Tallinna Ülikool

Digitehnoloogiate instituut

CSS ja JS animatsioonid Internetis

Bakalaureusetöö

Autor: Risto Kitsing

Juhendaja: Andrus Rinde

Autor: .....,,.....,,2016

Juhendaja: .....,,.....,,2016

Instituudi direktor: .....,,.....,,2016

Tallinn 2016

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

Sõnastik 5

Sissejuhatus 8

1 Veebilehtede visualiseerimise protsess 9

1.1 DOM ja CSSOM 9

1.2 Visualisatsioonipuu 10

1.3 Graafikakonveier 11

2 CSS animatsioonid 13

2.1 CSS transitions 13

2.2 CSS keyframes 16

2.3 CSS animatsioonide jõudlus 19

2.3.1 CSS animatsioonide jõudluse testimine 19

2.3.2 Kokkuvõte CSS animatsioonide testimisest 22

3 JavaScript animatsioonid 23

3.1 JavaScripti taimerfunktsioonid 23

3.2 JavaScripti meetod requestAnimationFrame 24

3.3 Web Animations API 24

3.4 JavaScript animatsioonide jõudlus 26

3.4.1 JS animatsioonide jõudluse testimine 26

3.4.2 Kokkuvõtte JS animatsioonide testimisest 29

Kokkuvõte 31

Summary 32

Kasutatud kirjandus 33

Sõnastik

Markup language või HTML	**Hüpertekst-märgistuskeel** – kodeerimissüsteem veebidokumentide loomiseks
CSS ehk Cascading Style Sheets	**Kaskaadlaadistik** – laadilehed, mis koosnevad reeglitest, mis kirjeldavad veebilehe elementide väljanägemist
JS ehk JavaScript	**Skriptikeel** – vahend loomaks interaktiivseid veebisaite ja animatsioone
Jitter või jank	**Värin** – animatsiooni kaadrisageduse kõikumine madala jõudluse tõttu
Parse	**Sõelumine** – keele jagamine koostisosadeks, mida on võimalik analüüsida, kompileerida
Node	**Sõlm** – puu struktuuri tipp; graafi sõlmpunkt
Conversion	**Kodeerimine** – binaarmärkide muutmine tähejadaks
Tokenizing	**Märgistamine** – märgistuskeele elemendi loomine tähejadast, näiteks HTMLis <p>
Lexing	**Sõnavaraline töötlus** –vea avastamise protsess
DOM/CSSOM construction	**Olemipuu loomine** – HTML/CSS märkide lisamine graafi

Render tree	**Visualisatsioonipuu** – märgistuskeele elementide olemust ja välimust kirjeldav andmestruktuur
Pixel rendering pipeline	**Graafikakonveier** – erinevate matemaatiliste protsesside läbimine visualiseerimiseks
Style calculations	**Stiili kalkulatsioonid** – HTML elemendi stiili visuaalne rehkendus
Layout	**Küljendamine** – HTML elementide paigutamine brauseri kuvavälja
Paint või rasterizing	**Võõpamine** või **rasterdamine** – lehekülje kuvamiseks loodud masinloetav maatriks
Layers	**Kiht** – antud kontekstis graafikaprotsessori poolt visualiseeritav lehekülje osa
Compositing	**Ladumine** – HTML elementide ladumine brauseri kuvaväljale
Vendor-specific prefixes	**Tarkvara tootja spetsiifilised eesliited** – erinevad veebilehitsejate tarkvara tootjad kasutavad erinevaid küljendusmootoreid, mis nõuavad kaskaadlaadistikus reeglitele spetsiaalseid eesliiteid
Timing function	**Ajaarvutusfunktsioon** – funktsioon kirjeldab aega, mille jooksul animatsioon liigub ühest olekust teise, harilikult mõjutades animatsiooni mängimise kiirust

Debugging	**Silumine** – programmides vigade avastamine ja kõrvaldamine
Hardware acceleration	**Riistvaraline kiirendus** – antud kontekstis mõiste, mis tähendab rasterdamist graafikaprotsessoris
Parallax scrolling	**Parallaks kerimine** – võte veebiarenduses, kus veebilehe kerimisel eespool olevad komponendid liiguvad kiiremini kui tahapoole paigutatud elemendid, mis liiguvad aeglasemalt
Main thread	**Pealõim** – antud kontekstis tähendab see seadme protsessori poolt brauserile jagatud protsessori aega, mille tsüklite jooksul tuleb täide viia vajalikud operatsioonid
Garbage collection	**Mälukoristus** – vabaksjäänud mäluplokkide kogumine skripti täitmise kestel; koristuseta kasvaks skripti mäluvajadus ning ületaks süsteemi virtuaalmälu piirid
API (Application Programming Interface)	**Rakendusliides** – rakendusprogrammiga määratud reeglistik, mille alusel pakutakse teise rakendusprogrammi teenuseid
Selector	**Selektor** – Kaskaadlaadistikus ja JavaScriptis esinev viide valimaks kindlaid veebilehe elemente
FPS ehk *Frames Per Second*	**Kaadrisagedus** – kaadrisageduse mõõtühik; paljud ekraanid ja monitorid kuvavad kaadreid 60 Hz sagedusega

Sissejuhatus

Tänapäeval on veebibrauserid suutelised rohkemaks, kui lihtsalt dokumentide serveerimiseks. Erinevad tehnoloogiad võimaldavad disaineritel ja arendajatel luua animatsioone, mille abil saab kanda suuremas koguses informatsiooni kasutajani, kui läbi lugude või piltide. Viimastel aastatel on veebianimatsioonid muutunud äärmiselt populaarseks teemaks veebidisainerite ja –arendajate seas: mullu toimunud animatsiooni- ja disainikonverentsil *Blend*, nimetas Justin Cone veebianimatsioone tulevikuks. (Nabors, 2016)

Ometi pole animatsioonid Internetis midagi uut. Kunagi massiliselt kasutusel olnud pistikprogramm Adobe Flash on alates aastast 2010, kui Steve Jobs selle kohta arvamust avaldas, aina vähem kasutust leidnud. Turvalisus, jõudlus ja kohmakas kasutajakogemus olid peamised aspektid, mille pärast Apple otsustas lõpetada Adobe Flashi kasutamise oma toodetes. Selle asemel soovitas Jobs kasutada HTML5't, CSS3'e ja JavaScripti – tehnoloogiad, mis võimaldavad arendajatel luua paremaid animatsioone ning graafikat ja mida toetatakse mitmel erineval platvormil pistikprogrammideta. (Jobs, 2010)

Autor valis antud teema, kuna huvitub eesrindlikkest veebitehnoloogiatest ja võimalusest rakendada neid parema visuaali ja kasutajakogemuse eesmärgil oma igapäevases töös. Käesoleva bakalaureusetöö peamiseks eesmärgiks on tutvustada sujuvate animatsioonide loomist Internetis, kasutades selleks CSSi ja JavaScripti erinevaid tehnikaid, sealhulgas rakendades Web Animations APIt (edaspidi WAAPI) mis on veel väga uus ja jätkuvalt töös olev tehnoloogia. Töö võiks pakkuda huvi veebiarendajatele ja –disaineritele ning informaatika tudengitele, kes soovivad luua sujuvaid animatsioone kolmanda osapoole pistikprogrammideta või väliste teekideta.

Autor tutvub tehnoloogia spetsifikatsiooniga World Wide Web Consortium (edaspidi W3C) poolt, mis on veel eskiisi tasandil ning avatud muudatustele, kuid mille üldine suunitlus jääb samaks. Lisaks tutvub autor Google Developers, Mozilla Developer Networki ja teiste eriala spetsialistide poolt avaldatud materjalidega ning katsetab praktikas olulisemaid võimalusi ja annab omapoolse hinnangu.

