Tallinna Ülikool

Informaatika Instituut

HTML elementide animatsioonide optimeerimine

Bakalaureusetöö

Autor : Manuel Vulp

Juhendaja : Andrus Rinde

Autor: ..... „ ..... „2015

Juhendaja: ..... „ ..... „2015

Instituudi direktor: ..... „ ..... „2015

Tallinn 2015

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina _____ (sünnikuupäev: _____)

(*autori nimi*)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
   _____
   _____
   _____

   (*lõputöö pealkiri*)

   mille juhendaja on _____,

   (*juhendaja nimi*)

   säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas/Haapsalus/Rakveres/Helsingis, _____

(digitaalne) allkiri ja kuupäev

Sisukord

Sissejuhatus

Veebi jätkuva kiire leviku ja arenguga kasvavad ka kasutajate ootused, lisanduvad standardid ning uued tehnoloogiad. Järjest enam on veebis näha lehti, kus on rohkelt animatsioone ning paljud elemendid on stiilide rohked.

Käesoleva bakalaureusetöö autori peamiseks ülesandeks tööl on maksimaalsete jõudlustega ning erinevatele platvormidele optimeeritud veebirakenduste loomine. Saavutamaks kompetentsi vastavate rakenduste ehitamiseks, on autor pidanud uurima palju teiste veebiarendajate tehtud lahendusi ning selle põhjalt on välja tulnud järgnev – paljud loodud rakendused ei ole optimeeritud ning ei võta arvesse mitmekülgsete seadmete spetsifikatsioone.

Kuigi rohkete animatsioonidega ning interaktiivsete lahendustega rakendusi võib pidada veebievolutsiooni normaalseks arenguks siis kaasnevad sellega ka mitmed probleemid, millega veebiarendajad peavad tegelema. Erilist tähelepanu nõuab lehe optimeerimine võimalikult paljude, sealhulgas ka väiksema jõudlusega, seadmete jaoks. Näiteks mobiiltelefonide jõudlust ei saa enamjaolt arvutite omadega võrrelda, mistõttu vajavad nad ka teistsugust lähenemist optimeerimise vaatenurgast. Mõistliku lihtsusega saab arendada lahenduse, mis töötab kõige uuematel brauseritel ning võimsatel seadmetel, kuid mille loomise käigus pole arvestatud nõrgemate seadmetega. Optimeerimata animatsioonide puhul võib tulemus jääda hüplev või kõige hullemal juhul mitte töötav.

Käesoleva töö eesmärk on välja selgitada kõige problemaatilisemad küljed HTML elementide animeerimises ning pakkuda välja nii optimaalseid lahendusi kui ka teadmisi, kuidas ohukohti ning kontrollimist vajavaid osasid analüüsida. Töö sobib lugemiseks kõikidel tasemetel veebihuvilistele.

Töö käigus otsiti vastused järgnevatele küsimustele:

- kuidas hinnata veebirakenduse visuaalset jõudlust,
- kuidas parandada veebirakenduse visuaalset jõudlust,
- mida peaks silmas pidama animatsioonirohkete veebilehtede ehitamisel.

Vastuste leidmiseks võrdleb autor erinevaid vahendeid, millega saab veebirakenduse jõudlust mõõta ning analüüsib mitmete raamistike ning metoodikate tulemusel saadud andmeid.

1 Animatsioonid veebilehel

Praegusel ajahetkel on veebi enim levinumateks arenduses kasutatavateks keelteks HTML, CSS ning JavaScript. Kõik rakendused mis on veebiga seotud, nõuavad mingitel arendustsüklite etappides eelnimetatud kolme keele kombinatsioone.

Oskused aga on suhtelised; mõistagi on pea võimatu olla kõikides vajalikes keeltes oskuste tipus, kuid see ei olegi peamine. Pigem on tähtis teada kõige suurema mõjuga osasid veebirakenduse töötamise kohapealt. (Web Standards Project, 1998)

Standardiks kujunenud veebilehtede ülesmärkimise keele HTML 5. versiooni kasutatakse kasutajale kuvatava dokumendi kirja panemiseks tekstina. Brauser töötleb teksti läbi ning loob selle põhjal esialgse dokumendi puu, millele lisatakse seejärel CSS faili(de)st tulenevad stiilid. Valmis joonistatud elementide stiilide muutmist nimetatakse veebis animeerimiseks ning seda saab teha kas JavaScripti või CSS3 abil. JavaScriptis saab elementide animeerimiseks kasutada erinevaid tehnikaid. Näiteks võib animeerida setTimeout või setInterval meetodi abil, mille abil saab muuta elementide stiile pärast etteantud millisekundite möödumist. Meetod setInterval käivitatakse iga etteantud aja tagant nii kaua kuni ta käsitsi peatatakse, setTimeout meetod käivitatakse aga ühe korra. Animeerimiseks peab teda rekursiivselt välja kutsuda, kus igal sammul tuleb muuta elemendi CSS stiile. Uuematel brauseritel on võimalik kasutada ka requestAnimationFrame meetodit, mis võimaldab arendajal öelda brauserile, et soovitud stiili muudatused tuleks realiseerida esimesel võimalikul hetkel; täpsemalt peatükis 1.5. CSS3ga on võimalik elemente animeerida kas lisades transition attribuudi või keyframes abil. Valikuvõimalusi on arendajal mitmeid ning kõigil vahenditel on omad head ja vead. Järgnevates peatükkides antakse ülevaade terminitest ning vahenditest mida peaks teadma, et mõista paremini töö sisu.

1.1 Kaadri- ja värskendussagedus

Kõige olulisem animatsiooni mõõtmiseks kasutatud ühik on kaadrisagedus, mis näitab mitu kaadrit sekundis vaatajale genereeritakse. Inimese silmale on kõige sujuvam animatsioon selline, kus kaadrisagedus on võimalikult kõrge ning konstantne. Animatsioon ei ole silmale sujuv, kui kaadrisagedus on liiga tihedalt muutuv. Levinud näiteks võib tuua filmides kasutatud 24 kaadrit sekundis, mis peaks olema kõige madalam kaadrisagedus, mille puhul inimene ei saa aru, et tegu on piltide näitamisega ning peab seda

animatsiooniks. See tähendab, et inimesele kuvatakse sekundis 24 erinevat pilti, mis moodustavad silmale liikumise. (PC Magazine Encyclopedia, kuupäev puudub)

Värskendussagedus näitab mitu korda sekundis kuvab ekraan graafikamälust uue pildi vaatajale. Näiteks ekraan, mille värskendussagedus on 60 näitab vaatajale maksimaalselt 60 kaadrit sekundis. See aga ei tähenda, et iga näidatud kaader on unikaalne, sest need võetakse ekraani puhvrist ning kui rakendus pole sinna vastava aja sees jõudnud uut pilti laadida värskendatakse juba eelnevalt näidatud pilti. See on ka üks põhjus, miks osad animatsioonid tunduvad väga aeglased – rakendus ei suuda toimetada uusi pilte ekraani graafikamälusse piisavalt kiiresti mistõttu näidatakse vaatajale viimast puhvris leiduvat pilti. (144Hz Monitors, 2015)

Kaadri- ja värskendussagedusel on ka tihe seos – nende kombinatsioonist saab teada mitut unikaalset kaadrit vaatajale näidatakse. Kui rakendus jookseb kaadrisagedusel 30 ning ekraani värskendussagedus on 60 siis näidatakse vaatajale igat unikaalset kaadrit 2 korda sekundi jooksul. Sagedus 60 kaadrit sekundis on ideaalne sihtmärk, sest sel juhul on iga kasutaja poolt nähtud kaader unikaalne tagamaks silmale sujuva animatsiooni.

Saamaks teada kui kiirelt peab brauser suutma lõpetada oma tegemistega säilitamaks sujuva kasutamise kogemuse, tuleb jagada 1000 (millisekundit) soovitava kaadrisagedusega. Kui sihtmärk on 60 kaadrit sekundis, peavad brauseri arvutused, dokumendi elementide paigutamine ja selle värvimine võtma iga kaadri kohta maksimaalselt aega 16,7 millisekundit.

1.2 Animatsioonide jõudluse mõõtmine

Üks osa veebilehe animatsioonide optimeerimisest on jõudluse jälgimine ja mõõtmine, mille jaoks on võimalikult informatiivset vahendit vaja. Näiteks võib rakendus graafikaprotsessori mälu liigselt tarbida, JavaScripti valitud dokumendi mudeli muutmise metoodika võib blokeerida kasutaja sisendit, rohkete HTML tükkide animeerimine võib olla aeglane või midagi muud.

Üks efektiivsemaid metoodikaid probleemide avastamiseks ja lahendamiseks on kasutada korralikke analüüsimise tööriistu, mis ka levinumate brauseritega kaasa tulevad.

- Google Chrome'i jaoks The Chrome Developer Tools,
- Firefox'i jaoks Firefox Developer Tools.

Antud töös on analüüsimiseks kasutatud Google Chrome brauserit ning selle analüüsimise tööriista DevTools'i, sest selle abil on võimalik väga detailselt näha, mis veebilehel toimub.

1.2.1 The Chrome Developer Tools

The Chrome Developer Tools, lühendatult DevTools, on veebi analüüsimise ja mõõtmise tööriistade komplekt, mis annab arendajale võimaluse detailselt uurida brauseri poolt teostatavat tööd. Seda saab kasutada efektiivseks koodis esinevate jõudlust vähendavate probleemide leidmiseks. (Google Chrome, kuupäev puudub)

Meetodeid kuidas DevTools avada, on mitmeid. Näiteks saab konsooli avada vajutades F12, klahvikombinatsiooni Ctrl + Shift + I (Windows) või Cmd + Option + I (Mac) või paremat klahvi dokumendi peal ning seejärel valida kõige viimane valik Inspect Element.

