TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JANNE PELTONEN

SVG:N SOVELTUVUUS VERKKOSIVUKEHITYKSEN ENSISIJAISEKSI GRAFIIKKAFORMAATIKSI

Diplomityö

Tarkastaja: Prof. Petri Ihantola
Tarkastaja ja aihe hyväksytty
29. maaliskuuta 2017

TIIVISTELMÄ

JANNE PELTONEN: SVG:n soveltuvuus verkkosivukehityksen ensisijaiseksi grafiikkaformaatiksi
Tampereen teknillinen yliopisto
Diplomityö, 70 sivua, 8 liitesivua
Joulukuu 2017
Tietotekniikan koulutusohjelma
Pääaine: Pervasive Systems
Tarkastajat: Prof. Petri Ihantola
Avainsanat: SVG, HTML5, grafiikkaformaatit, web

Verkkosivujen selaamiseen käytetään nykyään monenlaisia päätelaitteita. Tänä päivänä jo suurin osa selaamisesta tapahtuu pienikokoisilla mobiililaitteilla työpöytätietokoneiden sijaan. Päätelaitteiden monimuotoisuus asettaa haasteen perinteisille verkkosivujen grafiikkaformaateille, sillä ne ovat rasterigrafiikkaa, joka mukautuu huonosti eri kokoisille näytöille. Rasterigrafiikan sijaan voitaisiin käyttää resoluutioriippumatonta vektorigrafiikkaa, ja verkkosivukehitykseen onkin saatavilla standardoitu vektorigrafiikkaformaatti: SVG. Tässä työssä selvitetään, voidaanko kaikki verkkosivukehityksen rasterigrafiikkaformaatit ja -teknologiat korvata SVG:llä.

SVG:n soveltuvuutta rasterigrafiikan korvaajaksi tutkittiin vertaamalla sitä rasteriformaatteihin visuaalisen laadun, selaintuen, suorituskyvyn ja tiedostokoon osalta. Työssä myös luotiin katsaus sellaisiin SVG:n erityispiirteisiin, jotka ovat oleellisia verkkosivukehityksessä. Testit toteutettiin valmiin testikuvamateriaalin ja työtä varten luotujen testisovellusten avulla.

Työssä havaittiin, että SVG:llä voi ja kannattaa korvata ainakin staattinen ja tietyin ehdoin myös animoitu piirrosgrafiikka. Valokuvamateriaalin esittämiseen JPG on edelleen paras, koska tämän tyyppisen materiaalin esittäminen vektorigrafiikkana vie huomattavasti muistia. Toisaalta valokuvamateriaali on tarvittaessa mahdollista sisällyttää SVG-tiedostoon, jolloin molempien grafiikkalajien edut yhdistyvät. Lopulta dynaamisen grafiikan generointiin käytetty HTML5:n `<canvas>`-elementti soveltuu paremmin mutkikkaaseen ja käyttäjän interaktiota edellyttämättömään dynaamiseen grafiikkaan. SVG:hen on oleellisesti helpompaa liittää tapahtumankäsittelijöitä, mutta sen manipuloiminen vaatii hieman `<canvas>`-elementtiä enemmän suorituskykyä.

ABSTRACT

JANNE PELTONEN: Feasibility of SVG as the primary graphics format in web development
Tampere University of Technology
Master of Science thesis, 70 pages, 8 Appendix pages
December 2017
Master's Degree Programme in Information Technology
Major: Pervasive Systems
Examiner: Prof. Petri Ihantola
Keywords: SVG, HTML5, graphics formats, web

Nowadays people use a wide variety of devices to browse the Internet. Today most of the browsing is done using small mobile devices rather than desktop computers. The diversity of these devices poses a challenge to the traditional graphic file formats used in web development as these formats are resolution-dependent raster graphics. Resolution-independent vector graphics is a potential solution and a standardized vector graphics format exists for web development: SVG – Scalable Vector Graphics. The goal of this thesis is to study whether the raster formats could indeed be replaced with SVG in the context of web development.

To study the feasibility of SVG as a replacement for raster graphics a series of comparison tests was created. The feasibility was evaluated based on visual quality, browser support, performance and file size. Furthermore the unique nature of the format was explored to assess its features' pertinence to web development as a whole.

It was found that SVG is a feasible replacement to store and present static, drawn graphics. SVG is also a viable alternative for animations under some conditions while vector graphics is poor fit for storing photographic images. Thus JPG is still a reasonable choice where appropriate. Finally the canvas element of HTML5 is a sound choice for complex dynamically generated graphics while SVG should be preferred in case of low volume manipulation and prominent user interaction.

ALKUSANAT

Tavallisesti tässä kohtaa kiitetään yritystä, joka tilasi työn. Omani on kuitenkin sikäli poikkeava, että työ on kirjoitettu kokonaan omaan piikkiini. Siten olen saanut valita aiheen puhtaasti sen perusteella, mikä herättää oman intohimoni. Toivottavasti se näkyy ja välittyy sinulle lukijana... mieluiten positiivisesti.

Matkani koko IT-alalle on lähtenyt peruskouluaikojen innostani piirtelyyn ja edennyt tietokonegrafiikan kautta verkkosivuohjelmointiin sekä käyttöliittymäsuunnitteluun. Ei siis ole yllättävää, että kaikki nämä yhdistävänä teknologiana SVG on saanut minut innostumaan jopa niin paljon, että tahdon jakaa tietojani asiasta muiden kanssa.

Tahdon kiittää tarkastajanani ja ohjaajanani toiminutta apulaisprofessori Petri Ihantolaa asiantuntevasta ja rakentavasta palautteesta sekä hyödyllisestä ohjauksesta. Oikoluvusta kiitoksen ansaitsevat Jarmo Palviainen, Esa Väntsi sekä Robert von Zweygbergk. Kiitän syvästi äitiäni ja kummivanhempiani, jotka ovat tukeneet opintouraani mahdollistaen koko diplomi-insinöörin tutkintoni. Äidilleni olen myös kiitollinen kasvatuksesta arvostamaan uteliaisuutta, tiedon hankkimista ja sen ymmärtämistä.