1 Veebilehtede visualiseerimise protsess

Animatsioonid mängivad suurt rolli silmapaistvate veebirakenduste ja –lehtede loomisel. Need hõlmavad kasutaja tegevusse, andes sealjuures visuaalset tagasisidet interaktsioonide kohta, näiteks menüü avamine, sisu laadimine, nupule vajutus jne. Tihtilugu mahutatakse liialt palju erinevaid animatsioone veebilehele või jääb nende tehniline teostus nõrgaks, mistõttu kõik seadmed ei suuda pakkuda sujuvat kasutajakogemust. Probleemi lahendamiseks on arendajatel ning disaineritel oluline endale teadvustada, kuidas brauser sisemiselt muudab nende loodud koodi kasutatavaks ja interaktiivseks veebileheks. (Lewis, Animations | Web Fundamentals - Google Developers, 2016) Eesmärgiks on tagada, et veebilehe kerimine ja animatsioonid ühtiks seadme ekraani kaadrisagedusega, milleks on tänapäeval 60 korda sekundis, ehk 60 hertsi. See tähendab, et ühe kaadri valmistamiseks on seadmel $\frac{1\,s}{60\,Hz} = 16.66\,ms$ ehk umbes 17 millisekundi suurune ajavahemik. Tegelikkuses on see aeg väiksem, sest brauser peab tegelema ka muude kohustustega, näiteks mälukoristusega (inglise keeles *garbage collection*), mistõttu Google soovitab vajalikud toimingud sooritada 10 millisekundi jooksul. Vastasel juhul ilmneb ekraanil värin (inglise keeles *jitter* või *jank*) ning animatsioonid ei paista silmale sujuvad. (Lewis, Rendering performance | Web Fundamentals - Google Developers, 2016)

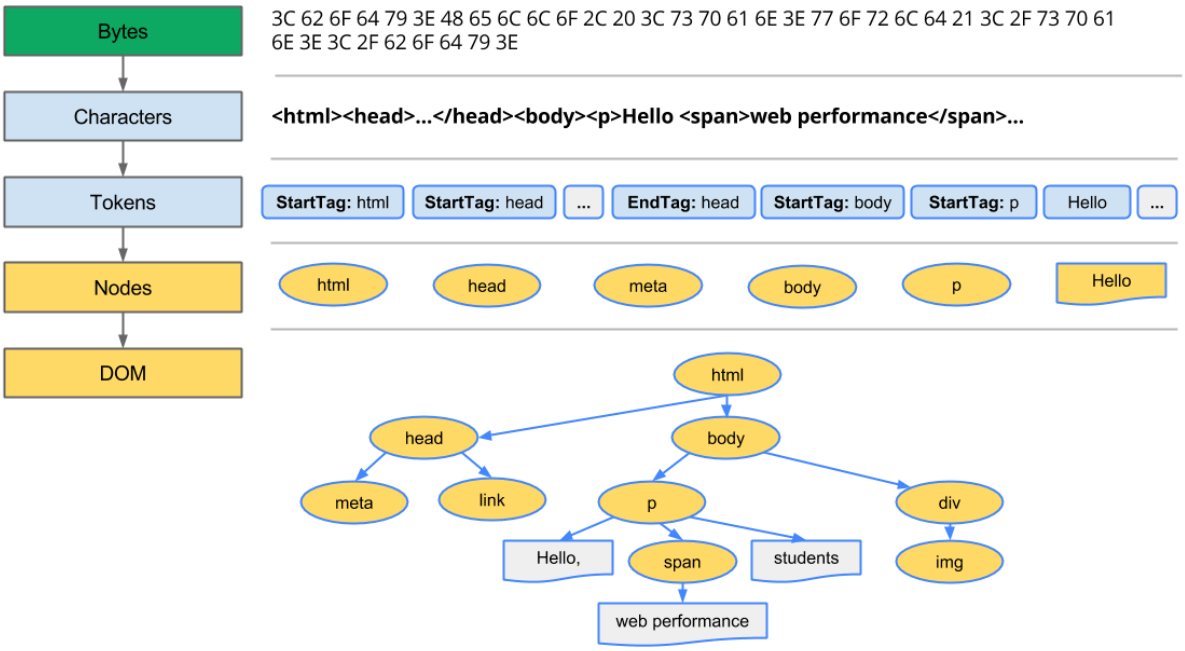
1.1 DOM ja CSSOM

HTML on oma olemuselt märgistuskeel, mis järgib puu-kujulist andmestruktuuri. Iga element on kapseldatud mõne suurema elemendi sisse, välja arvatud juurelement `<html>`. HTML ja CSS märgistuskeeled (inglise keeles *markup language*) sõelutakse (inglise keeles *parse*) brauseri poolt ning muudetakse *Document Object Model*'iks (edaspidi DOM) ja *CSS Object Modeliks* (edaspidi CSSOM). Olemipuud toimivad liidesena, mis võimaldavad arendajatel luua skripte, et dünaamiliselt muuta ja uuendada veebilehe sisu, struktuuri ning elementide välimust. DOM olemipuus kirjeldab sõlm (inglise keeles *node*) mõnda dokumendi elementi, näiteks `<p>` ja selle sisu *„Tere, maailm!"*. CSSOM-puus, aga kirjeldatakse stiilireegleid, kasutades selektoreid, millele vastab kindel element ning deklaratsiooni, mis kirjeldab rakenduvat stiili, näiteks `p {font-size: 16px;}`. (W3C, 2016) Olemipuude loomine on ajaliselt kallis tegevus, sest brauser peab läbima neli olulist sammu (vt Joonis 1):

- **kodeerimine** (inglise keeles *conversion*) – brauser loeb seadme kettalt või Internetist tulevad baidid tähejadaks kasutades ettenähtud kodeeringut (nt UTF-8);

- **märgistamine** (inglise keeles *tokenizing*) – brauser töötleb tähejada, et luua HTML standardile vastavad märgised, näiteks `<h1>…</h1>`;
- **sõnavara töötlus** (inglise keeles *lexing*) – märgised muudetakse objektideks, mis defineerivad nende omadused ja rakenduvad reeglid;
- **olemipuu loomine** (inglise keeles *DOM/CSSOM construction*) – objektid seostatakse olemipuusse, mida brauser kasutab hiljem veebilehe visualiseerimiseks.

Suure hulga HTML ja CSS koodi korral võtab kogu protsess pisut aega, mistõttu peavad arendajad olema ettevaatlikud, kui pidevalt muudetakse HTML elemente ja nende omadusi. (Grigorik, Constructing the Object Model | Web Fundamentals - Google Developers, 2016)

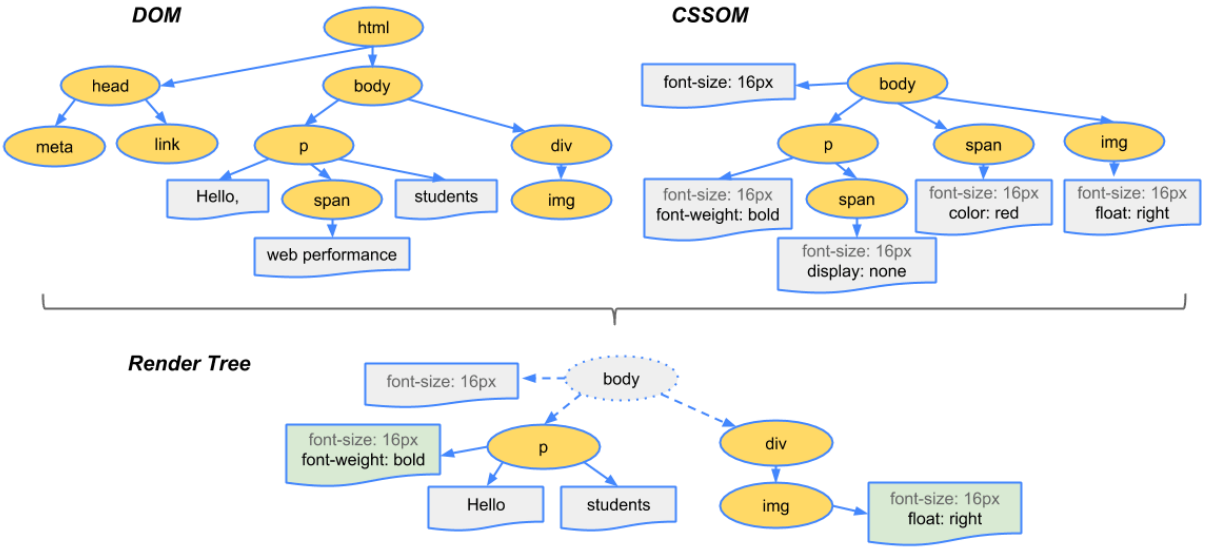


Joonis 1: Olemipuu koostamise protsess (Ilya Gregorik, Creative Commons Attribution 3.0 litsents, muutmata kuju)