Vaade, mis avaneb, peaks olema sarnane joonisel 1 kujutatud pildiga.

Joonis 1. Chrome DevTools

Animatsioonide optimeerimise seisukohast on kõige olulisem tööriist ajajoon, mille avamiseks tuleb valida tööriista rea pealt Timeline.

1.2.2 Ajajoon (Timeline)

Ajajoone paneel (Joonis 2) annab täieliku ülevaate sellest, mille peale kulub veebirakenduse näitamisel või kasutamisel aega.

Kõik sündmused, ressursside laadimine, JavaScripti tõlgendamine, dokumendi stiilide arvutamine ning värvimine, on vaates kuvatud ning võimaldatud on ka nende süvaanalüüs. (Google Chrome, kuupäev puudub)

Joonis 2. Chrome DevTools, ajajoone vaade

Töös kasutatud osad ajajoone vaatest:

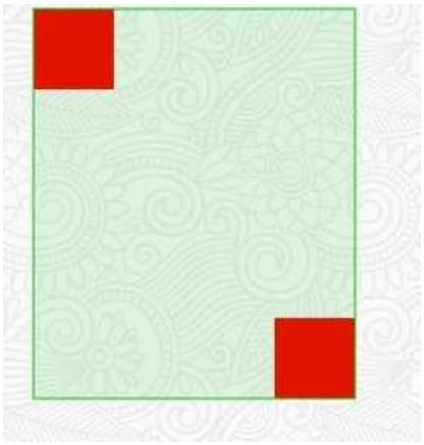
- Sündmuste salvestamise nupp (Record). Vajutades seda ühe korra, muutub nupp punaseks ning Chrome hakkab salvestama sündmusi, mis juhtuvad veebirakenduses. Uuesti vajutamisel lõpetatakse salvestamine ning kuvatakse salvestuse alustamise ning lõpetamise vahelise aja sündmusi.
- Kaadrite vaade (Frames view). Aktiveerituna on sinist värvi ning kuvab arendajale rakenduses toimuvaid sündmusi kaadrite põhiselt. Iga ajajoonel asuv blokk näitab kui kaua aega läks ühe kaadri loomiseks. Mida vähem aega kõik tegevused võtavad, seda kiirem on veebileht. Ideaalis peaksid kõik brauseri poolt sooritatud tegevused võtma ühe kaadri raames aega alla 16,7 millisekundi, sest siis jookseb veebileht kaadrisagedusel 60. Ajajoonel asuvad kaks kriipsu, millest ülemine tähistab 30 ning alumine 60 kaadrisageduse piiri. Bloki värvid kujutavad tegevusi, mis selle kaadri loomiseks aega võtsid.
- Salvestatud tulemused. Kuvatakse järjekorras tegevusi, mida brauser pidi tegema, kaua iga tegevus aega võttis ning mis seda põhjustas. Vajaminevaid tegevusi on kirjeldatud peatükis „Veebilehe visualiseerimine“.

1.2.3 Lisa tööriistad animatsioonide jõudluse mõõtmiseks

DevTools pakub ka vaheneid täiendavaks analüüsiks. Selleks tuleb vajutada DevTools’i vaates klahvi ESC, mis avab paneeli (Joonis 6), kus saab määrata lisa analüüsimise tööriistu. Olulisemad nende hulgas on:

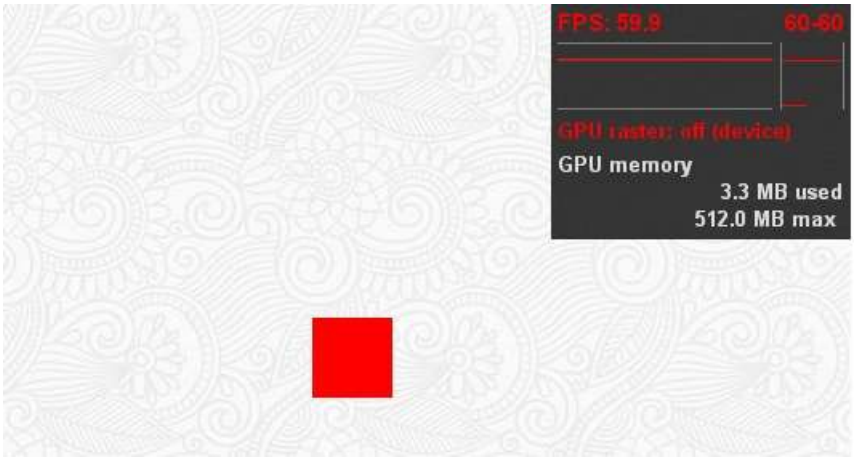
- Show paint rectangles – Märgib veebilehel rohelise ristkülikuga ala, mis vajab brauseri poolt uuesti joonistamist. Selle põhjal on hea näha kuidas elementide stiilide muutmine põhjustab brauseris alade uuesti joonistamist. Joonisel 3 muudeti

punase kasti positisiooni, mille tõttu pidi brauser joonistama terve rohelise kasti sisu uuesti, sealhulgas ka taustapildi.



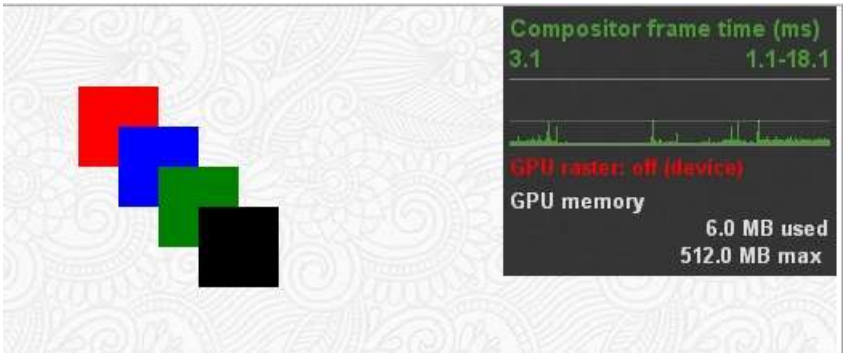
Joonis 3. Uuesti värvitava ala näide

- Show FPS meter – Joonistab brauseri paremale ülesse nurka väikse akna, kus kuvatakse arendajale hetkel olevat kaadrisagedust ning rakenduse graafikaprotsessori mälukasutust. Joonisel 4 animeeritakse punase kasti positsiooni ning tööriist näitab, et hetkel on brauseri poolt kuvatavaks kaadrisageduseks sellel veebilehel 60.



Joonis 4. Animatsiooni kaadrisagedus

- Enable continuous page repainting – Joonistab brauseri paremale ülesse nurka väikse akna, kus kuvatakse kaua võtab praeguse dokumendi ühe kaadri joonistamine millisekundites aega. Lisaks kuvatakse ka hetkel olevat kaadrisagedust. Joonisel 5 võtab selle veebilehe ühe kaadri joonistamine aega umbes 3.1 millisekundit.



Joonis 5. Veebilehe visuaalse osa joonistamise ajakulu

Joonis 6. Lisapaneel

1.3 Veebilehe visualiseerimine

Järgnevalt annab autor ülevaate veebilehe visualiseerimise protsessist ehk sellest, kuidas brauser tõlgendab HTML faili ning loob selle põhjal kasutajale lõpliku lahenduse.

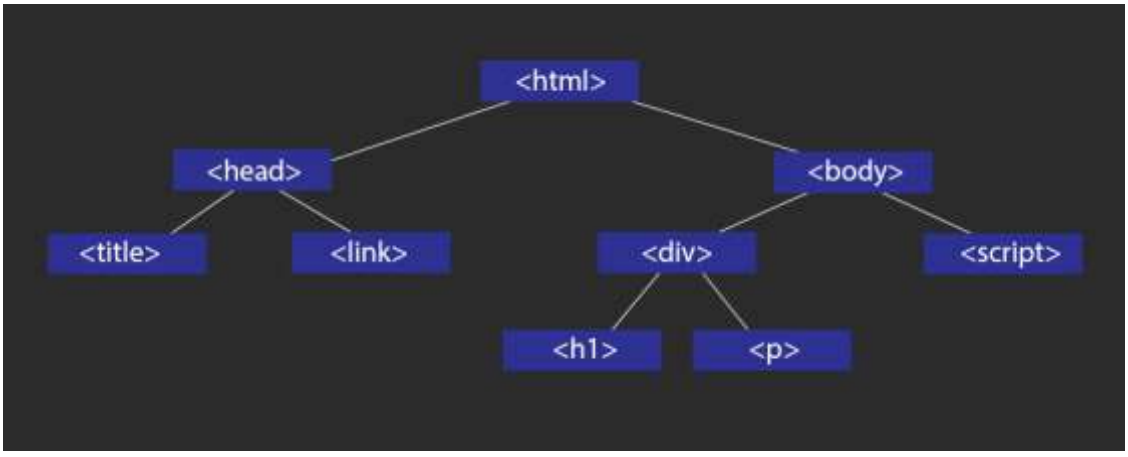
1. Brauser laeb alla HTML faili.