Usein unohdetaan osoittaa kiitollisuutta Suomen opetusjärjestelmää kohtaan. Itse koen, että ilmainen, jopa tuettu koulutus kaikille perustuen motivaatioon ja kyvykkyyteen eikä sosioekonomiseen statukseen on ensiarvoisen tärkeä etu. Valmistuttuani siirryn kiitollisena antamaan veroeurojeni muodossa yhteiskunnalle takaisin, jotta vastaava mahdollisuus voidaan tarjota myös tulevaisuuden menestyjille Suomen kilpailukyvyn ylläpitämiseksi.

Lopuksi tervehdykseni kaikille niille hienoille ihmisille, jotka yliopisto on tuonut ympärilleni. Teidän ansiostanne opiskeluaikani teekkarilakin alla on ollut elämäni hienointa aikaa!

Tampereella 22.11.2017

Janne Peltonen

SISÄLLYS

KUVALUETTELO

TAULUKKOLUETTELO

LISTAUSLUETTELO

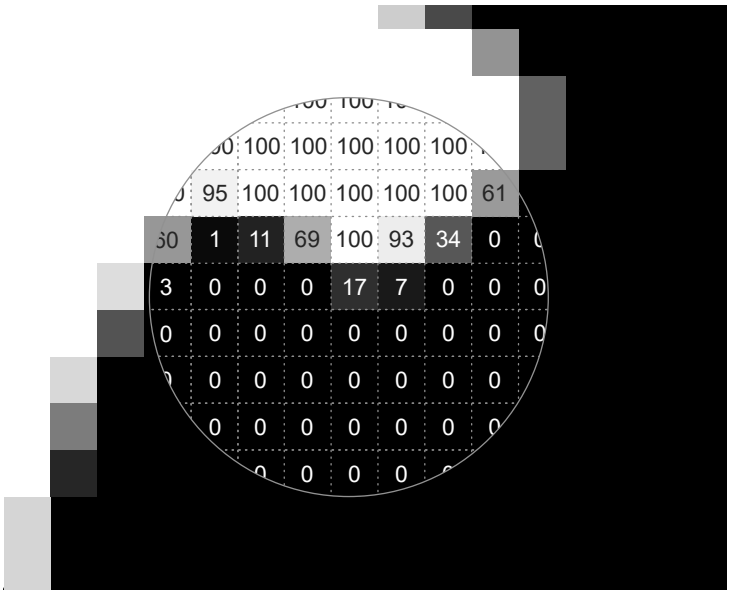
LYHENTEET JA KÄSITTEET

CSS	Cascading Style Sheets, HTML-dokumenttien tyylittämiseen käytettävä kieli
DPI	Dots Per Inch, pistettä tuumalla, pikselitiheyden mittayksikkö
FPS	Frames Per Second, ruutua sekunnissa, ruudunpäivitystaajuuden mittayksikkö
HTML	HyperText Markup Language, kieli hypertekstidokumenttien kuvaamiseen
SMIL	Synchronized Multimedia Integration Language, XML-pohjainen kieli multimedian kuvaamiseen
SVG	Scalable Vector Graphics, XML-pohjainen kieli vektorigrafiikan kuvaamiseen
URL	Universal Resource Locator, resurssin sijainnin osoittava merkkijono, joka voi myös itsessään sisältää resurssin (data-URL)
W3C	Worldwide Web Consortium, standardisointiorganisaatio
XML	eXtensible Markup Language, geneerinen merkintäkieli datan kuvaamiseen

Boidi	Craig Reynoldsin *Boids*-parvisimulaation yksi jäsen (engl. *Boid*, "Birdoid object")
Chrome	Google Chrome -internetselain
Edge	Microsoft Edge -internetselain, Internet Explorer -selaimen seuraaja
Firefox	Mozilla Firefox -internetselain
Hahmonnus	engl. *render*, ohjelmallinen kuvan muodostaminen
Resoluutio	Näytön tai kuvan pikselien lukumäärä

1. JOHDANTO

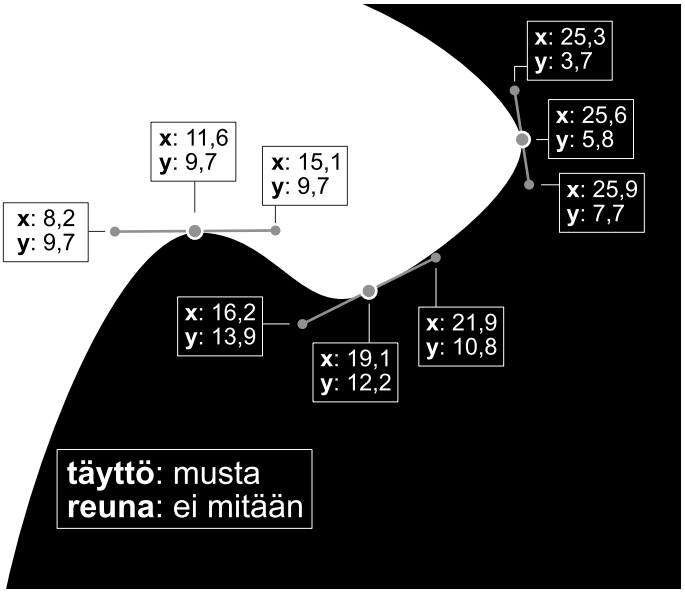
Tällä hetkellä valtaosa verkkosivuilla käytössä olevasta grafiikasta on rasterigrafiikkaa, eli se pohjautuu pikseleihin. Rasterigrafiikkakuva tallennetaan merkitsemällä muistiin kuvan jokaisen pikselin väriarvot. Menetelmää havainnollistaa kuva 1.1. Rasterigrafiikka on saavuttanut suosionsa yksinkertaisuutensa vuoksi, sillä tallennustapa on hyvin linjassa sen kanssa, miten kuvapisteisiin perustuvat näyttölaitteet esittävät grafiikkaa.