1.2 Visualisatsioonipuu

Peale DOMi ja CSSOMi loomist, liidab brauser olemipuud kokku, et moodustada visualisatsioonipuu (inglise keeles *render-tree*). Brauser käib esmalt läbi kõik nähtavad sõlmed, milleks on peamiselt `<body>` ja selle järglased; näiteks `<script>`, `<meta>` ja muud sarnased elemendid jäetakse puust välja, sest need on seotud sisemiste süsteemidega ning ei kajastu otseselt veebilehel. Mõned sõlmed on DOMis peidetud CSSi abil, näiteks `<span>` element (vt Joonis 2) `display:none;` atribuudiga ja neid ei lisata visualisatsioonipuusse. Ülejäänud CSSOM reeglid rakendatakse DOM elementidele kasutades selektoreid. Visualisatsioonipuu loomise

tulemusena hakatakse selles sisalduvaid elemente samm-sammult leheküljele paigutama, lähtudes brauseri vaateakna suurusest. (Grigorik, Render-tree construction, layout and paint | Web Fundamentals - Google Developers, 2016)



Joonis 2: Visualisatsioonipuu loomine (Ilya Gregorik, Creative Commons Attribution 3.0 litsents, muutmata kuju)

1.3 Graafikakonveier

Graafikakonveier (inglise keeles *pixel rendering pipeline*) kujutab endast toimingute jada, mida veebilehitseja täidab peale visualisatsioonipuu valmimist, kui brauser on valmis alustama elementide ekraanile paigutamisega. Kogu protsessi võib jagada viieks erinevaks lõiguks, mille käitumise üle on kujundajatel ja arendajatel kõige rohkem voli:

- JavaScript – dünaamilise HTMLi puhul võib muutuda lehekülje välimus ja sisu, näiteks kas läbi jQuery (populaarne JavaScripti raamistik) `animate` funktsiooni või DOM elementide lisamisega leheküljele, mistõttu tuleb visualisatsioonipuu uuesti genereerida;
- stiili kalkulatsioonid (inglise keeles *style calculations*) – lähtudes visualisatsioonipuust, otsitakse, selektorite abil vastavad elemendid ja arvutatakse välja nende suurused, värvid ning muud parameetrid;
- küljendamine (inglise keeles *layout*) – siis, kui veebilehitseja teab, millised reeglid rakenduvad elementidele, hakatakse looma lehe kompositsiooni, kus määratakse kõigi nähtavate elementide positsioon ja paigutus sõltuvuses teineteisega. Tegemist on üpriski aeganõudva toiminguga, sest käiakse läbi kõik nähtavad elemendid;

- võõpamine või rasterdamine (inglise keeles *paint* või *rasterizing*) – võõpamine on pikslite paigutamine veebilehitseja vaatevälja. Selle käigus joonistatakse teksti, värve, pilte, piirjooni, varje ning paljut muud. Enamjaolt toimub võõpamine eraldi pindadele, mida kutsutakse kihtideks (inglise keeles *layers*);
- ladumine (inglise keeles *compositing*) – kuna veebilehe osad joonistati potentsiaalselt mitmele kihile, tuleb need laduda ekraanile õiges järjekorras, et veebisait oleks korrektselt kuvatud. See on eriti oluline elementide puhul, mis kattuvad teineteisega, näiteks tekst pildil.

Graafikakonveier on protsess, mida kerimise, animatsioonide ja dünaamilise sisu kuvamisel ei läbita alati tervenisti. Google on avastanud kolm peamist stsenaariumit, mis võivad juhtuda visuaalsete muudatuste korral.

1) **JavaScript/CSS → stiili kalkulatsioonid → küljendamine → võõpamine → ladumine**

Kui muutub mõni küljenduse omadus, mis muudab elemendi geomeetriat nagu pikkus, laius või asukoht, siis veebilehitseja peab arvutama uued väärtused teiste elementide paiknemise jaoks. Kõik muudetavad alad tuleb taaskord võõbata ning laduda.

2) **JavaScript/CSS → stiili kalkulatsioonid → võõpamine → ladumine**

Kui muutub mõni võõpamise omadus, näiteks taustapildi muutus, tekstivärv, varjud või muu efekt, mis ei muuda küljendust, siis brauser loob uue rasterkujutise, mis laotakse ekraanile.

3) **JavaScript/CSS → stiili kalkulatsioonid → ladumine**

Kui muutub mõni elemendi omadus (näiteks teksti värv), mis ei sõltu lehekülje kompositsioonist ega võõpamise vajadusest, siis veebilehitseja laob elemendid uuesti ekraanile. Selline juhtum on kõige vähem ressurssi nõudev ning kujutab endast näiteks animatsioone ja kerimist. (Lewis, Rendering performance | Web Fundamentals - Google Developers, 2016)

Visualiseerimise protsessi juures saab minna veelgi tehnilisemaks ning pühendada palju aega optimeerimisele, kuid tehnoloogiate areng toob kaasa kiiremad ja efektiivsemad veebimootorid, mistõttu on autori arvates peamine disaineritel ja arendajatel mõista nende üldist töövoogu, et neile mitte vastu töötada ning saada tulemuseks hea jõudlusega animatsioonid sõltumata kasutatavast tehnoloogiast.

2 CSS animatsioonid

CSS on olemuselt deklaratiivne programmeerimise tüüp, kus arendaja või disaineri poolt loodud kood täidetakse brauseri poolt samm-sammult. W3C CSS Working Group on palju tööd teinud järgmise CSS versiooni, CSS3 arendamisega, mis võimaldab tänapäeval luua sarnaseid animatsioone, mida varem tehti Adobe Flashi või JavaScriptiga. Eelkõige on CSS3 animatsioonid mõeldud lihtsamate efektide saavutamiseks veebilehe või –rakenduse kasutajaliideses, mille abil saab muuta elementide läbipaistvust, paigutust, värvi, liikumist jne. (Google Developers, 2015) Ajalooliselt on CSSi kasutades olnud võimalik elementide olekut dünaamiliselt muuta alatest aastast 1998, mil avaldati W3C poolt CSS2.1 spetsifikatsioon, mis võimaldas kasutada pseudoklasse nagu `:hover`, `:active` ja `:focus`. (W3C, 2016) Viimastel aastatel on CSS3 rakendusliidest täiendatud uute funktsioonidega, mis on eeskätt mõeldud kasutajaliidese efektide, ehk animatsioonide jaoks, nagu `transition` ja `keyframes`. Kuna tegemist on veel W3C eskiisiga ning mitte kehtiva standardiga, ei pruugi kõik brauserid neid funktsioone toetada, mistõttu on oluline lisada juurde tarkvaratootja spetsiifilised eesliited (inglise keeles *vendor specific prefixes*), näiteks Opera ja Safari puhul `-webkit-*`. (W3C CSS Working Group, 2015)

2.1 CSS transitions

Harilikult, kui elemendi CSS omadus muutub, kajastub see koheselt brauseris. `transition` võtmesõna võimaldab disaineritel määrata animatsiooni viivituse, kestvuse ja käitumise ajaarvutusfunktsiooniga (inglise keeles *timing function*). Elementide puhul on lõplik hulk omadusi, mida saab animeerida (vt `https://www.w3.org/TR/css3-transitions` peatükk *Properties from CSS*), kuid see võib aja jooksul lähtuvalt W3C standardist muutuda. Siirde omadused kasutavad mitut parameetrit (vt Koodinäide 1):

- viivitus (`transition-delay`) - mitme (milli)sekundi pärast alustatakse siiret;
- kestvus (`transition-duration`) - millise aja jooksul siirde animatsiooni kuvatakse;
- omadus (`transition-property`) - reeglite hulk, mida saab siirdeks määrata, näiteks laius (*width*), läbipaistvus (*opacity*), mastaap (*scale*) ja kõik omadused (*all*);
- ajaarvutusfunktsioon (`transition-timing-function`) – animatsiooni sujuvus suhtes ajaga, näiteks ühtlane kiirus, kiirenev või aeglustuv.

```
transition-delay: time;
transition-duration: time;
transition-property: none|all|property;
transition-timing-function: linear|ease|ease-in|ease-out|ease-in-out|cubic-
bezier(n,n,n,n);
/*Kiirkirja pilt:*/
transition: property duration timing-function delay;
```