2. Brauser käib üle terve HTML faili (Koodinäide 1) ning ehitab saadud andmete põhjal kokku esialgse HTML puu (Joonis 7), mis sisaldab kõiki HTML elemente ning nende sõltuvusi, edaspidi nimega dokument. Antud protsessiks kulunud aega on Chrome DevTools'i ajajoone peal võimalik jälgida Parse HTML nimega paneeli alt. Näide:

```
<!doctype html>
<html>
<head>
    <title>HTML elementide animatsioonide optimeerimine</title>
    <link rel="stylesheet" href="assets/style.css">
</head>
<body>
<div id="content">
    <h1>Pealkiri</h1>
    <p>Sisu</p>
</div>
<script src="bower_components/requirejs/require.js" data-
main="app/index.js"></script>
</body>
</html>
```

Koodinäide 1. HTML kood

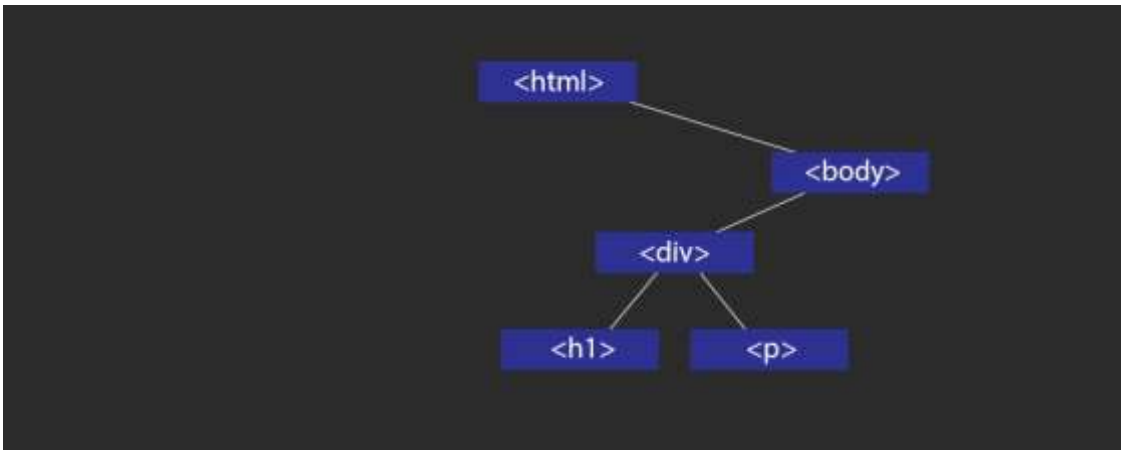
Koodinäite 1 põhjal loob brauser esialgselt järgneva dokumendi, kus igast elemendist hargnevaid harusid nimetatakse alampuudeks :



Joonis 7. HTML koodi põhjal brauseri poolt kokku pandud dokument

3. Dokumendi elementidele lisatakse CSS'i poolt määratud stiilid. Selle sammuga tehakse dokumendist valmis uus puu (Joonis 8), mis sisaldab ainult joonistamist vajavaid elemente. Lisaks eemaldatakse kõik elemendid, mille kohta on CSS'i

poolt defineeritud stiili reegel display: none. Antud protsessiks kulunud aega on Chrome DevTools'i ajajoone peal võimalik jälgida Recalculate styles nimega paneeli alt.



Joonis 8. HTML dokument, kus on ainult lehel visuaalselt nähtavad elemendid

4. Elemendid asetatakse brauseri vaatepinnale. Brauser peab selles sammus arvutama välja iga elemendi täpse geomeetrilise positsiooni, mis on väga töömahukas protsess. Kui kasutaja muudab koodi abil mõnda elemendi positsiooni mõjutavat parameetrit, tuleb tihtipeale terve dokumendi elementide positsioonid uuesti kalkuleerida. Antud protsessiks kulunud aega on Chrome DevTools'i ajajoone peal võimalik jälgida Layout nimega paneeli alt.
5. Kõik elemendid on algselt vektorgraafikas, kuid selleks, et kuvada neid brauseri vaatepinnale, on igal brauseril rasterdaja, mis muudab vektorkujutised nähtavateks piksliteks ekraanipinnale. Elemendid muudetakse rasterkujutisteks, pildid dekodeeritakse ning nende suuruseid muudetakse vastavalt stiilinõudmiste järgi. Algselt ei pruugi pildid olla vektorgraafikas ning selleks on vaja brauseril alla laetud pilt dekodeerida, vajaminevasse suurusesse viia ning seejärel ekraanipinnale pikslitena joonistada. Ka see samm on väga kulukas protsess. Antud protsessiks kulunud aega on Chrome DevTools'i ajajoone peal võimalik jälgida Paint nimega paneeli alt.
6. Elemendid jaotatakse ära kahte sorti graafikakihtidele: Joonistamiskihid (RenderLayer), mille töö on hallata dokumendi alampuid (mainitud punktis 2) ning graafikakihid, mille töö on hallata joonistatud elementide tõstmist graafikaprotsessorile. Graafikakihid laetakse arvuti põhimälust (RAM) graafikaprotsessori mälusse (VRAM) tekstuuridena ehk rasterdatud piltidena.

Antud protsessiks kulunud aega on Chrome DevTools'i ajajoone peal võimalik jälgida Layout nimega paneeli alt. (Wiltzius, 2013)

1.4 Riistvaraline kiirendus veebilehe visualiseerimisel (hardware acceleration)

Suurem osa brauseri arvutustest toimub arvuti protsessori peal. Riistvaraliseks kiirenduseks nimetatakse antud töö kontekstis seda, kui mingi osa tehtavast arvutusteks tehakse arvuti protsessori asemel graafikaprotsessoril. Riistvaraline kiirendus on hea, sest siis jäävad paljud keerukad arvutused graafikaprotsessor kanda, mis vähendab arvuti keskprotsessori koormust. (Lewis, 2013)

Graafikaprotsessori kasutamine pakub suurt jõudluse eelist, kuid kaasaskantavatel seadmetel pole väga praktiline rohke mälukasutuse tõttu. (Liu, Granados, Duarte, & Andrian, 2012)

1.5 Kaadri joonistamine brauseri poolt (requestAnimationFrame)

Mitmetel brauseritel (IE10+, Firefox 31+, Chrome 31+, Safari 7+, Opera 27+, iOS Safari 7.1+, Android Browser 4.4+, Chrome for Android 41+) on olemas globaalne meetod requestAnimationFrame, mis võimaldab arendajal öelda veebilehitsejale, et soovib luua animatsiooni, mille stiilide muudatused tulevad kaasaantud meetodist. Selle eelis seisneb stiilide muudatuste realiseerimise brauseri poolt kõige optimaalsemal hetkel. (Mozilla Developer Network, kuupäev puudub) (Can I use, 2015)

JavaScripti Koodinäide:

```
var width = 0;
var test = document.getElementById('box');
function animate() {
    requestAnimationFrame(animate);
    test.style.width = width.toString() + 'px';
    width += 1;
}
requestAnimationFrame(animate);
```

Koodinäide 2. requestAnimationFrame kasutuse näide

Näidiskoodis animeeritakse HTML elemendi laiust igal kaadril ühe piksli võrra suuremaks. Brauser sünkroniseerib selle kaadri joonistamise ülejäänud joonistamistega ise, tagamaks sujuvama animatsiooni.

1.6 Tüüpilised veebianimatsioonidega seotud probleemid

Rohkelt kasutatakse veebilehtedel nähtavate animatsioonide loomiseks JavaScripti, mille abil muudetakse elementide stiilide väärtusti. Selle kasutamisel tuleb aga teada mitmeid ohukohti, mis võivad muuta animatsioonid aeglaseks.

1.6.1 Dokumendi kehtetuks muutmine (layout thrashing)

Levinud probleem JavaScripti abil veebilehe elementide animeerimise puhul on tihe dokumendis asuvate elementide väärtuste lugemine ning kohene kirjutamine. Näiteks muutes dokumendis asuva elemendi laiust tuleb terve dokumendi geomeetria uuesti välja arvutada. Brauser üritab teha seda võimalikult hilja – kas hetkel teostatava funktsiooni või kaadri lõpus. Kui aga lugeda funktsiooni sees koheselt dokumendist mõne elemendi väärtust, peab brauser terve dokumendi uuesti välja arvutama, sest esialgse elemendi muutmine võis põhjustada ka loetava elemendi atribuutide väärtuste muutust.

JavaScripti koodinäide halvast lähenemisest, kus päritakse dokumendi käest element, mille stiili atribuute loetakse ja uuendatakse mitu korda järjest:

```
var element1 = document.getElementById('element1');
var element2 = document.getElementById('element2');
var height1 = element1.clientHeight;
element1.style.height = height1 + 100 + 'px';
var height2 = element2.clientHeight;
element2.style.height = height2 + 100 + 'px';
```

Koodinäide 3. Näide halvast elemendi stiilide muutmise meetodist

Lahendusena tuleks üritada vältida kordamööda andmete lugemist ja kirjutamist. Koodi struktureerides peaks kõigepealt tegema lugemised ning seejärel kirjutamised. Sellisel juhul peab dokumendi ehitama uuesti ainult ühe korra. (Page, 2013)

JavaScripti koodinäide paremast lahendusest, kus kõigepealt salvestatakse elementide kõrguse stiilid muutujatesse ning alles seejärel muudetakse nende väärtusi:

```
var element1 = document.getElementById('element1');
var element2 = document.getElementById('element2');
var height1 = element1.clientHeight;
var height2 = element2.clientHeight;
element1.style.height = height1 + 100 + 'px';
element2.style.height = height2 + 100 + 'px';
```

