dit geval tot *31ms* wat overeenstemt met een afspeelsnelheid van 32fps.



Figuur 2: Repaint wordt uitgesteld tot main thread vrij is.

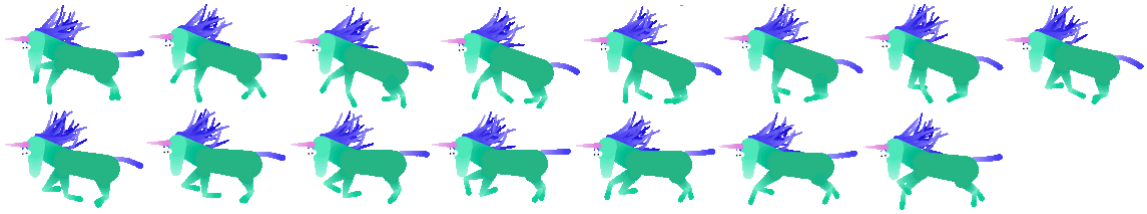
2.4.2 Rendertijd

Voor elke frame dat weergeven word moet de browser alle pixels tekenen met gebruik van de GPU. Dit kan soms veel tijd innemen waardoor de framerate drastisch kan verlagen. Hierbij kan de rendertijd ook niet altijd hetzelfde op elk toestel vanwege de verschillen in hardware.
Dit is getest door een animatie te tekenen met een gegeven aantal cirkels. Bij een aantal van 100 cirkels hadden de geteste toestellen allemaal ongeveer 60FPS. Dit wilt zeggen dat het berekenen en het tekenen van de animatie kleiner of gelijk aan 16.7ms is.
Bij het tekenen van 1000 cirkels heeft een van de toestellen nog steeds 60FPS terwijl de andere al gezakt is. In de profiler is het zichtbaar dat de meeste tijd niet in het berekenen van de posities is, maar wel in het tekenen zelf.
Hierbij kan geconcludeerd worden dat het belangrijk is om niet te veel GPU intensieve animaties te maken om er voor te zorgen dat de framerate zo hoog mogelijk is.

Figuur 3: Profiler voor animatie met 3000 cirkels

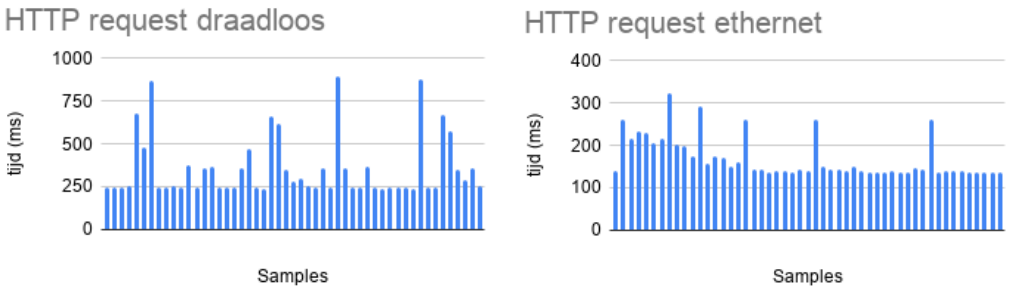
2.4.3 HTTP request frames

Voor deze test werd gebruik gemaakt van een sprite sheet met 15 frames op (figuur 4) en een animatie snelheid van 30 fps.



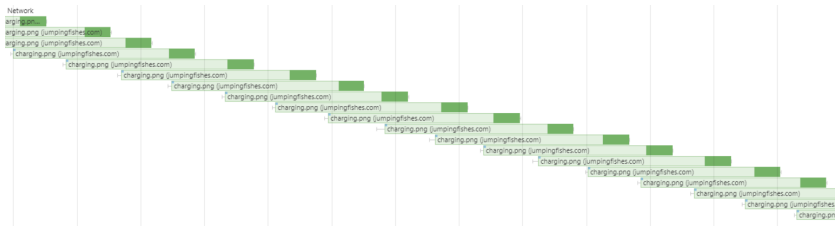
Figuur 4: Sprite sheet met 15 frames (Figuur van [5])

Bij de eerste test werdt 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 *140ms* en met draadloos internet rond de *250ms* ligt (figuur 5a en figuur 5b).