Kuva 1.1 *Rasterigrafiikan tallennusperiaate. Jokaisesta pikselistä tallennetaan väritiedot numeerisesti; tässä mustaa vastaa arvo 0 ja valkoista arvo 100. Harmaasävyt sijoittuvat tälle numerovälille. Värikuvissa jokainen värikomponentti on tallennettava erikseen ja esimerkiksi PNG-kuvissa tarvitaan vielä arvo peittävyydelle eli alfalle.*

Vektorigrafiikka koostuu matemaattisista määritteistä, joiden pohjalta kuva hahmonnetaan näytöllä esitettäväksi vasta katseltaessa [1]. Määritteet voivat olla esimerkiksi alkeismuotoja, kuten suorakulmioita, ellipsejä tai polkuja. Muodoista tallennetaan kulmien tai ankkuripisteiden koordinaatit ja vaikkapa visuaalisia tyylejä, kuten täyteväri tai reunaviivan paksuus. Tätä menetelmää havainnollistaa kuva 1.2.

Vektorigrafiikan pohjautuminen matemaattisiin mallinnoksiin mahdollistaa sen, et-



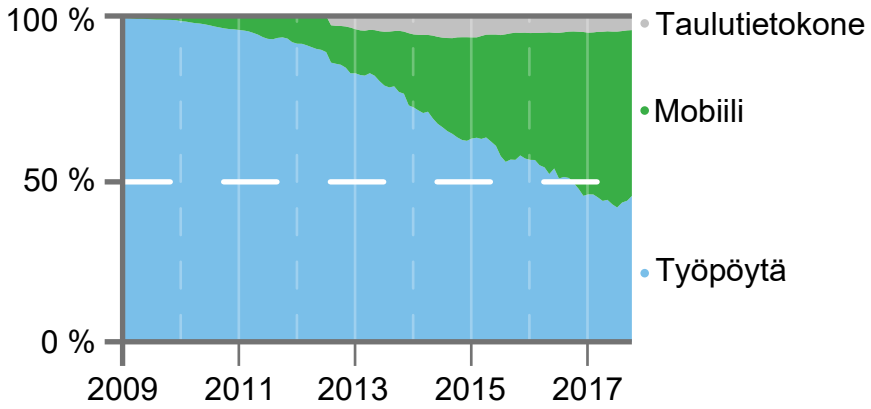
Kuva 1.2 *Vektorigrafiikan tallennusperiaate. Kuvassa näkyy muotoa rajaava Bézier-käyrä, jonka ankkuripisteet koordinaatteineen on merkitty näkyviin. Muodon täyttöväriksi on määritetty musta, eikä sille tule piirtää reunaviivaa.*

tä kuvioista voidaan laskea näytepisteitä mielivaltaisella tarkkuudella. Rasterigrafiikassa jo lasketut näytepisteet ovat ainoa käytettävissä oleva informaatio. Siksi rasterikuvan natiiviresoluutiosta poikkeava esitystarkkuus edellyttää puuttuvan informaation approksimoimista interpolomalla olemassa olevista pikseleistä.

Jos puhutaan tietokoneella luodusta grafiikasta, niin vektorigrafiikkaformaatissa on mahdollista tallentaa tietokoneen ymmärtämässä muodossa kuvaus siitä, mitä kuva esittää. Kuva rasteroidaan eli muunnetaan pikselipohjaiseen esitystapaan vasta tulostamista varten. Tietokone voi soveltaa rasteroimattomaan grafiikkaan matemaattisia alkeismuunnoksia, kuten kiertoa tai skaalausta [2]: esimerkkikuvan 1.2 skaalaaminen onnistuisi yksinkertaisesti kertomalla kaikki esillä olevat koordinaatit jollain luvulla. Tämä mahdollistaa sen, että vektorigrafiikkana tallennettu kuva voidaan tulostaa aina päätelaitteen tarkkuudella, oli kyseessä sitten älypuhelin, tietokone, tulostin tai painokone.

Rasterigrafiikan kiinteän resoluution rajoite ei ole ollut merkittävä ongelma internetkäytössä niin kauan, kun selaamiseen käytettiin lähes ainoastaan työpöytä- tai kannettavia tietokoneita. Niissä näyttöjen pikselitiheydet pysyivät melko vakioina, sillä suuremmissa näytöissä on yleensä suurin piirtein samassa suhteessa enemmän pikseleitäkin. Nykyään tämä kaava ei enää kuitenkaan välttämättä päde. Kuluttajamarkkinoille on ilmestynyt työpöytänäyttöpäätteitä, joiden resoluutiot voivat olla FullHD:stä 5K:hon, eli niiden vaakaresoluutio voi olla 1920 pikselistä 5120 pikseliin.

Vielä suurempi haaste käy ilmi kuvasta 1.3, jonka mukaan työpöytälaitteiden osuus kaikesta internetselaamisesta on jo alle puolet ja suurempi osa selailusta tapahtuu taskukokoisilla mobiililaitteilla. Niiden näytöt ovat fyysiseltä kooltaan pieniä, mutta resoluutio voi hyvinkin vastata työpöytäkäyttöön tarkoitettujen näyttölaitteiden resoluutioita. Tästä syystä yllä listatut rasterigrafiikan rajoitteet ovat muodostuneet ongelmaksi, johon on syytä löytää ratkaisu.



Kuva 1.3 *Internetselaamiseen käytettyjen päätelaitetyyppien osuuksien kehitys maailmanlaajuisesti tammikuusta 2009 lokakuuhun 2017 (tiedot [3]). Työpöytäkäytön osuus on laskenut hiljalleen koko tarkastelujaksolla ja päätyi vähemmistöön vuonna 2016.*

Vektorigrafiikka on potentiaalinen ratkaisu myös verkkosivuilla esitettävän grafiikan resoluutioriippuvuuteen, ja verkkosivuteknologioiden standardeja kehittävä W3C (*WorldWide Web Consortium*) onkin jo kehittänyt julkisen ja ei-kaupallisen vektorigrafiikkaformaatin. Formaatin nimi on **SVG** eli *Scalable Vector Graphics* (suom. skaalautuva vektorigrafiikka). Kaikki tärkeimmät selaimet (Applen Safari, Googlen Chrome, Microsoftin Internet Explorer, Mozillan Firefox [4]) ovat tukeneet SVG:n esittämistä jo vuodesta 2011 [5] ja W3C:n työstämä HTML5-standardi lisää SVG:n mahdollisuuksia verkkosivukehityksessä entisestään [6].