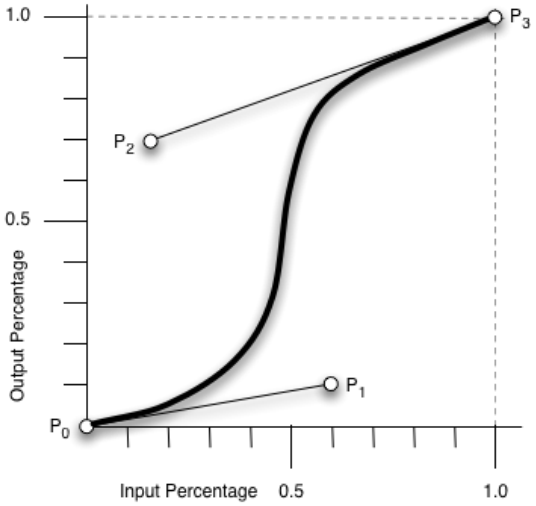
Koodinäide 1: `transition` omadused välja kirjutatuna (W3Schools, http://www.w3schools.com/css/css3_transitions.asp)

Mozilla soovitab kasutada CSS3 kiirkirja pilti, mis võimaldab arendajatel vältida liigset koodi ning hõlbustada silumisprotsessi (inglise keeles *debugging*). (Mozilla, 2015) Siirde näiteks võib tuua ruudu, mis `:hover` sündmuse puhul muudab oma kuju (vt Koodinäide 2).

```
HTML

<div class="box"></div>

CSS

div.box {
  width: 100px;
  height:100px;
  background-color:#BADA55;
  transition: border-radius 300ms;
}

div.box:hover {
  border-radius: 50%;
}
```

Koodinäide 2: Polümorfne ruut (Interaktiivne näide: `http://codepen.io/ristokit/pen/vGQGQG`)

Siirde puhul on kõige keerulisem omadus ajaarvutusfunktsioon, mille abil on võimalik muuta animatsiooni järkjärgulist kulgemise (inglise keeles *easing*) kiirust teatud aja jooksul (vt Joonis 3). Funktsioon võtab sisendiks matemaatilise polünoomi, mida tuntakse Bézieri kõverana (inglise keeles *Bézier curve*) ning kirjeldatakse nelja argumendiga Bézier-funktsioonina. (Mozilla, 2015)



Joonis 3: Bézieri kurv animatsiooni järkjärguliseks muutmiseks, W3C (https://www.w3.org/TR/css3-transitions)

W3C spetsifikatsioonis on eeldefineeritud viis järkjärgulist ajaarvutusfunktsiooni, mis hõlbustavad disainerite tööd:

- *ease* – `cubic-bezier(0.25, 0.1, 0.25, 1);`
- *linear* - `cubic-bezier(0, 0, 1, 1);`
- *ease-in* - `cubic-bezier(0.42, 0, 1, 1);`
- *ease-out* - `cubic-bezier(0, 0, 0.58, 1);`
- *ease-in-out* - `cubic-bezier(0.42, 0, 0.58, 1);` (W3C , 2016)

Bézieri funktsiooni väärtused peavad jääma vahemikku nullist üheni (vt Koodinäide 3); piire ületades ei pruugi animatsioon toimida.

HTML

```
<div class=“box“></div>
```

CSS

```
div.box {
  width: 100px; height: 100px; background-color: #BADA55;
  transition-delay: 0;
  transition-duration: 300ms;
  transition-property: width;
  transition-timing-function: cubic-bezier(0.75, 0.25, 0.5, 1);
}
```

```
div.box:hover {
  width: 200px;
}
```

Koodinäide 3: Bézieri ajaarvutusfunktsiooni näide kasti laiuse muutmise läbi (Interaktiivne näide: `http://codepen.io/ristokit/pen/mPQPgO`)

Siirde puhul tõlgendab brauser animatsiooni alg- ja lõpp-seisundit lähtudes CSSist, ning arvutab vahepealsed sammud iseseisvalt, mis hõlbustab arendajate ja disainerite tööd. Tunduvalt keerukamate animatsiooni saavutamiseks tuleb rakendada `keyframes` funktsiooni. (Mozilla, 2015)

2.2 CSS keyframes

`@keyframes` reegel võimaldab disaineritel kontrollida animatsioonide kulgu konkreetsete kaadrite defineerimise abil, erinevalt siirdest (`transition`), mille vahekaadrid animeeritakse täielikult brauseri poolt. Selleks, et kasutada `@keyframes` reeglit, ehk võtmekaadrit, tuleb eelnevalt HTML elemendile lisada külge `animation` omadus(ed) (vt Koodinäide 4).

`animation` omadus kasutab mitut parameetrit:

- nimi (`animation-name`) – animatsiooni autori poolt lisatud identifitseerimistunnus;
- kestvus (`animation-duration`) – millise aja jooksul animatsiooni kuvatakse;
- ajaarvutusfunktsioon (`animation-timing-function`) – animatsiooni sujuvus suhtes ajaga, näiteks ühtlane kiirus, kiirenev või aeglustuv. Sarnane siirdele;
- viivitus (`animation-delay`) – mitme (milli)sekundi pärast alustatakse animatsiooni;
- korduste arv (`animation-iteration-count`) – mitu korda animatsiooni kuvatakse. `Infinite` väärtuse korral korratakse animatsiooni lõputult;
- suund (`animation-direction`) – võimaldab animatsiooni mängida tagurpidi või edasi-tagasi;
- paigalseisund (`animation-fill-mode`) – peale animatsiooni kuvamist või viite ajal elemendile kehtivad stiilireeglid;
- olek (`animation-play-state`) – JavaScripti liidese kaudu dünaamiliselt juhitav omadus, mille abil saab peatada animatsiooni.

```
animation-name: keyframename;
animation-duration: time;
animation-timing-function: linear|ease|ease-in|ease-out|ease-in-out|cubic-
```

```
bezier(n,n,n,n);
animation-delay: time;
animation-iteration-count: number|infinite;
animation-direction: normal|reverse|alternate|alternate-reverse;
animation-fill-mode: none|forwards|backwards|both;
animation-play-state: paused|running;
/*Kiirkirjapilt:*/
animation: name duration timing-function delay iteration-count direction
fill-mode play-state;
```

Koodinäide 4: `animation` omadused kiirkirjapildiga

Kasutades kiirkirjapilti animatsiooni loomiseks, tuleb omaduse parameetriteks anda vähemalt nimi (*name*) ja kestvus (*duration*). Võtmekaadri reegli kasutamiseks tuleb seadistada vähemalt alg- ja lõpp-staadiumid: `0% {…}` ja `100% {…}` (vt Koodinäide 5). Võtmekaadrile saab määrata sajandiku täpsusega ehk kahe komakohaga ajahetke, mis annab teoreetiliselt 10001 erinevat võtmekaadrit, mis peaks olema piisav mistahes animatsiooni loomiseks. (Mozilla, 2016)

```
HTML

<div class="box"></div>

CSS

div.box {
  width:100px;
  height:100px;
  background-color:#BADA55;
  animation: example 300ms;
}

@keyframes example {
  0% { width:100px; height:100px; }
  100% { width:200px; height:200px; }
}
```

Koodinäide 5: Võtmekaadri funktsiooni kasutamine

Defineerimata võtmekaadrite puhul arvutab brauser vaheetapid iseseisvalt, sarnaselt siirdele. Dubleeritud vahekaadrite deklaratsiooni korral võetakse kasutusele vaid viimasena esinenud versioon, sest neid ei ole võimalik (välja arvatud Firefox 14ndas ja hilisemas versioonis) laiendada sarnaselt kaskaadlaadistikule, mistõttu `!ignore` ja muud üle kirjutamised ei rakendu.

Stiile, mis on defineeritud mõnes võtmekaadris, kuid mitte alg- ega lõpp-staadiumis, ignoreeritakse (vt Koodinäide 6). (Mozilla, 2016)

```css
@keyframes badExample {
  0% { top: 0; }
  50% { left: 10px; }
  100% { top: 20px; }
}
```

Koodinäide 6: Algus- või lõpp-staadiumis deklareerimata filmikaadrit ei animeerita

Vajadusel saab kasutada mitut `animation` reeglit ühe elemendi kohta, kuid need tuleb komaga eraldada. (Mozilla, 2016) Järgnev näide sisaldabki endas mitut animatsiooni, mida saab rakendada ühele HTML elemendile (vt Koodinäide 7).

```HTML
<div class="box"></div>
```

```css
div.box {
  width:100px;
  height:100px;
  background-color:#BADA55;
  position:relative;
  animation: pulse 2s Infinite, move 2s Infinite;
}

@keyframes pulse {
  0% { width:100px; height:100px; transform:rotate(-90deg); }
  50% { width:120px; height:120px; transform:rotate(90deg); }
  100% { width:100px; height:100px; transform:rotate(0deg); }
}

@keyframes move {
  0% { left:0; }
  50% { left:100px; }
  100% { left:0; }
}
```

Koodinäide 7: Mitut võtmekaadri funktsiooni kasutav animatsioon (Interaktiivne näide: http://codepen.io/ristokit/pen/wGQWPx)

2.3 CSS animatsioonide jõudlus

Omaduste muutmine ei ole brauseri jaoks kõige lihtsam tegevus. Näiteks, animeerides elemendi laiust, muutub selle geomeetria, ja see võib põhjustada teiste elementide asukoha muutust, peale mida läbitakse uuesti visualiseerimise protsess. Protsessi läbimise aeg sõltub sellest, kui palju elemente on veebilehel ning milliseid omadusi peale geomeetria veel muudetakse. Google soovitab vältida animatsioone, mis käivitavad graafikakonveieri protsesse nagu stiilide arvutamine, küljendamine ja ladumine.