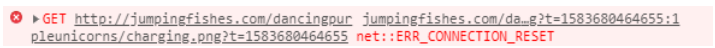


(a) HTTP request met draadloze verbinding (b) HTTP request met ethernet verbinding

Hoewel de frames asynchroon worden ingeladen (figuur 6) 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 7).

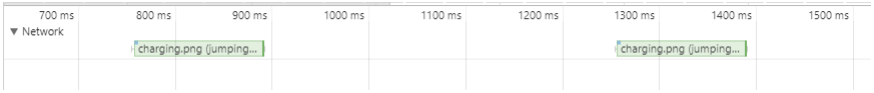


Figuur 6: Inladen van elke frame



Figuur 7: 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 ($30\frac{frames}{sec}\frac{1}{15frames} = \frac{2}{sec}$). 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 8) 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 8: Inladen van de buffer

2.5 Performance

`requestAnimationFrame()` lost een groot deel van globale timing op, maar garandeert daarom geen vlotte animation playback. We gaan er verder vanuit dat er geen devices een hogere refresh rate dan 60Hz utiliseren en mikken dus op een playback snelheid van 60fps. Dit doel kan enkel gerealiseerd worden als de tijd om een frame te berkenen minder dan *16.7ms* duurt. Zoals eerder vermeld in 2.4 zijn er meerdere factoren dat invloed hebben op deze rekentijd.

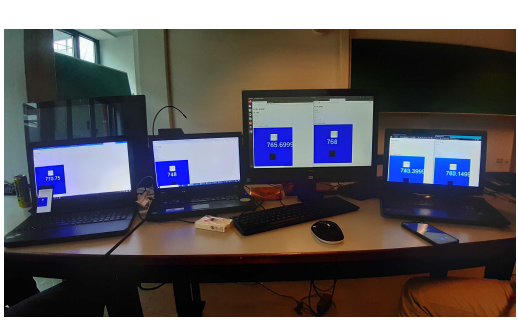
Om een pagina zo responsief mogelijk te houden, moet het aantal reflows en repaints geminimaliseerd worden. Er zijn nog verscheidene elementen die de reflow snelheid beïnvloeden zoals de diepte en afhankelijkheden van nodes in de DOM-tree van de webpagina. Aangezien er in dit geval enkel gewerkt wordt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

3 Synchronisatie testen

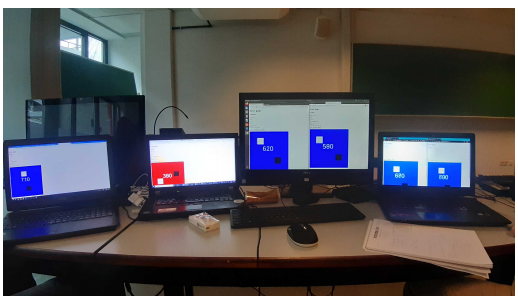
3.1 Frame per frame

In de eerste instantie worden de klokken van de toestellen gesynchroniseerd met de server klok. Dit maakt het mogelijk om de animatie tegelijkertijd te starten op elk scherm. Vervolgens wordt de animatie geüpdate bij elke nieuwe frame. De frames worden hier opgevraagd door de `requestAnimationFrame()` methode. Bij deze test wordt er geassumeerd dat de animaties in een actieve tab (voorgrond) worden uitgevoerd zodat de browser `requestAnimationFrame()` niet afremd voor energiebesparingsredenen [3]. De animatie is zeer envoudig dus wordt wel verwacht dat elk apparaat deze animatie zonder probleem aan 60fps kan afspelen.