Resoluutioiden vaihtelevuuteen näyttäisi siis olevan tarjolla ratkaisu, mutta SVG ei silti näytä saavuttaneen suosiota. HTML5-aiheisessa kirjallisuudessa SVG:tä yleensä vain sivutaan ja tieto on jonkin verran jopa ristiriitaista. Jukka K. Korpelan kirjassa *HTML5 - Uudet ominaisuudet* vuodelta 2011 todetaan, että "... SVG on alkanut kiinnostaa monia, vaikka sen tuki on vielä varsin puutteellinen web-käyttöä ajatellen." [7, s. 181] Toisaalta Korpelan ja Tero Linjaman teoksessa *Web-suunnittelu* vuodelta 2005 lukee, että "Vektorigrafiikoita ei web-sivuilla vielä juurikaan näe, vaikka uusimmat selaimet tukevat jo svg-grafiikkaa." [8, s. 232] Teoksessa *Introducing HTML5* SVG esitellään vain `<canvas>`-elementin vaihtoehtona dynaamisen grafiikan toteuttamiseksi, mutta teos ei käsittele sen mahdollisuuksia korvata rasterigrafiikan tiedostoformaatteja [9]. Katsaus tuoreempiin tieteellisiin teoksiin osoittaa, että

SVG:hen toteutusteknologiana suhtaudutaan nykyäänkin varauksella ja erinäisistä syistä rasteripohjaisiin tekniikoihin turvaudutaan edelleen [10]–[12].

Kirjallisuusselvitys ei siis tuota selkeää kuvaa SVG:n tilanteesta. Tämän diplomityön tarkoituksena on koota ajantasainen, kattava ja luotettava katsaus siitä, onko rasterigrafiikkaformaatteja edelleen tosiasiassa syytä suosia vai sopisiko SVG verkkosivukehityksen ensisijaiseksi grafiikkaformaatiksi. Jotta näin olisi, SVG:n pitäisi olla tarpeeksi laajasti internetselainten tukemaa, tiedostokooltaan kompaktia, visuaaliselta laadultaan riittävää ja suorituskykyvaatimuksiltaan matalaa. Näitä tullaan käyttämään mittareina, kun työssä verrataan SVG:tä verkkosivukehityksen tyypillisiin rasteriformaatteihin.

Verkkosivukehityksen yleisimmät tiedostoformaatit ovat GIF, JPG ja PNG. Kullekin formaatille sopivin käyttökohde on esitetty seuraavassa [1], [8], [13], [14]:

GIF	Väriavaruudeltaan suppea, yksinkertainen piirrosgrafiikka sekä animaatiot.
JPG	Valokuvamainen grafiikka, kuten valokuvat ja fotorealistinen 3D-grafiikka.
PNG	Väriavaruudeltaan laaja, kohtalaisen yksinkertainen piirrosgrafiikka.

SVG:n tulisi soveltua näihin käyttökohtiesiin asianosaisia rasterigrafiikkaformaatteja paremmin, jotta se olisi kelvollinen verkkosivukehityksen ensisijaiseksi grafiikkaformaatiksi. Työn piiriin kuuluu myös dynaaminen eli selaimessa generoitava ja/tai manipuloitava grafiikka. HTML5:n myötä dynaamista grafiikkaa varten on kehitetty erityinen piirtoalue- eli `<canvas>`-elementti. SVG:tä vertaillaan tässä työssä myös siihen.

SVG on hyvin erikoislaatuinen grafiikkaformaatti verrattuna rasteriformaatteihin. Tästä syystä on syytä selvittää myös se, miltä osin sen erityispiirteet ovat oleellisia verkkosivukehityksen saralla. Työssä etsitään vastaus siis kahteen tutkimuskysymykseen:

1. Kykeneekö SVG korvaamaan verkkosivukehityksessä vallalla olevat rasterigrafiikkaformaatit?

2. Mitä rasterigrafiikasta puuttuvia erityispiirteitä SVG:ssä on verkkosivutekniikkana?

Työssä pyritään tuottamaan yleistasoinen mutta kattava kuva tutkimuskysymyksistä. Tarkoitus on, että havaintojen pohjalta voidaan tehdä päätös SVG:n käytöstä verkkosivuprojekteissa, mutta monimutkaisempiin erityistapauksiin työn ei ole tarkoitus ottaa kantaa. Työn yleistasoisuus voi myös ohjata mahdollista jatkotutkimusta tuomalla esille oleellisia SVG:n käyttöön liittyviä kysymyksiä. Koska SVG kykenee esittämään ainoastaan kaksiulotteista grafiikkaa, WebGL:ään pohjautuvat kolmiulotteisen grafiikan esitystekniikat eivät kuulu tämän työn aihepiiriin.

Työn aluksi luvussa 2 esitellään työn aihepiirin teknilliset taustat ja luodaan tarkempi katsaus muuhun aihetta käsittelevään tutkimusmateriaaliin. Se käy läpi näyttölaitteiden, grafiikkaformaattien ja internetin historiaa sekä erittelee SVG:tä koskevia väitteitä muista tieteellisistä lähteistä. Luku 3 esittelee vertailumenetelmät, joilla SVG:tä verrataan rasterigrafiikkaformaatteihin ja -tekniikoihin. Luvussa 4 listataan vertailujen tulokset sekä analysoidaan ne, eli muodostetaan vastaus tutkimuskysymykseen 1. Lukuun 5 on koottu SVG:n erityispiirteet verkkosivukehityksen grafiikkaformaattina, eli se vastaa tutkimuskysymykseen 2. Lopulta luku 6 kokoaa yhteen työn tuottamat löydökset ja niistä vedettävät johtopäätökset. Luvussa luodaan myös lyhyt katsaus käsiteltyjen tekniikoiden tämänhetkiseen kehitykseen ja lopulta pohditaan sitä, miten tutkimuskysymyksiin voisi perehtyä syvällisemmin jatkotutkimuksin.