Koodinäide 4. Näide heast elemendi stiilide muutmise meetodist

1.6.2 Elementide stiilide muutmise mõju brauserile

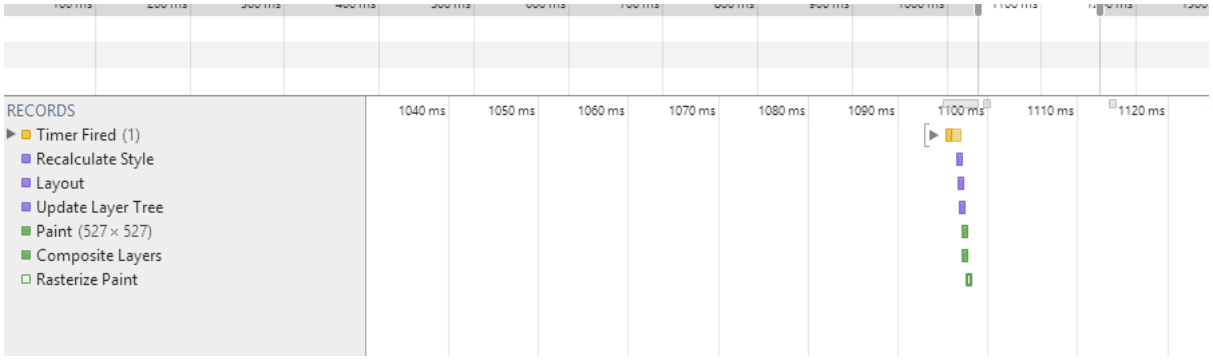
HTML elementide animeerimine kujutab endast vastavate dokumendi osade CSS väärtuste muutmist. Erinevate väärtuste muutmine on ka erineva mõjuga brauseri jaoks. Laiemas plaanis võib jaotada need kaheks:

- Parameetrid, mis on seotud elementide asetusega.
- Parameetrid, mis on seotud elementide välimusega.

1.6.2.1 Asetusega seotud parameetrite muutmine

Animeerides väärtusi mis on seotud elemendi asetusega, muutub dokument kehtetuks ning vajab uuesti geomeetrilist ülesehitamist, positsioneerimist, joonistamist ning kihtidele tõstmist. Näiteks muutes ühe elemendi kõrgust võib see põhjustada ka kõikide teiste elementide positsiooni muutust millest tulenevalt peab brauser kokku panema uue dokumendi uuendatud väärtuste põhjal. Lisaks peab brauser asetuse muutuse tõttu joonistama uuesti kõik elemendid, mis asuvad muudetava elemendi alguse- ja lõpupositsiooni vahel.

Antud muutusi on võimalik jälgida Chrome DevTools'i ajajoone peal, kus neid võib leida layout, paint ja composite layers nimedega paneelide alt (Joonis 9).



Joonis 9. Asetusega seotud parameetri muutmise tulemus Chrome DevTools vaates

Stiilid mille muutmisel peab dokumendis asuvate kõikide elementide asetused uuesti välja arvutama, on vastavalt enam kasutatavuse järjekorras: width, height, padding, margin, display, border-width, border, top, position, font-size, float, background-color, border-color, text-align, overflow-y, font-weight, overflow, left, font-family, line-height, vertical-align, right, clear, white-space, bottom, min-height (CSS properties by style operation required, 2015)

1.6.2.2 *Välimusega seotud parameetrite animeerimine*

Animeerides välimusega seotud väärtusi nagu värv või vari, peab brauser joonistama kõik elemendid uuesti, mis on lisaks mõjutatavale elemendile kas animeeritava dokumendi osa all või peal. Kuna elementide positsioon ei muutunud, pole geomeetriat vaja uuesti arvutada.

Antud muutusi on võimalik jälgida Chrome DevTools'i ajajoone peal, kus neid võib leida paint ja composite layers nimedega paneelide alt (Joonis 10).

Joonis 10. Välimusega seotud parameetri muutmise tulemus Chrome DevTools vaates

Parameetrid mille muutmisel peab muudatusest tulenevad dokumendi osad uuesti joonistama, on vastavalt kasutatavuse järjekorras: color, border-style, visibility, background, text-decoration, background-image, background-position, background-repeat, outline-color, outline, outline-style, border-radius, outline-width, box-shadow, background-size. (CSS properties by style operation required, 2015)

2 Animatsioonide optimeerimine

Hoolimata arvutite jõudluse kasvust ning uuemate brauserite tekkest, peavad veebiarendajad siiski koodi rohkelt optimeerima. Veebilehed või muud rakendused, mis baseeruvad HTML ja CSS peal, vajavad teadlikku optimeerimist, sest suur osa turule lisanduvatest seadmetest on nutiseadmed, mis pole tugeva jõudlusega.

Uudne standardne lähenemine on hakata arendama kõigepealt mobiilseadmetele ning selle põhjal ehitada üles toetus suuremate ekraanidega seadmetele. Sellele võib leida rohkelt viiteid ingliskeelse sõnapaarina Mobile First. Selline lähenemine tekitab aga arenduse koha pealt mitu suurt probleemi. Esiteks on vaja kohendada oma disainid sobilikuks kõikide võimalikkude ekraani suuruste jaoks (inglise keeles Responsive Design), mis on ka peamine külg, millele kujundajad rõhku panevad. Samas on tekkinud ka teine väga oluline aspekt, millele väga ei rõhuta - väiksemate seadmete jõudlus. Kaasaskantavad seadmed töötavad enamasti akutoitel mis tähendab, et iga loodud lahendus kasutab seadmesse alles

jäänud energiat. Mida optimeeritum on rakenduse, seda vähem ressurssi ta kasutab. Ka graafilise jõudlusega pole mobiilseadmetes veel priisata.

Selliseid kohti kus arendajal tuleb teha teadlikke optimeerimisi, on palju, sest rakenduse visuaalne esitlus on kirjeldamatult tähtsal kohal selles külluslikul ning tihedal turul.

2.1 Brauserite tugevustega arvestamine

Kõige lihtsam viis optimaalsete animatsioonide loomiseks on enimkasutatavate brauserite tugevuste arvestamine. Kaasaaegsed brauserid (IE10+, Firefox 32+, Chrome 31+, Safari 7+, Opera 27+) suudavad animeerida efektiivselt nelja parameetrit: positsiooni (position), suurust (scale), rotatsiooni (rotation) ning läbipaistvust (opacity). Kui animeerida mingit muud parameetrit tuleb arvestada, et see võib märgatavalt vähendada animatsioonide jõudlust.

Saavutamaks maksimaalselt efektiivse animatsiooni oleks hea ideaaltingimustes animeerida ainult parameetreid, mis mõjutavad elementide kompositsiooni. Mida rohkem peab brauser tegelema geomeetria kalkuleerimise ja elementide joonistamisega, seda aeglasemaks animatsioonid muutuvad. (Irish & Lewis, 2013)

2.2 Erinevate tehnoloogiate tugevustega arvestamine

Praegusel ajal on peamine osa veebi animatsioonidest loodud kas JavaScript'i või CSS3 abil. Eelnevalt populaarset Flash'i ei leidu peaaegu enam üldse. Peamisteks põhjusteks miks Flash enam populaarne ei ole, on lisa plugina installeerimise vajadus, et brauseris animatsiooni näidata ning ka protsessori suurem koormamine, mis tõttu tuleks eelistada võimaluse korral JavaScript'i või CSS'i.

Peamine JavaScript'i eelis on ühtlasi ka tema üks suurimaid negatiivseid külgi: jooksutatav kood jookseb brauseri pealõimu (main thread) peal, mis tähendab, et animatsioonidega seotud koodi ei saa jooksutada iseseisvalt näiteks arvutustest ja kõigest muust osast, mis on vajalik veebilehe näitamiseks. Pealõimu ülesandeks on niigi elementide stiilide, geomeetria arvutamine ning joonistamine. Rohkete arvutuste tõttu võib kaadrisagedus langeda.

JavaScript pakub rohkelt kontrolli animatsioonide üle ning teda oleks hea kasutada kui on vaja keerukaid animatsioone teostada. Samas lihtsate animatsioonide jaoks, nagu elemendi positsiooni liigutamine, võiks kasutada CSS'i, sest see on jõudluse kohapealt optimaalsem. (Irish & Lewis, 2013)

Peamiseks CSS3 animatsioonide eeliseks on nende automaatne optimeerimine brauseri poolt – vastavalt vajadusele tekitab veebilehitseja ise kihte juurde või jooksutab operatsioone paralleelselt pealõimuga, mis soosib kiiremaid ning sujuvamaid animatsioone. CSS abil oleks tark animeerida võimalikult palju elementide parameetreid, mis ei sõltu rakenduse olekust nagu värvid.

Peamiseks CSS animatsioonide nõrkuseks on vähese kontrollimise võimalus. Keeruline on luua animatsioone, mida saaks lihtsalt ajas manipuleerida vastavalt vajadusele. (Irish & Lewis, 2013)

2.3 Sobiva vahendi valimine

Vähe leidub projekte, kus otsustatakse tulemus realiseerida ilma ühegi abivahendita. Pea alati kasutatakse arendustööd hõlbustavaid raamistikke. Sellepärast on ka töös otsitud välja kasutuses olevad populaarsed teegid, mille põhjal on hea välja tuua näiteid, miks üks või teine on efektiivsem oma töös. Nimetatud abivahendid on valitud Github'i kasutajate eelistuste põhjal. Valikut tehes, pidas autor silmas kõigi valitud raamistike erinevaid eesmärke ning lahendusviise. Lisaks oli tähtis, et iga laialdaselt kasutusel olev animeerimiseks kasutatav tehnoloogia või metoodika oleks kaetud.