Op de starttijd zijn alle schermen synchroon. Er is dan enkel een tijdsverschil dat evenredig is met de precisie van de klok. Na 30 seconden wachten is het al duidelijk zichtbaar dat er sommige browsers voor loopt op anderen. Dit is vanwege de irregulariteit van de framerate op verschillende toestellen dat er voor zorgt dat de animatie iets sneller of trager zal verlopen. Deze techniek is zeker geen goede methode om animaties te synchroniseren.



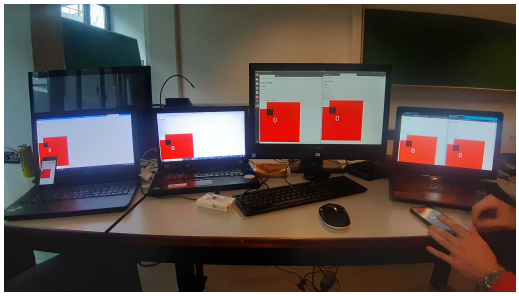
(a) animatie in het begin (gesynchroniseerd)



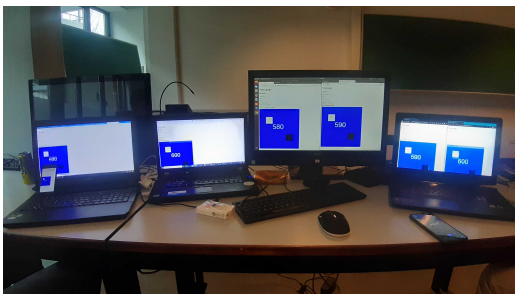
(b) animatie na 30 seconden (niet gesynchroniseerd)

3.2 Klok synchronisatie

Vervolgens is hetzelfde algoritme als de sectie hierboven geïmplementeerd met het verschil dat elke frame dat getekend wordt afhankelijk is van de initieel gesynchroniseerde tijd voor het afspelen van de animaties.Ook hier wordt er geassumeerd dat elk gebruikt apparaat deze eenvoudige animatie aan 60fps kan afspelen.
In deze omstandigheden bleven alle animatie synchroon en had het verschil van frame dus praktisch geen invloed meer. Deze tijd hoefde enkel in de initiële stap gesynchroniseerd worden omdat er in vorige testen al bewezen werd dat de drift van elke klok geen zichtbare invloed zal hebben.



(a) animatie in het begin (gesynchroniseerd)



(b) animatie na 30 seconden (gesynchroniseerd)

4 Besluit

Uit de tests volgt dat animaties niet altijd even snel verlopen op verschillende toestellen. Dit heeft te maken met hoe lang het toestel nodig heeft om een frame te tekenen. Dit heeft te maken met hoeveel berekeningen er gemaakt moeten worden voor een frame, de tijd die de GPU nodig heeft om de frame te tekenen, de tijd die nodig is om http requests te versturen en ook de hardware waarop dat het toestel uitvoert.
Daarnaast zijn er ook nog verschillende manieren om elke frame te starten. Waarbij de `setTimeout` functie bijvoorbeeld meerdere keren de berekeningen maakt voor een stap voordat de frame helemaal getekend is. Dit kan zorgen voor een animatie die veel sneller loopt dan normaal. De oplossing hiervoor is de functie `requestAnimationFrame` dat mogelijk maakt de berekeningen voor de frame te maken voordat die getekend wordt.
Deze redenen zorgen ervoor dat de tijd om een frame te tekenen niet vertrouwbaar is om te synchroniseren. Wat wel mogelijk is is om bij elke frame de juiste positie op de animatie te tekenen gebaseerd op de gesynchroniseerde tijd met de server.

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] Google Chrome Documentation. Background tabs in chrome 57. https://developers.google.com/web/updates/2017/03/background_tabs, Last accessed on 09-02-2020.

[4] Mozilla. Window.requestanimationframe(), 2020. https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame, 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.