Kasutades moodsate brauserite poolt juurutatud omadusi nagu `transform: translate(n px, n px)`, `transform: scale(n)`, `transform: rotate(n deg)` ja `opacity`, saab visualiseerimise protsessi viia üle graafikaprotsessorile (inglise keeles *hardware acceleration*), mis kuvatakse veebilehel eraldi kihina. (Irish & Lewis, 2014) Sarnaselt toimib veel vähe toetatud (Chrome, Safari, Opera; Firefox Developer Edition) CSS3 reegel, `will-change`, mis annab veebilehitsejale teada elemendi omaduste muudatustest ning loob selle jaoks iseseisva kihi, mida töötleb graafikaprotsessor. Google soovitab antud reeglit lisada veebilehe või –rakenduse nendele elementidele, mille omadused muutuvad oleku või animatsiooni tõttu. Samas Google hoiatab, et reegli üleliigne kasutamine (rohkem kui kümme) toob kaasa jõudluse vähenemise. (Lewis & Thorogood, Animations and performance | Web Fundamentals - Google Developers, 2016)

2.3.1 CSS animatsioonide jõudluse testimine

Autor sooritas CSS animatsioonide jõudluse mõõtmiseks kolm katset:

1) `keyframes` animatsioon absoluutse positsioneerimisega;
2) `keyframes` animatsioon absoluutse positsioneerimisega, `will-change` reegliga;
3) `keyframes` animatsioon kasutades `translate3d(x, y, z)` reeglit.

Iga katse puhul loodi sama animatsioon, kus 50x50 ruudustikus 20x20 pikslit suured ruudud liiguvad ekraanil edasi-tagasi 200px võrra, viis korda järjest. Selline olukord sunnib brauserit läbima tervet graafikakonveierit või kasutama graafikaprotsessorit animatsiooni kuvamiseks. 2500 HTML elemendi pidev manipuleerimine on teoreetiliselt kurnav tegevus, mis nõuab arvutilt palju tööd. Säärane koormustest aitab tuvastada kõige optimaalsema animeerimise tehnika.

Seade, millel katsed läbi viidi on mõeldud eelkõige tarkvara arenduseks ja multimeediumi töötlemiseks, ehk tegu on tööjaamaga:

- Intel i7-6700K protsessor (4 tuuma, 8 lõime, 4.0 GHz);
- nVidia 980Ti graafikakaart (2816 CUDA tuuma, 1 GHz);
- ProLite E2607WS monitor (60 Hz);
- Google Chrome versioon 50.0.2661.94 m (kasutades selle *DevTools* konsooli keskmise kaadrisageduse mõõtmiseks).

Mõõtmine toimus *Codepen.io* keskkonnas, kus autor oli loonud eelnimetatud animatsioonid kasutades CSS3 erinevaid võtteid. Iga katse sooritati kolm korda, igal korral lehte laadides `CRTL+F5`, mis tühjendab brauseri vahemälu.

Katse 1: `keyframes` animatsioon absoluutse positsioneerimisega

2500 genereeritud HTML elementi, klassiga *box*, animeeriti kasutades `keyframes` reeglit. Elemente paigutati ümber, määrates nende `left` omadust, sundides brauserit läbima tervet graafikakonveierit, mis on ajaliselt kulukas.

Veebiaadressil `http://codepen.io/ristokit/pen/VaEmWL` on näha autori poolt kirjutatud koodi animatsiooniga. Testi sooritati kolm korda, igal korral mõõtes keskmist kaadrisagedust (vt Tabel 1).

Test nr	Keskmine kaadrisagedus
1	41
2	41
3	41

Tabel 1: Katse 1 mõõtmistulemused

Katse 1 keskmine kaadrisagedus oli **41**. Tegemist on väga stabiilse tulemusega, mis autori arvates on selgitatav läbi võimsa protsessori, mis suudab hakkama saada kõigi elementide kuvamisega kasutades ainult protsessorit.

Katse 2: `keyframes` animatsioon absoluutse positsioneerimisega, `will-change` reegliga;

2500 genereeritud HTML elementi, klassiga *box*, animeeriti kasutades `keyframes` reeglit ning määrates `will-change` omaduse, mis ütleb brauserile, et elemendi `left` omadus muutub. See sunnib elemendi eraldi kihti, mida kuvatakse graafikaprotsessori abil ning ei tohiks olla kulukas

tegevus. Google hoiatas, et antud omadust mitte rakendada rohkem kui umbes kümnele elemendile, kuid autor usub, et jõudluse poolest kõrgelt hinnatud graafikakaart saab sellega hakkama.

Veebiaadressil `http://codepen.io/ristokit/pen/Myzjgw` on näha autori poolt kirjutatud koodi koos animatsiooniga. Testi sooritati kolm korda, igal korral mõõtes keskmist kaadrisagedust (vt Tabel 2).

Test nr	Keskmine kaadrisagedus
1	17,7
2	22
3	20

Tabel 2: Katse 2 mõõtmistulemused

Katse 2 keskmine kaadrisagedus: **19.9**. Tegemist on väga nõrga tulemusega, mis autori arvates on põhjustatud CSS3 `will-change` rakendusliidese juurutamisest. Tegu on veel niivõrd uue tehnoloogiaga, mis ei pruugi olla veel optimeeritud. Autor julgeb kindlalt väita, et madal kaadrisagedus pole põhjustatud graafikakaardi poolt.

Katse 3 `keyframes` animatsioon kasutades `translate3d(x, y, z)` reeglit

2500 genereeritud HTML elementi, klassiga *box*, animeeriti kasutades `keyframes` reeglit ning määrates `translate3d(x, y, z);` omaduse, mis sarnaselt `will-change` sunnib elementi kuvama eraldi kihina, mis ei tohiks olla kulukas tegevus.

Veebiaadressil `http://codepen.io/ristokit/pen/vGVyVy` on näha autori poolt kirjutatud koodi koos animatsiooniga. Testi sooritati kolm korda, igal korral mõõtes keskmist kaadrisagedust (vt Tabel 3).

Test nr	Keskmine kaadrisagedus
1	59.9
2	59.9
3	59.9

Tabel 3: Katse 3 mõõtmistulemused

Katse 3 keskmine kaadrisagedus: **59.9**. Tegemist on seni parima tulemusega. Autorile üllatuseks on vaid 59.9 kaadrisagedus, mis võib olla monitori või Chrome konsooli eripära mitte kuvada täit 60 kaadrit sekundis.

2.3.2 Kokkuvõte CSS animatsioonide testimisest

Katse 1 tulemuste põhjal võib kindlalt väita, et brauser pidi läbima iga liigutuse jaoks graafikakonveieri, mis nõudis palju seadme protsessorilt. Nõrgemate seadmete puhul, mis ei kasuta niivõrd võimast protsessorit, on autor veendunud, et tulemused on madalamad. Stabiilset kaadrisagedust tõlgendab autor kui protsessori ja brauseri suutlikkust jagada koormus mitme lõime peale.

Katse 2 puhul ei oska autor välja tuua muud põhjendust, kui optimeerimata rakendusliidese juurutamine, mis aja jooksul võib paraneda.

Katse 3 näitas, et tunduvalt vanem rakendusliidese komponent, mis toimib sarnaselt katses 2 kasutatud tehnoloogiaga, toimis kõige paremini. Nii Google kui Mozilla soovitasid oma materjalides kasutada `translate3d(x, y, z);` omadust animatsioonide jaoks. Autorit hämmastab asjaolu, et 2500 HTML elemendi animeerimisel ei tekkinud ühtegi vahele jäetud kaadrit ega värinat.