2. TAUSTAA

Ymmärtääksemme, miksi rasterigrafiikkaformaatit ovat saavuttaneet suosionsa ja miksi ne ovat muodostuneet nyttemmin ongelmallisiksi, on syytä katsoa menneeseen. Tässä luvussa luodaan katsaus näyttölaitteiden ja tiedostoformaattien historiaan. Sen jälkeen selvitetään tarkemmin, miten päätelaitteiden monipuolistuminen on saanut rasterigrafiikan rajoitteet muodostumaan ongelmaksi. Luvussa esitellään myös tarkemmin työn aihepiiriin liittyvät teknologiat. Lopuksi luvussa luodaan katsaus siihen, miten SVG:tä on käsitelty muissa tieteellisissä lähteissä viimevuosina.

2.1 Grafiikan esittämisen ja tallentamisen historiaa

Ensimmäinen varsinainen näyttölaite koostui elektronisuihkusta, joka aiheutti luminesenssi-ilmiön osuessaan fosforisoivaan näyttöpintaan. Tällaista näyttölaitetta käytetään oskilloskoopeissa, joiden tehtävä on visualisoida signaaleja. Menetelmä on luonteva, sillä signaali on mahdollista vähäisellä käsittelyllä ohjata magneetteihin tai sähköjohtimiin, jotka puolestaan vaikuttavat elektronisuihkun liikerataan. Tällainen näyttölaite on nimeltään katodisädeputki (*Cathode Ray Tube, CRT*), ja se keksittiin vuonna 1897. [15]

Oskilloskoopissa signaaleja voidaan visualisoida siis varsin yksinkertaisesti, mutta toisaalta signaalia ohjailemalla näytölle voidaan tuottaa halutunlaista grafiikkaa. Näin tuotettu grafiikka on luonteeltaan alkeellista vektorigrafiikkaa. Monipuolisempi menetelmä grafiikan tuottamiseksi on jakaa näyttöpinta pieniin kenttiin, joihin elektronisuihku ohjataan yhteen kerrallaan. Suihkun voimakkuutta säätelemällä voidaan näyttöpinnalle piirtää pikseleistä koostuva sävykuva eli rasterikuva. [15]

Niin kauan, kun tällaisia näyttöjä käytettiin yleiskäyttöisten tietokoneiden kanssa, rasterigrafiikka oli täydellinen ratkaisu. Vaikka näyttöjen resoluutio kasvoi pikkuhiljaa, myös näyttöjen fyysinen koko kasvoi enemmän tai vähemmän samassa suhteessa. Siten rasterigrafiikkana tallennettu ja esitetty kuvamateriaali oli käytännössä aina sopivan tarkkaa, eikä vaihtoehdoille ollut merkittävää tarvetta. Lisäksi tietokoneiden suorituskyvyn kannalta oli parempi, että grafiikka oli tallennettuna suoraan mahdollisimman ”esitysvalmiiseen” muotoon [1].

Rasterigrafiikan rajoitteet olivat kuitenkin tiedossa jo tuolloin painoteollisuudessa [2]. Tavallinen pikselitiheys näyttölaitteilla oli 1990-luvulla 72 pistettä tuumalla, mutta ollakseen painokelpoista grafiikan tuli jo silloin olla tarkkuudeltaan vähintään 300 pistettä tuumalla. Näin tarkan rasterigrafiikan tallentaminen ja käsittely vaati sen ajan resursseilla paljon suorituskykyä ja tallennustilaa. Niinpä painotuotteiden työstämiseen on hyödynnetty vektoripohjaisia työkaluja jo 1980-luvulta lähtien. Tuolloin julkaistiin ensimmäinen vektorigrafiikkaa tukeva ja laitteistoriippumaton sivunkuvauskieli PostScript [16], joka voidaan toisaalta mieltää myös vektoripohjaisena grafiikkaformaattina.

2.2 Internetin selaamisen päätelaitteet nykyään

Taulukossa 2.1 on listattuja nykypäivänä tyypillisiä päätelaitteita ja niiden näyttöpaneelien ominaisuuksia. Siinä on kolme tavallista tietokoneen monitoria, joiden pikselitiheydet ovat melko lähellä toisiaan. Vaikka tiheydet ovatkin kasvaneet jonkin verran näyttölaitteiden historian aikana, monitorien tapauksessa tiheydet eivät ole hajaantuneet merkittävästi. Taulukosta huomataan kuitenkin kolme laitetta, joiden resoluutio on 1920 kertaa 1080 pikseliä, mutta niiden fyysiset koot muodostavat jopa kertaluokan skaalan: Panasonic-televisio on lävistäjältään 50 tuumaa kun taas Huawein älypuhelin on vain 5,2 tuumaa. Niiden pikselitiheydet noudattavat kääntäen samaa kaavaa: television pikselitiheys on 44 DPI (*Dots Per Inch* eli pistettä tuumalla) kun taas älypuhelimessa se on 424 DPI. Internetin selaamiseen käytettyjen päätelaitteiden näyttökoot vaihtelevat siis huomattavasti, joten verkkosivukontekstissa ei voi enää olettaa mitään tyypillistä pikselitiheyttä.