Väljavalitud teekideks on:

- Move.js

Teek, mis võimaldab HTML elemente animeerida manipuleerides CSS parameetritega. Kasutusel versioon 0.3.3 - https://github.com/visionmedia/move.js/tree/0.3.3

- jQuery

Teek, mis võimaldab JavaScript'i abil HTML elementide kasutust optimeerida. Laialdaselt kasutuses olev teek dokumendi manipuleerimise jaoks, kuid pole mõeldud ega optimeeritud animatsioonide teostamiseks, mistõttu ei tasuks ka seda elementide animeerimiseks kasutada. Sellest hoolimata kasutatakse jQuery't rohkelt animeerimise eesmärgil. Kasutusel versioon 1.11.2 - https://github.com/jquery/jquery/tree/1.11.2

- GreenSock Animation Platform

Teek, mis võimaldab JavaScript'i abil HTML elemente animeerida. Vahend ongi loodud HTML elementide animeerimiseks ning sihib seda teha väga optimaalselt ning lihtsasti

kasutavalt. Kasutusel versioon 1.16.1 - https://github.com/greensock/GreenSock-JS/tree/1.16.1

3 Animeerimise raamistike ja metoodikate võrdlus

Saamaks reaalseid andmeid, mille põhjal võiks analüüsida erinevaid raamistikke ja metoodikaid arendas töö autor veebirakenduse, mis võimaldab erinevate vahendite abil animeerida HTML elemente. Töös kasutatud kood asub aadressil https://github.com/manuelvulp/html-elementide-optimeeritud-animatsioonid juhuks, kui lugejal tekib soov kas ise üle kontrollida töös väidetut või lihtsalt omal käel järele proovida. Rakenduse ülespanemiseks vajalik informatsioon asub GitHub'i projekti hoiukohas, README.md nimelises failis. Lisaks asub ka kood eelnevalt ülesseatuna autori poolt veebiaadressil http://manuelvulp.com/tlu/.

Kõikide kasutusel olevate valitud teekide koodi testitakse ühtemoodi – dokumendi alale suurusega 350x500 (laius, kõrgus) joonistatakse või lisatakse teatud hulk objekte ning seejärel animeeritakse nende erinevaid parameetreid, kuid kõiki alati samadel tingimustel. Suurus on valitud selline, sest mobiiltelefoni resolutsioon oli 350x500 pikslit suur. Saavutamaks võrreldavaid sülearvuti ja mobiiltelefoni tulemusi, on vaja sama suurt kuvatavat pinda, mida brauser peab uuesti joonistama. Suuremate pindade puhul kulub ka rohkem graafikaprotsessori ressurssi.

Iga lisatud element viitab HTML div elemendile, mille küljed on 50 pikslit suured. Selline suurus on optimaalne, sest siis hõlmab hästi dokumendil asuvaid elemente ning nende animatsioonide liikumisi.

Elementide kogusteks on valitud 100, 500 ja 1000. 100 elemendi puhul peaks saama iga vahend nii arvuti kui ka mobiiltelefoni peal hästi hakkama. 500 peaks olema piir, kus tugevam arvuti saab veel animeerimisega hakkama, kuid mobiil jääb hätta. 1000 elemendi hulgal peaks tekkima mõlemal seadmel animeerimisega probleeme.

Järgnevas tekstis nimetatud CSS3 stiil box-shadow lisatakse alati parameetritega 0 0 20px #333, mis lisab elemendile 20 piksli suuruse taustaga varju, mille värv on tume hall. Värvi valiku põhjuseks on valge tausta peal elemendi varju selge nähtavus.

Järgnevas tekstis nimetatud CSS3 stiil border-radius lisatakse alati parameetriga 50px, mis teeb 50 piksli laiuse küljega ruudust ringi.

Animeerimisel kasutatud CSS3 stiilid on valitud näitamaks, mis efektiga võivad osad stiilid jõudlusele mõjuda.

Rutiiniks mida läbitakse iga teegi või puhta JavaScripti koodiga on järgnev:

- Dokumendile lisatakse erinevad hulgad (100, 500, 1000) elemente ning animeeritakse elementide positsiooni x ja y telje suhtes 3 sekundi jooksul. Elementidele määratakse JavaScript'i koodi poolt juhuslik alguspositsioon (Joonis 11) ning seejärel animeeritakse neid järgmisesse juhuslikku asukohta (Joonis 12).
- Elementidele lisatakse vari kasutades CSS3 box-shadow omadust.
- Elementidele lisatakse ümarad ääred kasutades CSS3 border-radius omadust.
- Elemendid tõstetakse omale kihile lisades CSS3 omaduse translateZ(0).

Võimalusi loomaks animatsioone puhta JavaScripti abil on mitmeid. Antud töös on kasutatud rekursiivselt setTimeout meetodit, mis animeerib elemendi parameetreid regulaarselt pärast igat etteantud ajaühikut kasutamata requestAnimationFrame meetodit (https://github.com/manuelvulp/html-elementide-optimeeritud-animatsioonid/blob/master/app/main.js#L135-L163).