Kokkuvõtteks autor teeb järelduse, et lähtudes oma ala ekspertide soovitustest ning järgides nende poolt loodud juhendeid, on võimalik saavutada sujuvaid animatsioone ka kõige absurdsemate stsenaariumite puhul, mis on 2500 HTML elemendi pidev edasi-tagasi liigutamine ekraanil ühtegi kaadrit vahele jätmata.

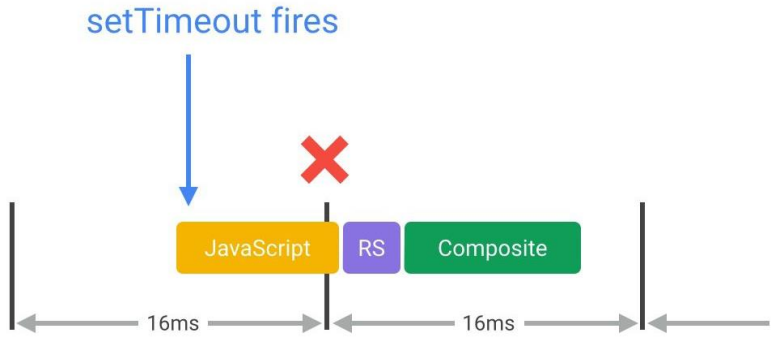
3 JavaScript animatsioonid

JavaScript võimaldab luua nii visuaalselt kui tehniliselt keerukamaid animatsioone, kui CSS3 seda praegusel momendil võimaldab. Erinevate graafiliste liideste jaoks kohati sobib paremini CSS3, kuid keerukamate animatsioonide tarbeks sobib hästi JavaScript. JavaScripti saab kasutada ka koos CSS animatsioonidega, kasutades brauseri rakendusliidest, et ligi pääseda CSSOMile, millega on võimalik CSS animatsioone dünaamiliselt alustada, lõpetada, tagurpidi mängida ja katkestada. See pole kõige lihtsamini teostatav, kuid selleks eksisteerivad välised skriptiteegid nagu jQuery. Näiteks parallaks-kerimine (inglise keeles *parallax scrolling*) veebilehtedel on võimalik ainult JavaScriptiga oma imperatiivse omaduse poolest, võrreldes CSSi deklaratiiviga. `<canvas>` elemendi animeerimine mängude loomiseks on võimalik ainult JavaScriptiga.

Suure valikuvabadusega kaasnevad aga ka mõned piirangud: näiteks asjaolu, et JavaScript toimib veebilehitseja pealõimes (inglise keeles *main thread*) ning brauser peab samaaegselt töötlema HTMLi, koostama olemipuid, tegelema graafikakonveieri alamprotsessidega jne. See tähendab, et ajavahemik, milles arendaja oma animatsioone luua saab, on väga väike. Suurte ja keeruliste animatsioonide puhul, kus tuleb palju arvutusi sooritada, kahaneb see aeg võrdlemisi suure kiirusega ning brauser võib jätta kaadri kuvamata.

3.1 JavaScripti taimerfunktsioonid

Traditsiooniliselt on skriptide alusel animatsioonide loomiseks kasutatud `setTimeout()` ja `setInterval()` funktsioone, mis käivituvad 60 korda ühe sekundi jooksul, et ühtida 60 kaadri taktsagedusega enamus ekraanidel. Tuntud raamistikud nagu jQuery, kasutavad animeerimiseks näiteks `animate()` funktsiooni, mis lähtekoodis avaldub kui `setTimeout(function() {…}, 1000/60)`, ehk 16.67ms jooksul sunnitakse brauseri pealõime kõige vajalikuga valmis saama. Ometi ei pruugi see nii olla, sest samal ajal toimuvad pealõimes muud kalkulatsioonid, sealhulgas mälu prügikoristus (inglise keeles *garbage collection*), mistõttu kaader võib hilineda või kõige hullemal juhul kuvamata jääda. (vt Joonis 4). (Lewis, Optimize JavaScript execution | Web Fundamentals - Google Developers, 2016)



Joonis 4: Taimerfunktsiooni kasutades (kollane) võib kaader vahele jääda, sest brauser peab tegelema ka stiili arvutustega (lilla: RS – Recalculate Style) ja ladumisega (roheline). (Paul Lewis, Creative Commons Attribution 3.0 litsents, muudetud kujul)

3.2 JavaScripti meetod requestAnimationFrame

Taimerfunktsioonide alternatiiv, `requestAnimationFrame()` meetodi välja kutsumisel reserveeritakse brauseri pealõimes kaadri loomiseks vajalik aeg. Sellist lähenemist soovitavad nii Google ja Mozilla. Funktsioon toimib rekursiivselt, mistõttu on oluline see iga kaadri joonistamise korral uuesti välja kutsuda. Brauseri pealõime reserveerimisega tagatakse võimalikult kõrge värskendussagedus, mida seade või ekraan toetab. Juhul, kui brauser on üle koormatud või töötab taustal, `requestAnimationFrame()` meetodit välja enam ei kutsuta või ei eraldata enam maksimaalset protsessori aega. Nõnda tagatakse hea jõudlus ning mobiilsete seadmete puhul pikem aku kestvus. Rakendusliidese tasemel lahendatud automatiseeritus vähendab oluliselt animeerimiseks vajalikku koodibaasi ja tööd, mida arendajad tegema peavad. (Mozilla, 2016)

3.3 Web Animations API

Web Animations API (WAAPI) on hübriid CSS `keyframes` reeglist ja JavaScripti `requestAnimationFrame()` meetodist, mis kirjeldati W3C eskiisis 2015 aasta juulis. (Birtles, Stephens, Danilo, & Atkins, 2016) Oma unikaalsuse tõttu on viimasel ajal sellest veebiarenduse maailmas palju juttu olnud, kuid oma uudsuse tõttu pole veel täielikult toetatud kõikides brauserites – vaid WebKit mootorit kasutavad veebilehitsejad nagu Chrome, Opera ja Safari kuvavad WAAPIga loodud animatsioone. Mozilla Firefox ja Microsoft Edge tugi lisandub tulevikus, kui W3C eskiisist jõutakse soovituseni. (caniuse, 2016)

WAAPI võimaldab disaineritel ja arendajatel:

- inspekteerida töötavaid animatsioone – WAAPI pakub dünaamilist lähenemist animatsioonide silumiseks, kus arendaja ei pea ootama animatsiooni lõppu vaid saab jooksvalt teha vajalikke muudatusi ning mõõta elementide omadusi;
- juhtida töötavaid animatsioone – WAAPI liides pakub lihtsat ligipääsu animatsioonidele, et neid vajadusel korraga peatada või kerida teatud olekuni silumise protsessi või kasutajakogemuse parandamise tarbeks;
- luua animatsioone skriptides – CSS animatsioonid on visualiseeritud läbi graafikaprotsessori, mistõttu on need sujuvad ning vähendavad brauseri pealõime tööd. WAAPI abil on võimalik luua JavaScripti baasil animatsioone, mille jõudlus praegusel momendil ühtib CSS animatsioonidega ning mis sarnanevad süntaksilt `keyframes` reegliga. (Birtles, Stephens, Danilo, & Atkins, 2016)

Lihtsama animatsiooni loomine kasutades WAAPIt on suhteliselt kerge: veebilehele tuleb lisada skript, mis valiks animeeritava objekti ning kirjeldaks muutusi (vt Koodinäide 8).

```
var player = document.querySelector('.animateObject').animate([
    { transform: 'scale(.8) ', opacity: 1 },
    { transform: 'scale(.6)', opacity: .8 },
    { transform: 'scale(.3)', opacity: .4 },
    { transform: 'scale(.1)', opacity: .1 }
  ], {
    duration: 1000,
    easing: 'ease-in-out',
    delay: 0,
    iterations: 0,
    direction: 'normal',
    fill: 'forwards'
  });
```

Koodinäide 8: Lihtsama animatsiooni loomine kasutades Web Animations APIt

Animatsioone saab WAAPI abil juhtida samuti kerge vaevaga: tuleb valida animeeritav objekt ning kutsuda välja rakendusliidese meetodid (vt Koodinäide 9).

```
var animation = element.animate(…); //Animeeritav element
console.log(animation.playState); //Annab tulemuseks "running"
animation.pause(); //Peatab animatsiooni
animation.play();  //Taastab animatsiooni
```

```
animation.cancel(); //Viib animatsiooni algusesse
animation.finish(); //Viib animatsiooni lõppu
animation.reverse(); //Paneb animatsiooni tagurpidi mängima
animation.playbackRate = 1;//Komakohaga arv, määrab animatsiooni kiiruse
```

Koodinäide 9: Animatsioonide juhtimine Web Animations API abiga

`document.timeline.getAnimations()` meetod tagastab kõik jooksvad animatsioonid massiivina.