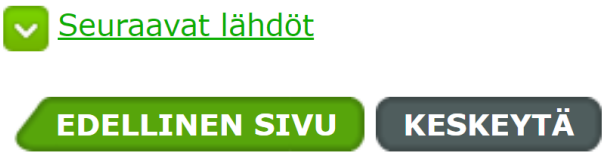
Taulukko 2.1 *Eräiden tyypillisten päätelaitteiden näyttöjen resoluutio ja pikselitiheys.*

Valmistaja ja laitemalli	**Tyyppi**	**Fyys. koko (*in*)**	**Vaaka-resoluutio**	**Pysty-resoluutio**	**Pikseli-tiheys (*DPI*)**	**Skaalaus**
Panasonic TX-50CS620E	Televisio	50	1920	1080	**44**	1,5
HP Pavilion w2408h	Monitori	24	1920	1080	**90**	1
Samsung SM 933HD	Monitori	17	1360	768	**92**	1
Eizo FlexScan 27"	Monitori	27	2560	1440	**109**	1
Apple iMac 27"	Tietokone	27	5120	2880	**218**	2
Jolla Jolla	Älypuhelin	45	540	960	**245**	1,5
Microsoft Surface Pro 4	Tabletti	12,3	2736	1824	**267**	2
Huawei Honor 7	Älypuhelin	5,2	1080	1920	**424**	3

Päätelaitteiden monimuotoisuudesta seuraa, että rasterigrafiikan fyysinen koko voi esitettäessä vaihdella paljonkin päätelaitteesta riippuen; pöytätietokoneella luotu

kuva voi älypuhelimessa näkyä niin pienenä, että sen sisällöstä ei saa selvää. Kuvan voi toki skaalata suuremmaksi liian tarkkoja näyttöjä varten, ja niin usein tehdäänkin; taulukossa 2.1 on listattuna myös skaalauskerroin, joka kertoo, kuinka paljon internetselain Google Chrome suurentaa verkkosivun sisältöä kyseisellä laitteella. Esimerkiksi Honor 7 -laitteella Chrome hahmontaa verkkosivun oletuksena virtuaalisesti 360 pikselin leveydellä ja suurentaa sen kolminkertaiseksi vastaamaan laitteen näytön todellista, 1080 fyysisen pikselin leveyttä. Muutoin 12 pikselin korkuinen teksti tulostuisi laitteen näytölle alle millimetrin korkuisena vaikka *HP w2408h* -työpöytänäytöllä se tulostuisi lukukelvollisena, 3,4 millimetrin korkuisena. Tämän ongelman kuvasi jo vuonna 2000 tunnettu käytettävyysasiantuntija Jakob Nielsen kirjassaan *WWW-suunnittelu* (*Designing Web Usability*). Hän kehottaakin verkkosivujen kehittäjiä julkaisemaan resoluutiosta riippumattomia verkkosivuja.

Tekstisisällön tapauksessa vaihteleva pikselitiheys on triviaali haaste, mutta rasterigrafiikka on luontaisesti kiinteäresoluutioista. Jotta rasterikuva voidaan esittää ominaisresoluutiostaan poikkeavassa koossa, sitä on interpoloitava, koska kuvasta ei ole tiedossa tarpeeksi monen pikselin väriarvoja [1]. Siksi rasterigrafiikan skaalaaminen ylöspäin heikentää kuvan visuaalista laatua, minkä tarkkasilmäinen selailija voi huomata kuvan puuroutuvina reunoina. Tämä näkyy kuvasta 2.1, joka on kuvankaappaus VR:n verkkokaupan painikkeista. Kaappausta otettaessa selaimen näkymä oli suurennettu kolminkertaiseksi, joka vastaa suurinta taulukossa 2.1 esiintyvää skaalauskerrointa. Kuvasta nähdään, että painikkeiden taustagrafiikka on rasterigrafiikkaa, sillä se pikselöityy. Jos painikkeet olisi toteutettu vektorigrafiikalla, ne säilyisivät yhtä terävinä kuin niiden sisältämä teksti. Siitä huolimatta tällä hetkellä tyypillisempi ratkaisu on luoda grafiikasta versiot eri tarkkuuksille ja noutaa JavaScriptin avulla kulloinkin oikea [17].



Kuva 2.1 *Kuvankaappaus VR:n verkkokaupan (shop.vr.fi) käyttöliittymästä kolminkertaiseksi skaalattuna. Skaalauskerroin vastaa selaamista Honor 7 -älypuhelimella. Rasterigrafiikkapohjaisten painikkeiden reunat ovat pikselöityneet tekstisisältöön verrattuna.*

Resoluutioriippumattoman esitystavan lisäksi toinen vektorigrafiikan sovelluskohde liittyy tilanteisiin, jossa visuaalisen materiaalin sisältö tulee olla semanttisesti kuvailtua ohjelmallisen tulkinnan ja manipuloinnin mahdollistamiseksi. Vektorigrafiikan avulla tietokoneen saatavilla pysyy tieto siitä, mitä käsiteltävä grafiikka esittää ja kykenee siten helpommin manipuloimaan sitä. Tämä mahollistaa esimerkiksi ruu-

dunlukijoiden paremman toiminnan ja sisällön dynaamisen animoinnin. Sen vuoksi muun muassa Adobe Flash hyödynsi sitä [18], mutta tähän palataan myöhemmin.

2.3 Verkkosivuteknologioiden kehitys

Tähän mennessä olemme perehtyneet grafiikkalajeihin ja ymmärtäneet vektorigrafiikan ja rasterigrafiikan erot. Työn toinen oleellinen aihepiiri on verkkosivukehitys, joten seuraavaksi luodaan katsaus sen historiaan. Aliluvun tiedot perustuvat Bill Kennedyn ja Chuck Muscianon teokseen *HTML: The Definitive Guide* [19] ellei toisin esitetä.

Tietokoneita käytettiin alunperin pelkästään tutkimuskäytössä, koska ne vaativat huomattavia resursseja. Jossain kohtaa katsottiin hyödylliseksi, että tutkimusraportteja olisi mahdollista jakaa tietokoneiden välillä. Tämän seurauksena syntyi tietoverkko, joka on nykyään kehittynyt internetiksi. Dokumenttien internetjakoa varten kehitettiin standardi, joka määrittelee merkintäsäännöt muotoillun tekstin tallentamiseen ja jakamiseen internetin kautta. Siinä tekstien vaihtelevat merkitykset merkittiin tekstin sekaan tunnistein, joilla merkittävä teksti ympäröitiin. Merkintätapa on nimeltään HTML (*HyperText Markup Language*), ja sen ensimmäinen versio julkaistiin vuonna 1993.