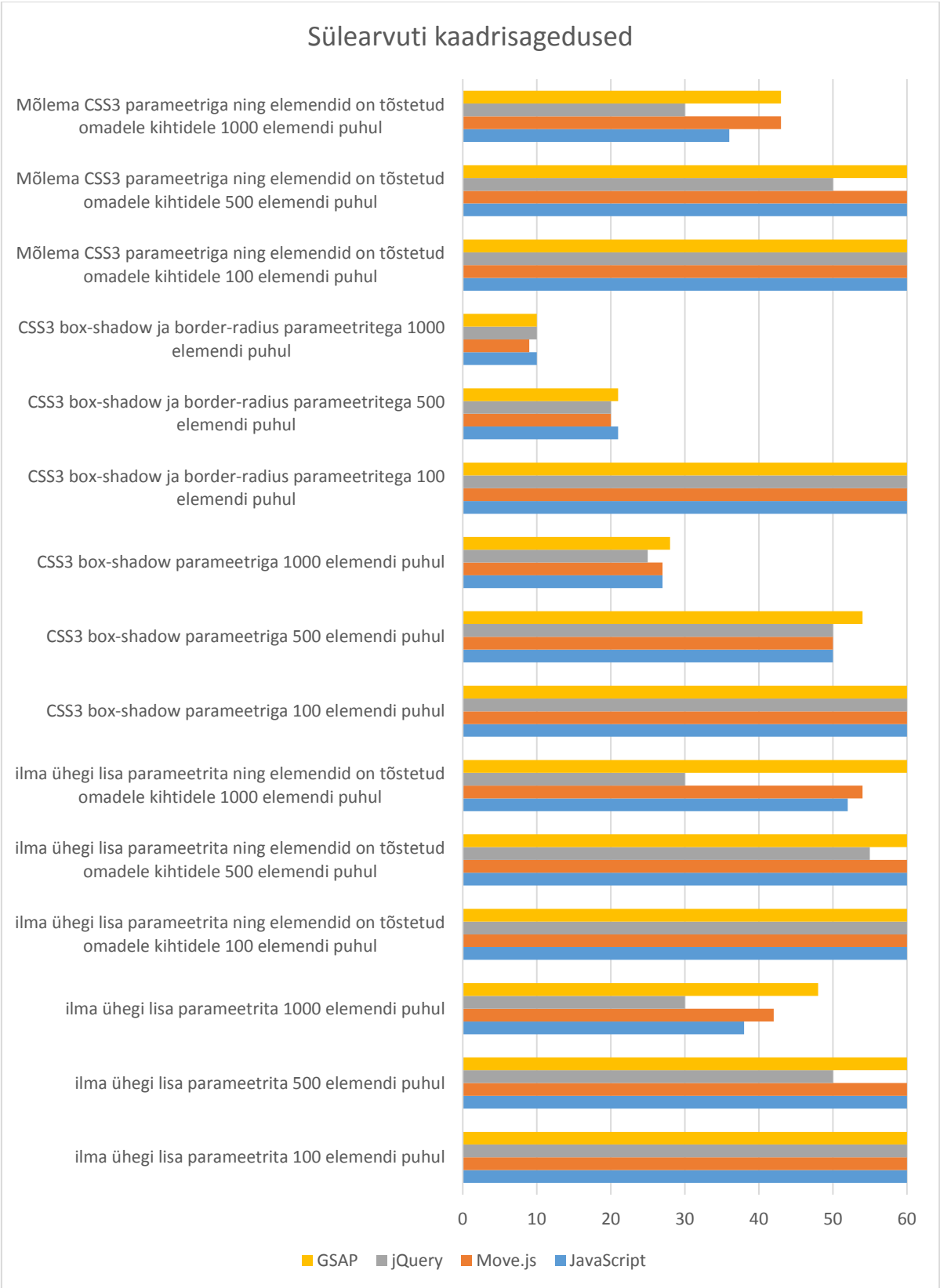
Joonis 11. Element alguspositsioonis

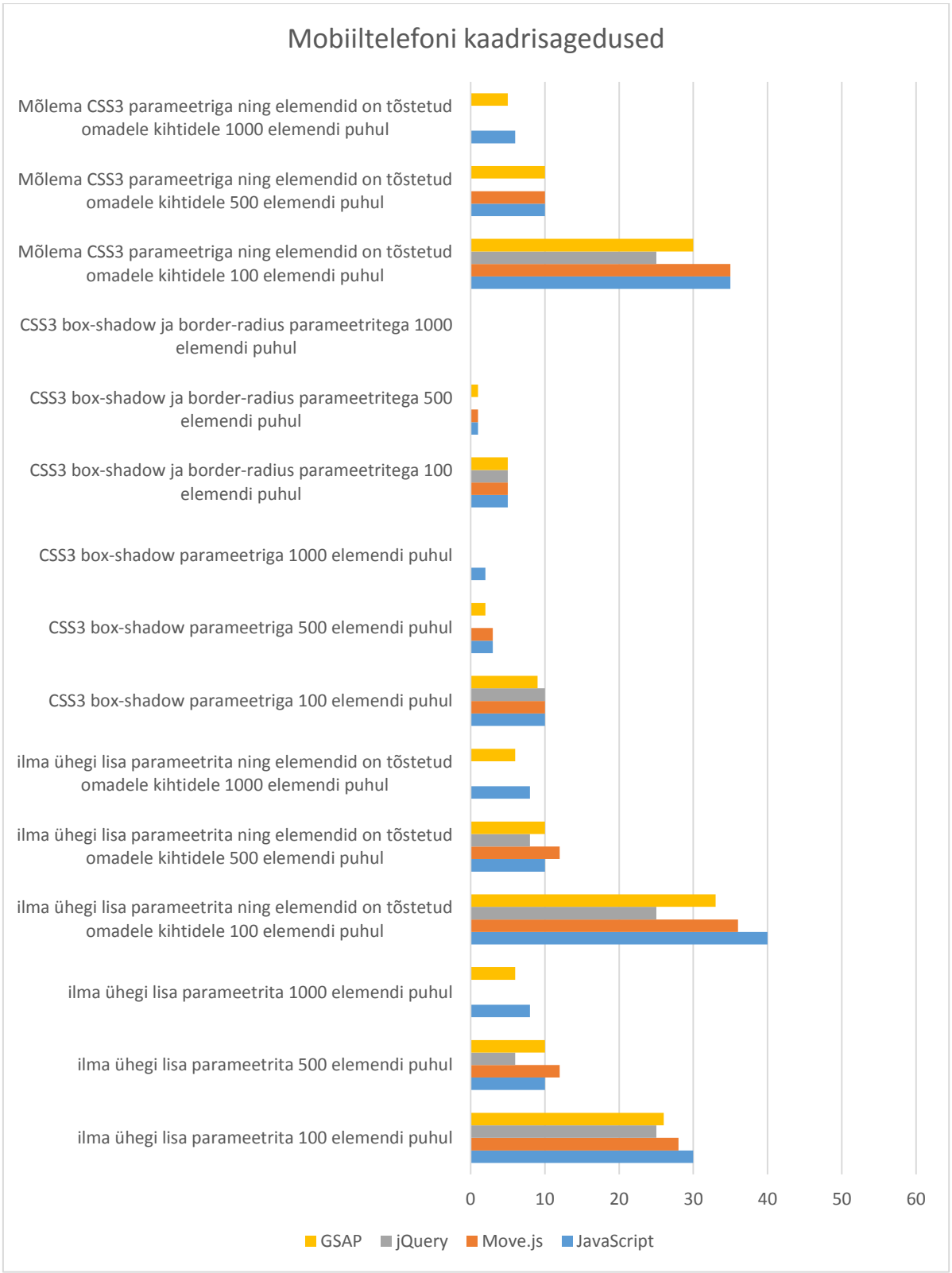
Joonis 12. Element lõpp-positsioonis pärast üht animeerimise rutiini

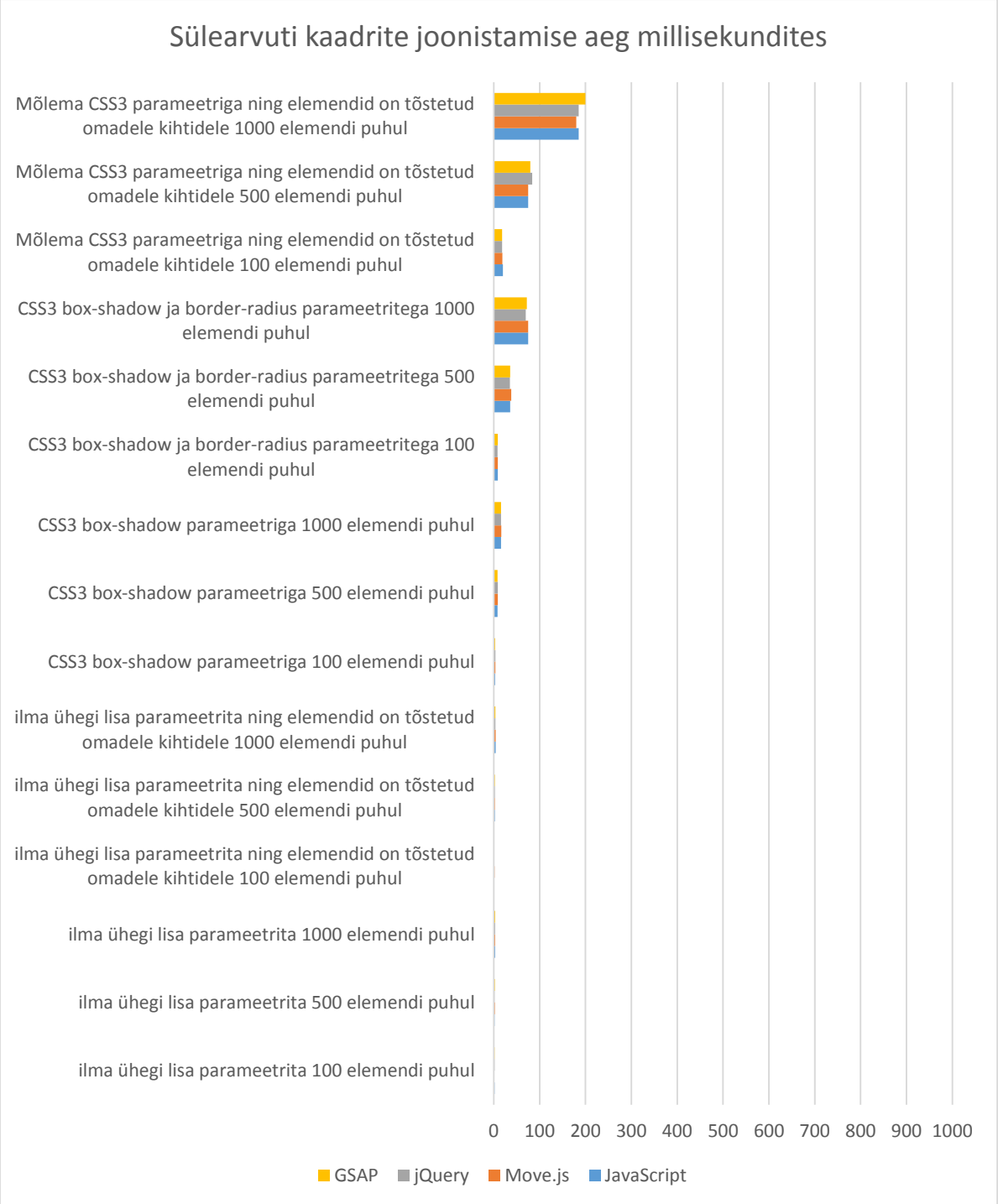
Uurimuse tulemused on saadud sülearvuti ja mobiiltelefoni testimisel.

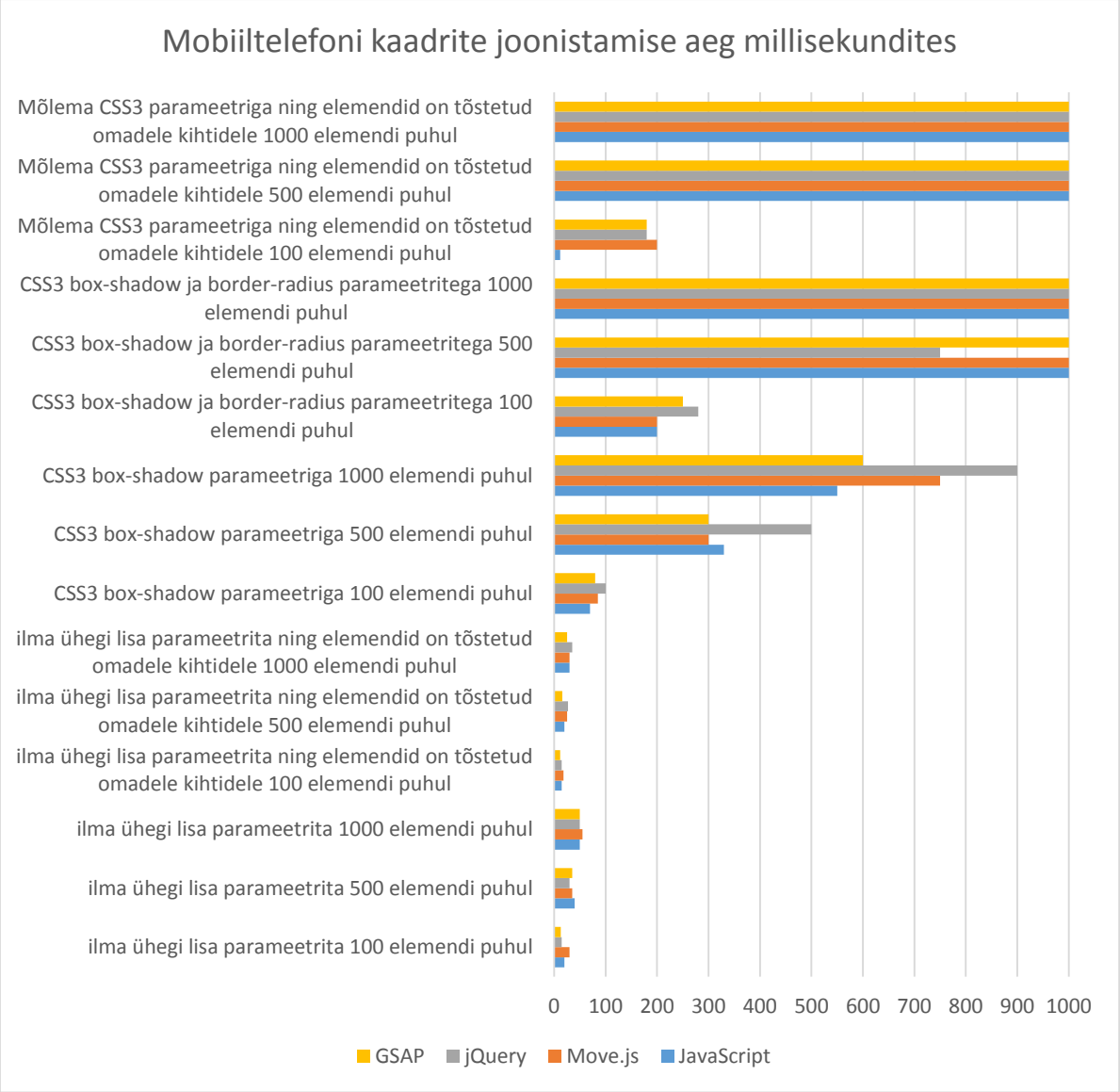
Analüüsiks teostatud arvuti protsessor on Intel® Core(TM) i7-4700HQ CPU @ 2.40GHz 2.40 GHz, mälu 8.00 GB millest 7.89 GB on kasutatav, 64-bitine operatsioonisüsteem Windows 8.1, graafikakaart Intel® HD Graphics 4600 ning NVIDIA GeForce GT 750M. Tulemuste analüüsis nimetusega sülearvuti.