Web Animations API on paljulubav rakendusliides, mida veebiarendajad alles hakkavad kasutusele võtma ning mille kohta ilmub järk-järgult aina enam materjali. Tõenäoliselt tulevikus muutuvad mõned detailid, sest praegune tehnoloogia on vaid W3C eskiis, kuid üldine suunitlus peaks jääma samaks. (Birtles, Stephens, Danilo, & Atkins, 2016)

3.4 JavaScript animatsioonide jõudlus

JavaScripti animatsioonide loomine on tehniliselt keeruline, sest tuleb arvestada väga väikese ajavahemikuga, milles veebilehel või –rakendusel saab brauseri pealõimes toiminguid läbi viia. Rohke töö tõttu ei sobi animatsioonide loomiseks `setInterval` ega `setTimeout`, pigem `requestAnimationFrame` või Web Animations API, sest ajaliste intervallidega tagatakse staatiline kaadrisagedus mis näiteks ei ühti seadme ekraani värskendamissagedusega või kaotatakse kaadreid suure koormuse tõttu. (Lewis, Optimize JavaScript execution | Web Fundamentals - Google Developers, 2016) Sarnaselt CSS animatsioonidele, kehtib JavaScripti animatsioonidele sarnane reegel, et ei tasu üle 100 elemente korraga animeerida, eriti omadusi, mis sunnivad brauserit läbima graafikakonveieri kulukaid protsesse (stiilide arvutamine, ladumine ja rasterdamine).

3.4.1 JS animatsioonide jõudluse testimine

Autor sooritas JavaScript animatsioonide jõudluse mõõtmiseks kolm katset:

1) Taimerfunktsiooni `setTimeout` kasutades;
2) `requestAnimationFrame` meetodit kasutades;
3) Web Animations APIt kasutades;

Iga katse puhul loodi sama animatsioon, kus 2x2 ruudustikus 100x100 pikslit suured ruudud liiguvad ekraanil edasi-tagasi 200px võrra, viis korda järjest. Selline olukord sunnib brauserit mitte ainult läbima graafikakonteineri protsessi, vaid kulutama palju aega skripti(de)

jooksutamisele, mis peaks mõjuma brauserile kurnavalt. Erinevalt CSS animatsioonide jõudlusest, ei saanud autor animeerida kõiki 2500 HTML elementi, mistõttu autor piirdus neljaga, sest eeltöö käigus selgus, et see võib olla piisav arv. Tegu ei ole kõige kurnavamate skriptide ega animatsioonidega ning mõni erinevus võib avalduda koodibaasi erinevusest, kuid autor ei pööranud sellele tähelepanu, sest võimsa protsessori puhul võib erinevus olla kaduvväikene.

Seade, millel katsed läbi viidi on mõeldud eelkõige tarkvara arenduseks ja multimeediumi töötlemiseks, ehk tegu on tööjaamaga:

- Intel i7-6700K protsessor (4 tuuma, 8 lõime, 4.0 GHz);
- nVidia 980Ti graafikakaart (2816 CUDA tuuma, 1 GHz);
- ProLite E2607WS monitor (60 Hz);
- Google Chrome versioon 50.0.2661.94 m (kasutades selle *DevTools* konsooli keskmise kaadrisageduse mõõtmiseks).

Mõõtmine toimus *Codepen.io* keskkonnas, kus autor oli loonud eelnimetatud animatsioonid rakendades JavaScripti erinevaid rakendusliideseid. Iga katse sooritati kolm korda, igal korral lehte laadides `CRTL+F5`, mis tühjendab brauseri vahemälu.

Katse 1: Taimerfunktsiooni `setTimeout` kasutades

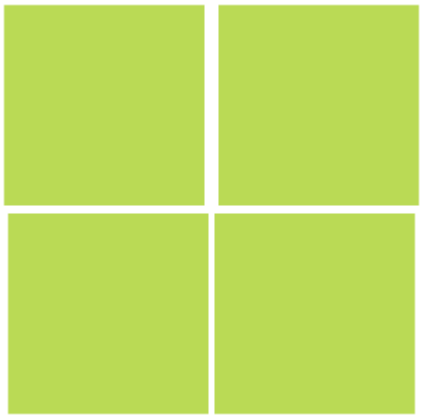
4 genereeritud HTML elementi, klassiga *box*, animeeriti kasutades `setTimeout` funktsiooni. Elemente paigutati ümber, määrates nende `left` omadust, sundides brauserit läbima tervet graafikakonveierit, mis on ajaliselt kulukas.

Veebiaadressil `http://codepen.io/ristokit/pen/vGVgKJ` on näha autori poolt kirjutatud koodi animatsiooniga. Testi sooritati kolm korda, igal korral mõõtes keskmist kaadrisagedust (vt Tabel 4).

Test nr	Keskmine kaadrisagedus
1	59,9
2	59,9
3	59,9

Tabel 4: Katse 1 mõõtmistulemused

Katse 1 keskmine kaadrisagedus oli **59,9**. Tegemist on stabiilse tulemusega, mis autori arvatest on selgitav võimeka arvuti läbi. Küll, aga, esines anomaalia, kus animeeritavad ruudud läksid teineteise suhtes nihkesse (vt Joonis 5).



Joonis 5: Sünkroonist väljas animatsioon kasutades taimerfunktsiooni

Katse 2: `requestAnimationFrame` meetodit kasutades

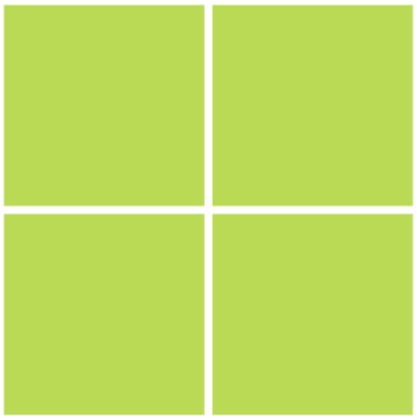
4 genereeritud HTML elementi, klassiga *box*, animeeriti kasutades `requestAnimationFrame` meetodit. Elemente paigutati ümber, määrates nende `left` omadust, sundides brauserit läbima tervet graafikakonveierit, mis on ajaliselt kulukas.

Veebiaadressil `http://codepen.io/ristokit/pen/QNJGzQ` on näha autori poolt kirjutatud koodi animatsiooniga. Testi sooritati kolm korda, igal korral mõõtes keskmist kaadrisagedust (vt Tabel 5).

Test nr	Keskmine kaadrisagedus
1	59,9
2	59,9
3	59,9

Tabel 5: Katse 2 mõõtmistulemused

Katse 2 keskmine kaadrisagedus oli taaskord **59,9**. Autor ei oska sellele põhjendust tuua, kuid täheldas, et käesoleva katse ajal ei tekkinud anomaaliat (vt Joonis 6) ja silmaga hinnates oli animatsioon tunduvalt sujuvam, tõenäoliselt kuna ruudud ei läinud nihkesse.



Joonis 6: requestAnimationFrame puhul ei tekkinud animeerimisel anomaaliat

Katse 3: Web Animations APIt kasutades;

4 genereeritud HTML elementi, klassiga box, animeeriti kasutades Web Animations APIt. Elemente paigutati ümber, määrates nende `left` omadust, sundides brauserit läbima tervet graafikakonveierit, mis on ajaliselt kulukas.

Veebiaadressil `http://codepen.io/ristokit/pen/mPzMGN` on näha autori poolt kirjutatud koodi animatsiooniga. Testi sooritati kolm korda, igal korral mõõtes keskmist kaadrisagedust (vt Tabel 6).

Test nr	Keskmine kaadrisagedus
1	59,9
2	60
3	59,9

Tabel 6: Katse 3 mõõtmistulemused

Katse 3 keskmine kaadrisagedus oli taaskord **59,93**. Autor ei oska tuua põhjendust anomaaliale, miks käesoleva katse teise testi kaadrisagedus oli pidevalt 60. Samuti ei oska autor hinnata animatsiooni sujuvust, sest silmnähtavat erinevust võrreldes eelmisega polnud. Ei ilmnenud värinat ega kaotatud kaadreid.