HTML kehittyi pikkuhiljaa siten, että sillä saatettiin luoda laajempia verkkosivustoja yksittäisten dokumenttien sijaan. Alunperin kuvia oli mahdollista käyttää vain tekstin tukena, mutta myöhemmin kieli kehittyi mahdollistamaan dokumenteille kokonaisvaltaisen visuaalisen ilmeen. Merkittävä uudistus oli HTML:n version 4 myötä vuonna 1996 kehitetty tyylikieli CSS (*Cascading Style Sheets*), jolla voi määrittää kokonaisen sivuston ulkoasun asettamisen kootusti yhdestä paikasta. Varsinaista kuvamateriaaliakin oli tällöin mahdollista käyttää pelkästään ulkoasutarkoituksessakin varsinaisen sisällön ohella esimerkiksi taustakuvina tai painikkeina.

HTML:n silloinen alkeellisuus johti siihen, että yksityiset toimijat pyrkivät kehittämään monipuolisempia tekniikoita verkkosivukehitykseen. Yksi tällainen tekniikka oli Shockwave Flash, joka tunnetaan nykyisin nimellä Adobe Flash. Adobe Flash [18] oli pitkään paljon käytetty teknologia monipuoliseen interaktiivisuuteen perustuvien selainsovellusten tekemisessä. Se julkaistiin alunperin vuonna 1995 ja hyödynsi vektorigrafiikkaa tämän helpon animoitavuuden ja kompaktiuden vuoksi.

Flashin ansiosta vektorigrafiikka oli verkkosivuilla yleistä jo tuolloin, mutta Flashin kaupallisuus ja yksityisomistuksellisuus olivat ongelmia. Soveltuvaa avointa standardia ei ollut olemassa, kunnes W3C julkaisi SVG:n ensimmäisen version vuonna

2001. SVG:stä osattiin innostua heti ja juuri niistä samoista syistä, miksi se on nykypäivänä viimein alkanut todella nostaa päätään: resoluutioriippumattomuus, itsekuvailevuus ja ohjelmoitavuus [20]. Flash oli kuitenkin silloin juurtunut vahvasti sisällöntuottajien työkaluvalikoimaan, eivätkä selainvalmistajat tai sisällöntuottajat katsoneet aiheelliseksi ryhtyä tukemaan SVG:tä [20]. Microsoftillakin oli tuolloin oma saman tyylinen ja kilpaileva teknologia, VML (*Vector Markup Language*), minkä vuoksi Microsoft ei toteuttanut Internet Explorer -selaimeensa tukea SVG:lle ennen versiota 9 vuonna 2011 [7], [21].

Flashin suosio alkoi hiipua 2010-luvulla tietoturvaongelmien ja kaupallisuuden vuoksi. Ensimmäisen merkittävä isku tuli Applelta, joka päätti jättää Flash-tuen pois laitteistaan [22]. Ratkaisu oli mahdollinen, koska HTML5:n (sisältäen SVG:n) katsottiin tarjoavan yhtä monipuoliset mutta avoimet ja standardoidut työkalut verkkosovellusten tekoon. Lopulta vuonna 2015 Adobe itse julkaisi tiedotteen, jossa sisällönkehittäjiä kehotetaan siirtymään Flashista HTML5:een [23]. Heinäkuussa 2017 Adobe ilmoitti, että Flashin ylläpito lopetetaan kokonaan vuoden 2020 lopussa [24].

HTML:n tuoreimman version, HTML5:n myötä merkintäkielen visuaaliset ominaisuudet ovat kehittyneet jo niin pitkälle, että verkkosivustojen lisäksi voidaan puhua jo kokonaisista verkkosovelluksista [6]. Verkkosovellukset ovat pohjimmiltaan verkkosivuja, joiden käyttö pohjautuu monipuoliseen käyttäjäinteraktioon ja ne edellyttävät rikasta graafista käyttöliittymää. Verkkosivuilla siis tarvitaan nykyisin huomattavaa määrää grafiikkaa, jonka olisi samalla mukauduttava monen kokoisille päätelaitteille.

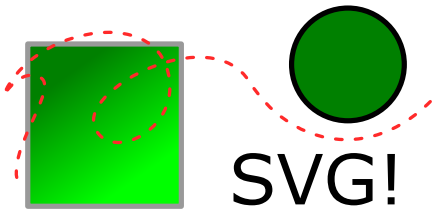
2.4 Lyhyt johdatus SVG:hen

> "Things to watch: SVG - Scalable Vector Graphics - at last, graphics which can be rendered optimally on all sizes of device."
>
> — Sir Tim Berners-Lee [25]

Nyt kun tunnemme taustat sekä grafiikkaformaateista että verkkosivukehityksestä, voimme tutustua teknologiaan, jossa molemmat yhdistyvät. SVG-kuvia tullaan tässä työssä käsittelemään melko yksityiskohtaisellakin tasolla, ja siksi tämä aliluku tarjoaa teknisluonteisen johdannon SVG-kuvien rakenteeseen. Aliluku on mukailtu SVG:n standardista [26].

SVG eli skaalautuva vektorigrafiikka (*Scalable Vector Graphics*) on tapa kuvata vektorigrafiikkaa XML-dokumenttina [1]. SVG on W3C:n avoin standardi, joka laadittiin korvaajaksi useille erilaisille suljetuille vektorigrafiikkaformaateille. SVG-standardia on ollut työstämässä edustajia näiden suljettujen formaattien omistajayhtiöistä ja selainvalmistajilta sekä useita riippumattomia asiantuntijoita.