Analüüsiks teostatud telefoni mudel on Sony Xperia Z3 Compact, protsessor Quad-core 2.6 GHz Krait 400, mälu 2 GB, Android OS, Lollipop versioon 5.0.2, graafikakaart Adreno 330. Tulemuste analüüsis nimetusega mobiil.


Sülearvuti kaadrisagedused
Mõlema CSS3 parameetriga ning elemendid on tõstetud omadele kihtidele 1000 elemendi puhul
Mõlema CSS3 parameetriga ning elemendid on tõstetud omadele kihtidele 500 elemendi puhul
Mõlema CSS3 parameetriga ning elemendid on tõstetud omadele kihtidele 100 elemendi puhul
CSS3 box-shadow ja border-radius parameetritega 1000 elemendi puhul
CSS3 box-shadow ja border-radius parameetritega 500 elemendi puhul
CSS3 box-shadow ja border-radius parameetritega 100 elemendi puhul
CSS3 box-shadow parameetriga 1000 elemendi puhul
CSS3 box-shadow parameetriga 500 elemendi puhul
CSS3 box-shadow parameetriga 100 elemendi puhul
ilma ühegi lisa parameetrita ning elemendid on tõstetud omadele kihtidele 1000 elemendi puhul
ilma ühegi lisa parameetrita ning elemendid on tõstetud omadele kihtidele 500 elemendi puhul
ilma ühegi lisa parameetrita ning elemendid on tõstetud omadele kihtidele 100 elemendi puhul
ilma ühegi lisa parameetrita 1000 elemendi puhul
ilma ühegi lisa parameetrita 500 elemendi puhul
ilma ühegi lisa parameetrita 100 elemendi puhul
0
10
20
30
40
50
60
GSAP
jQuery
Move.js
JavaScript


Mobiiltelefoni kaadrisagedused
Mõlema CSS3 parameetriga ning elemendid on tõstetud omadele kihtidele 1000 elemendi puhul
Mõlema CSS3 parameetriga ning elemendid on tõstetud omadele kihtidele 500 elemendi puhul
Mõlema CSS3 parameetriga ning elemendid on tõstetud omadele kihtidele 100 elemendi puhul
CSS3 box-shadow ja border-radius parameetritega 1000 elemendi puhul
CSS3 box-shadow ja border-radius parameetritega 500 elemendi puhul
CSS3 box-shadow ja border-radius parameetritega 100 elemendi puhul
CSS3 box-shadow parameetriga 1000 elemendi puhul
CSS3 box-shadow parameetriga 500 elemendi puhul
CSS3 box-shadow parameetriga 100 elemendi puhul
ilma ühegi lisa parameetrita ning elemendid on tõstetud omadele kihtidele 1000 elemendi puhul
ilma ühegi lisa parameetrita ning elemendid on tõstetud omadele kihtidele 500 elemendi puhul
ilma ühegi lisa parameetrita ning elemendid on tõstetud omadele kihtidele 100 elemendi puhul
ilma ühegi lisa parameetrita 1000 elemendi puhul
ilma ühegi lisa parameetrita 500 elemendi puhul
ilma ühegi lisa parameetrita 100 elemendi puhul
0
10
20
30
40
50
60
GSAP
jQuery
Move.js
JavaScript

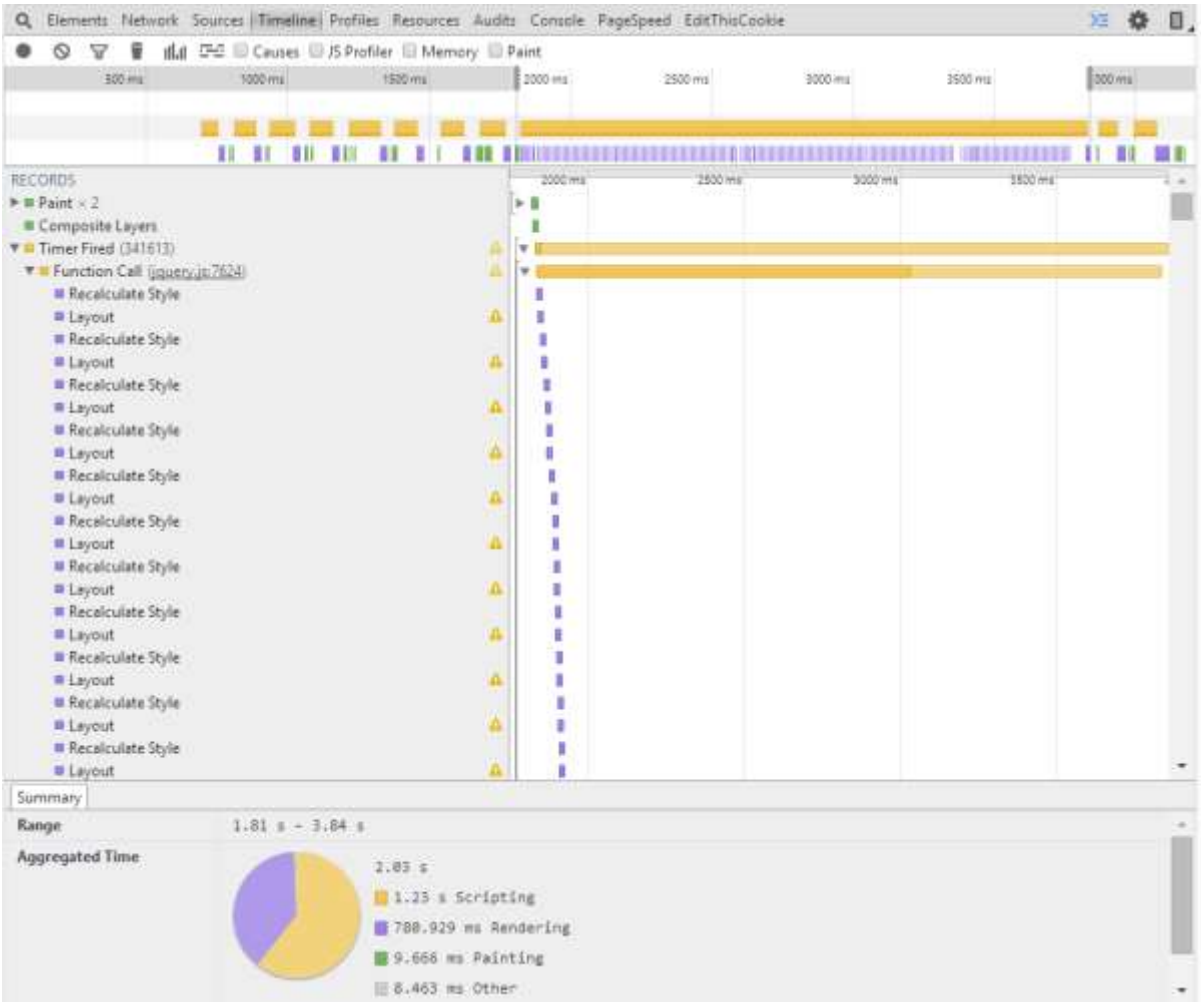

Sülearvuti kaadrite joonistamise aeg millisekundites
Mõlema CSS3 parameetriga ning elemendid on tõstetud omadele kihtidele 1000 elemendi puhul
Mõlema CSS3 parameetriga ning elemendid on tõstetud omadele kihtidele 500 elemendi puhul
Mõlema CSS3 parameetriga ning elemendid on tõstetud omadele kihtidele 100 elemendi puhul
CSS3 box-shadow ja border-radius parameetritega 1000 elemendi puhul
CSS3 box-shadow ja border-radius parameetritega 500 elemendi puhul
CSS3 box-shadow ja border-radius parameetritega 100 elemendi puhul
CSS3 box-shadow parameetriga 1000 elemendi puhul
CSS3 box-shadow parameetriga 500 elemendi puhul
CSS3 box-shadow parameetriga 100 elemendi puhul
ilma ühegi lisa parameetrita ning elemendid on tõstetud omadele kihtidele 1000 elemendi puhul
ilma ühegi lisa parameetrita ning elemendid on tõstetud omadele kihtidele 500 elemendi puhul
ilma ühegi lisa parameetrita ning elemendid on tõstetud omadele kihtidele 100 elemendi puhul
ilma ühegi lisa parameetrita 1000 elemendi puhul
ilma ühegi lisa parameetrita 500 elemendi puhul
ilma ühegi lisa parameetrita 100 elemendi puhul
0 100 200 300 400 500 600 700 800 900 1000
GSAP
jQuery
Move.js
JavaScript