3.4.2 Kokkuvõtte JS animatsioonide testimisest

Katse 1 tulemuste põhjal saab visuaalse muudatuse tõttu öelda, et JavaScripti taimerfunktsioonid ei sobi animatsioonide loomiseks. Skript ei koormanud protsessorit ega

brauserit, mistõttu animatsiooni sünkroonist väljumine on selgitatav vaid taimeri kasutuse kaudu, mis kipub kaadreid vahele jätma.

Katse 2 ja katse 3 olid visuaalselt samad. Silmaga polnud võimalik mõlemat animatsiooni eristada. Ainuke erinevus tekkis lehe laadimisel, mil `requestAnimationFrame` animatsioon töötas õite pisut aeglasemalt kuna brauser pidi taustal laadima *Codepen.io* skripte ja stiile.

Autor nõustub eelnevalt viidatud Google'i ja Mozilla spetsialistidega, kes rõhusid õigete vahendite kasutamisele sujuvate animatsioonide loomiseks, milleks on `requestAnimationFrame` ja Web Animations API. Autori arvates võisid katsed olla sooritatud liialt väikeste HTML elementidega, kuid juba esimese katse korral, mis kasutas vaid nelja elementi, ilmnes silmaga märgatav erinevus.

Kokkuvõte

Käesolev bakalaureusetöö lähtus eesmärgist tutvustada sujuvate animatsioonide loomist Internetis, kasutades selleks CSSi ja JavaScripti erinevaid tehnikaid ja võtteid, sealhulgas tutvustades Web Animations APIt, mis on veel vähe toetatud ning tuntud, kuid mille potentsiaal on suur. Lisaks sai selgitatud veebilehtede visualiseerimise protsessi, mille mõistmine aitab disaineritel ja arendajatel vältida vigu, et animatsioonide jõudlus ei kannataks ning tagataks seadmele võimalikult optimaalne kaadrisagedus, milles ei esineks värinat.

Autor tutvus erinevate materjalidega W3C, veebisirvijate tootjate ning inseneride poolt, et leida võimalikult optimaalseid ja efektiivseid vahendeid loomaks CSS ja JavaScript animatsioone ning neid proovile pannes läbi sooritatud katsete. Katsetest selgus, et isegi võimsa tööjaama puhul on valede võtete kasutamine kohutava mõjuga jõudlusele. Autoris tekitas huvi Web Animations API, mis toimis sama hästi, kui CSS animatsioonid, mis näitab, et arenevad veebistandardid ja korrektsed võtted tagavad hea tulemuse.

Autor soovitab lihtsamate animatsioonide loomisel kasutada CSS3e, mis on toetatud praeguseks kõikides brauserites ning mille osad omadused on ligipääsetavad eesliitega. Keerukamate animatsioonide tarbeks soovitab autor rangel eemale hoida välistest skriptiteekidest, mis kasutavad taimerfunktsioone, sest autori katsetest selgus, et see pole parim tehnika animeerimiseks. Pigem soovitab autor tutvuda lähemalt Web Animations APIga, mis kombineerib nii CSS3e kui JavaScripti parimad omadused üheks.

Kasu võiks bakalaureusetööst olla arendajatel, disaineritel ja informaatika tudengitel, kes soovivad tutvuda eesrindliku veebitehnoloogiaga, mis mõne aja pärast võib muutuda standardiks ning rikastada paljusid veebilehti ja –rakendusi.

Summary

Title: CSS and JS animations on the Internet

The aim of this Bachelor thesis was to introduce new and improved ways of creating high performance Web animations using only CSS and JavaScript, as Adobe Flash has been neglected by browser vendors for various reasons.

Continuously evolving interfaces allow developers to have greater control over various CSS3 and JavaScript functionalities that enable them to create even complex and better optimized animations than before. Current front-end developers and designers should not hesitate to use fully unimplemented solutions to create Web animations, because most of the support already exists in modern web browsers.

The purpose of this Bachelor thesis was fulfilled. The author introduced a browsers visualization process that aims to help developers and designers understand how they could win in performance. In addition, using various CSS and JavaScript techniques that are recommended by browser vendors like Google and Mozilla, even World Wide Web Consortium, guarantees smooth animations that do not take up unnecessary resources on a device.

This paper should be useful to all web designers, developers and IT students, who are interested in new and emerging technologies that are being used to make websites and apps more interactive and visually appealing through the use of animations.

Kasutatud kirjandus

Birtles, B., Stephens, S., Danilo, A., & Atkins, T. (28. aprill 2016. a.). *Web Animations*. Allikas: Web Animations: http://w3c.github.io/web-animations/

caniuse. (30. aprill 2016. a.). *Web Animations API*. Allikas: Web Animations API: http://caniuse.com/#feat=web-animation

Google Developers. (22. august 2015. a.). *Modern Animation Fundamentals (100 Days of Google Dev) - YouTube*. Allikas: Modern Animation Fundamentals (100 Days of Google Dev) - YouTube: https://www.youtube.com/watch?v=WaNoqBAp8NI

Grigorik, I. (29. aprill 2016. a.). *Constructing the Object Model | Web Fundamentals - Google Developers.* Allikas: Constructing the Object Model | Web Fundamentals - Google Developers: https://developers.google.com/web/fundamentals/performance/critical-rendering-path/constructing-the-object-model

Grigorik, I. (29. aprill 2016. a.). *Render-tree construction, layout and paint | Web Fundamentals - Google Developers.* Allikas: Render-tree construction, layout and paint | Web Fundamentals - Google Developers: https://developers.google.com/web/fundamentals/performance/critical-rendering-path/render-tree-construction

Irish, P., & Lewis, P. (7. november 2014. a.). *High Performance Animations - HTML5 Rocks*. Allikas: High Performance Animations - HTML5 Rocks: http://www.html5rocks.com/en/tutorials/speed/high-performance-animations/

Jobs, S. (aprill 2010. a.). *Thoughts on Flash.* Allikas: Thoughts on Flash: https://www.apple.com/hotnews/thoughts-on-flash/

Lewis, P. (29. aprill 2016. a.). *Animations | Web Fundamentals - Google Developers.* Allikas: Animations | Web Fundamentals - Google Developers: https://developers.google.com/web/fundamentals/design-and-ui/animations/

Lewis, P. (30. aprill 2016. a.). *Optimize JavaScript execution | Web Fundamentals - Google Developers*. Allikas: Optimize JavaScript execution | Web Fundamentals - Google Developers:

https://developers.google.com/web/fundamentals/performance/rendering/optimize-javascript-execution

Lewis, P. (29. aprill 2016. a.). *Rendering performance | Web Fundamentals - Google Developers.* Allikas: Rendering performance | Web Fundamentals - Google Developers: https://developers.google.com/web/fundamentals/performance/rendering

Lewis, P., & Thorogood, S. (30. aprill 2016. a.). *Animations and performance | Web Fundamentals - Google Developers*. Allikas: Animations and performance | Web Fundamentals - Google Developers: http://www.html5rocks.com/en/tutorials/speed/high-performance-animations/

Mozilla. (13. september 2015. a.). *Using CSS transitions - CSS | MDN*. Allikas: Using CSS transitions - CSS | MDN: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions

Mozilla. (24. aprill 2016. a.). *@keyframes - CSS | MDN*. Allikas: @keyframes - CSS | MDN: https://developer.mozilla.org/en/docs/Web/CSS/@keyframes

Mozilla. (15. jaanuar 2016. a.). *Window.requestAnimationFrame() - Web APIs | MDN*. Allikas: Window.requestAnimationFrame() - Web APIs | MDN: https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame

Nabors, R. (22. märts 2016. a.). *Web Animation Past, Present, and Future.* Allikas: Web Animation Past, Present, and Future: http://alistapart.com/article/web-animation-past-present-and-future

W3C . (18. veebruar 2016. a.). *CSS Transitions*. Allikas: CSS Transitions: https://drafts.csswg.org/css-transitions/

W3C. (12. aprill 2016. a.). *HTML 5.1*. Allikas: HTML 5.1: https://www.w3.org/TR/html51/introduction.html#a-quick-introduction-to-html

W3C. (12. aprill 2016. a.). *Selectors.* Allikas: Selectors: https://www.w3.org/TR/CSS21/selector.html#dynamic-pseudo-classes

W3C CSS Working Group. (13. oktoober 2015. a.). *CSS Snapshot 2015*. Allikas: CSS Snapshot 2015: https://www.w3.org/TR/CSS/