Ilma lisadeta JavaScript'i puhul oli sülearvuti elementide animeerimises ootuspäraselt efektiivsem kui mobiil. Sülearvutiga sai isegi 1000 elemendi puhul parameetreid lihtsalt muuta; JavaScript'i arvutused ei takistanud arvuti protsessorit ega piiranud kasutaja tegevust. Mobiilis tuli 500 elemendi juures kasutusmugavust häiriv piir ette. Lisaks tekkis nii sülearvutil kui mobiilil uute kaadri positsioonide kalkuleerimisega probleeme ning animatsioonid ei lõpetanud enam sihitud 3 sekundi sees kuigi animatsioon oli sülearvutil mõistliku kiirusega ka 1000 elemendi piiril. Peamine mida mõõdetud tulemustest välja võib tuua, on mobiili graafikaprotsessori mälu kasutatavus. Kui CSS3 parameetrid on lisatud elementidele, tõuseb mälu kasutus rohkelt ning mobiilseadmed ei suuda enam kasutajale head kasutuskogemust pakkuda.

Move.js puhul olid mõõtmistulemused väga sarnased ilma lisadeta JavaScriptiga. CSS'i abil animeerimisega on aga välistatud mälulekked, mis on väga kasulik; JavaScriptiga animeerides võib näiteks tänu setTimeout meetodile jääda mingi animatsioon, mida näha pole, rakenduse elutsükliks jooksma ning see vähendab veebilehe jõudlust. Lisaks jäävad setTimeout animatsioonid taustal käima kui kasutaja brauseris ei ole jooksev veebileht aktiivne. Suuremate elemendi hulkade juures (500+) muutusid mobiili animatsioonid üsnagi vaevaliseks, sest uue animatsiooni tsükli positsioonide arvutamiseks lisab Move.js vastavalt igale elemendile uue stiili, mis sätestab kuhu tuleb element animeerida. Küll aga ei jõua mobiilibrauser kõikidele elementidele korraga uusi klasse rakendada ning on sunnitud lisama astmeliselt, mida on ka veebilehitsejas selgelt näha.

jQuery tulemuste peamiseks negatiivseks küljeks on rohke arvutuste tegemine. Raamistik ei ole animatsioonide jaoks ehitatud ning tulemustest on seda ka selgelt näha. Kui teiste raamistike või vahenditega võttis veebirakendus kasutaja sisendit vastu siis jQuery puhul toimub see väga aeglaselt, sest pealõim on pidevalt blokeeritud. Põhjuseks on eelnimetatud layout thrashing (Joonis 13); igal uuel koordinaatide arvutamisel loetakse ning kirjutatakse kõikide elementide parameetreid, mistõttu pole kasutajal sel ajal võimalik midagi teha.



Joonis 13. Näide pidevast dokumendi manipuleerimise tulemusest

GreenSock Animation Platform oli kõige paremate tulemustega raamistik, mis ühtlasi ongi loodud HTML elementide animeerimiseks. Kuigi kohati võivad tulemused olla üpriski sarnased ilma lisadeta JavaScripti omadele, on siiski neil väga suur vahe sees – esiteks läbivad animatsioonid oma teekonna sihitud 3 sekundi jooksul ning tänu kaadrite joonistamisele brauseri poolt, näevad selle raamistikuga animatsioonid sujuvamad välja kui teiste JavaScript'i raamistikega.

Kokkuvõte

Käesoleva bakalaureusetöö eesmärk oli anda ülevaade probleemidest, mis võivad tekkida HTML elementide animeerimisel ning tutvustada veebi animatsioonide optimeerimiseks mõeldud tööriistu ning kuidas nende abil mõõdetud tulemuste põhjal oma veebirakenduses jooksvaid animatsioone optimeerida. Antud töö jaoks loodi veebirakendus, mis animeerib erinevate raamistike abil HTML elemente ning saadud tulemuste põhjal analüüsiti erinevate vahendite efektiivsust.

Autori loodud veebirakenduse põhjal saab selgelt järeldada, et kasutades optimaalsemaid teeke on animatsioonide jõudlus märgatavalt suurem. Levinumad teegid toetavad ka kaasaaegseid brausereid, mis tähendab, et arendaja ei pea iga veebilehitseja jaoks koodi eraldi optimeerima. Ka analüüsimise tööriistad on tähtsal kohal, sest animatsioone ei saa optimeerida kui ei tea nõrku kohti.

Kuigi turul on praeguseks väga palju nutiseadmeid ning nende arv on kasvamas, jääb nende võimsus süle- ja lauaarvutitele märgatavalt alla. Luues veebilahendusi tuleb olla teadlik erinevate seadmete jõudlustest ning sellest kuidas oma rakendust vastavale platvormile kohendada. Palju aitab kaasa õige vahendi valimine soovitud animatsioonide teostamiseks, sest paremad metoodikad soodustavad suuremat jõudlust. Kiirelt muutuvad ja uuenevad ka brauserid mistõttu on kindlaid reegleid, mis kestaksid igavesti, jõudluse suurendamiseks raske välja tuua. Kõige tähtsam on oskus mõõta oma rakenduse jõudlust ning selle nõrku kohti, sest optimeerida ei ole võimalik teadmata, mis viga on.

Töö on mõeldud kõikidel tasemetel veebiarendajatele, kes tunnevad huvi oma rakenduse visuaalse jõudluse vastu ning sooviksid seda õppida mõõtma ning paremaks muutma. Lisaks on töös välja toodud ka erinevad teegid, mis arenduse lihtsamaks peaksid tegema.

Summary

Animations Optimization of HTML Elements

The aim of this Bachelor thesis was to give an overview of performance issues that might occur while animating HTML elements and also to introduce tools that help developers measure their work for better code optimization. The author also created an application that allows users to test different popular JavaScript and CSS animation frameworks to see how well each performs. The source code of the application is available on https://github.com/manuelvulp/html-elementide-optimeeritud-animatsioonid.

The thesis is divided into two parts. First part provides reader with necessary knowledge of how animations work, what tools can be used and how to use them. Second part describes efficient ways of animating HTML elements and includes a result set of laptop and smartphone performance for comparison. Based on these results, two things can be clearly stated: smartphones are significantly weaker and thus need more code optimization, and by using better tools the outcome is superior.

The main takeaway from this research is the knowledge of how to use tools for measuring and detecting weaknesses of any web application and also how to optimize code for better performing animations.

Kasutatud kirjandus

(kuupäev puudub). Allikas: https://docs.google.com/spreadsheet/pub?key=0ArK1Uipy0SbDdHVLc1ozTFlja1dhb25QNGhJMXN5MXc&single=true&gid=0&output=html

144Hz Monitors. (4. Aprill 2015. a.). *So what does 144Hz or 120Hz actually mean?* Allikas: 144Hz Monitors: http://www.144hzmonitors.com/knowledge-base/so-what-does-144hz-or-120hz-actually-mean/

Can I use. (21. Veebruar 2015. a.). Allikas: Can I use requestAnimationFrame: http://caniuse.com/#search=requestAnimationFrame

CSS properties by style operation required. (2015). Allikas: CSS properties by style operation required: https://docs.google.com/spreadsheets/d/1Hvi0nu2wG3oQ51XRHtMv-A_ZlidnwUYwgQsPQUg1R2s/edit#gid=0

Google Chrome. (kuupäev puudub). *Chrome DevTools Overview*. Allikas: Google Chrome: https://developer.chrome.com/devtools

Irish, P., & Lewis, P. (7. November 2013. a.). *High Performance Animations*. Allikas: HTML5 Rocks: http://www.html5rocks.com/en/tutorials/speed/high-performance-animations/

Lewis, P. (27. Veebruar 2013. a.). *On translate3d and layer creation hacks*. Allikas: Aerotwist: http://aerotwist.com/blog/on-translate3d-and-layer-creation-hacks/

Liu, C., Granados, O., Duarte, R., & Andrian, J. (2012). Journal of Information Processing Systems, Vol.8, No.1. *Energy Efficient Architecture Using Hardware*, 134.

Mozilla Developer Network. (kuupäev puudub). *Window.requestAnimationFrame().* Allikas: The Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

Page, W. (19. September 2013. a.). *Preventing 'layout thrashing'*. Allikas: Wilson Page: http://wilsonpage.co.uk/preventing-layout-thrashing/

PC Magazine Encyclopedia. (kuupäev puudub). *Encyclopedia*. Allikas: PC Magazine: http://www.pcmag.com/encyclopedia/term/58898/frame-rate

Web Standards Project. (1998). *The Web Standards Project*. Allikas: WaSP: Fighting for Standards: http://www.webstandards.org/about/mission/

Wiltzius, T. (11. Märts 2013. a.). *Accelerated Rendering in Chrome*. Allikas: HTML5 Rocks: http://www.html5rocks.com/en/tutorials/speed/layers/