SVG-kuvien rakenteen esittelemisen pohjana tutkitaan kuvaa 2.2, jonka merkintäkoodi on esitetty listauksessa 2.1. Pohjimmiltaan SVG koostuu alkeismuodoista, joita vastaavat muiden muassa elementit `<rect>`, `<circle>`, `<ellipse>`, `<polygon>` ja `<path>`. Näillä elementeillä on attribuutit, joilla muodoille voi asettaa sijainti ja koko, kulma- tai ankkuripisteitä tai vaikkapa ympyrän tapauksessa säteen. Alkeismuotojen esittäminen SVG:n merkintäkoodissa käy ilmi listauksen 2.1 riveiltä 10-18. Oleellinen elementti on myös rivillä 19 esiintyvä `<text>`, jolla grafiikkaan voidaan liittää tekstiä siten, että se säilyy semanttisestikin tekstinä. Visuaalisen tyylin asettamiseksi muodoille voidaan määrittää reunaviivan (`stroke`) ja täytön (`fill`) tyylit. Näistä molemmat voivat olla tasainen väri, lineaarinen tai radiaalinen väriliuku tai kuvio. Reunaviivalle voidaan lisäksi määrittää katkotus (`stroke dash array`) sekä päiden ja kulmien tyylit. SVG:ssä on myös `<image>`-elementti, jolla kuvaan voidaan lisätä ulkopuolista kuvamateriaalia – myös rasterigrafiikkaa. Kaikille elementeille voi asettaa viittaamista varten HTML-elementtien tavoin tunnisteen `id`-attribuutilla sekä luokkia `class`-attribuutilla.



Kuva 2.2 *Esimerkki SVG-kuvasta.*

```
1 <svg xmlns="http://www.w3.org/2000/svg" height="40mm" width="80mm"
      version="1.1" xmlns:xlink="http://www.w3.org/1999/xlink"
3     viewBox="0 0 284 142">
  <defs>
5  <linearGradient id="liuku">
    <stop stop-color="#008000" offset="0"/>
7   <stop stop-color="#0f0" offset="1"/>
   </linearGradient>
9 </defs>
  <rect height="94" width="89" y="29" x="28"
```

[1] XML (*eXtensible Markup Language*) on merkintäkieli, jolla voidaan paitsi esittää, myös kuvailla informaatiota. Se on SVG:n tavoin W3C:n kehittämä. [27]

```
11        stroke-width="3" fill="url(#liuku)" stroke="#999"/>
    <circle cx="2.1e2" stroke="#000" cy="40"
13          r="32" stroke-width="3" fill="#008000"/>
    <path d="m2.6e2 62c-40 36-80 24-1e2 -12-30-35-1e2 4-94 26 9
15           23 54 1 44-34-14-35-82-21-96 16 0 0 9.8-16 21-10s-21
             43-13 62"
17        stroke="#ff2a2a" stroke-linecap="round"
          stroke-dasharray="3, 7" stroke-width="1.9" fill="none"/>
19  <text font-size="40px" y="121" x="180"
          font-family="sans-serif" line-height="125%">
21   SVG!
    </text>
23 </svg>
```

Listaus 2.1 *Kuvan 2.2 SVG-merkintä.*

SVG:n elementtejä voidaan ryhmittää kokonaisuuksiksi asettamalla ne `<g>`-elementin sisään. Näin kaikille ryhmän sisältämille voidaan määrittää esimerkiksi yhteinen alkeismuunnos tai tunniste näin:

```
1 <g id="my-group" transform="rotate(45)">
2   <circle cx="30" cy="20" r="50" />
3   <rect x="10" y="5" width="40" height="30" />
4 </g>
```

Tunnistetta käyttämällä kertaalleen määritelty elementti voidaan hahmontaa uudelleen `<use>`-elementin avulla johonkin toiseen sijaintiin. `<svg>`-elementti voi sisältää myös erityisen `<defs>`-elementin. Sen sisään voi asettaa muita SVG:n elementtejä ilman, että ne hahmontuvat näkyviin, mutta niihin voi silti viitata. Listauksessa 2.1 on `<defs>`-elementti rivillä 4. Siitä nähdäänkin, että `<defs>`-elementin sisällä määritellään muun muassa grafiikassa tarvittavat väriliu'ut (`<linearGradient>` tai `<radialGradient>`), suotimet (`<filter>`) ja täyttökuviot (`<pattern>`).

Graafisia objekteja voi käyttää muiden objektien muotoilemiseen. Tähän on kaksi elementtiä: `<mask>` ja `<clipPath>`, jotka rajaavat kohteena olevasta objektista näkyvää osaa. `<clipPath>` eli leikkuupolku rajaa kohteesta pois kaiken, mikä on leikkuupolun ulkopuolella. Toisin sanoen tuloksena on kohteen ja leikkuupolun leikkaus. `<mask>` eli maski muistuttaa toiminnaltaan leikkuupolkua, mutta leikkuupolusta poiketen sen toiminta ei perustu polkujen reunoihin vaan väriarvoihin. Väriarvojen perusteella kuvaa hahmonnettaessa kohteen jokaisen pikselin peittävyys määräytyy maskin vastaavan pikselin väriarvojen mukaan tietyllä matemaattisella kaavalla. Maskin avulla voidaan esimerkiksi rajata kohdetta häivytetyillä reunoilla.

SVG:ssä on sisäinen koordinaattijärjestelmä, jonka yksikköstä käytetään nimitystä

käyttäjäyksikkö (*user unit*). Se on erillinen selaimen pikselikoordinaatistosta, mutta oletuksena nämä koordinaatistot vastaavat toisiaan. Tämä seuraa siitä, että sisäinen koordinaatisto määräytyy oletuksena `<svg>`:lle asetettujen `width`- ja `height`-attribuuttien mukaiseksi. Julkaisija voi kuitenkin halutessaan määrittää `<svg>`:n `viewBox`-attribuutilla SVG:n koordinaatistosta alueen, joka kuva-alaan piirtyy. Listauksessa 2.1 `viewBox` on määritetty rivillä 3. Siinä kuvassa näkyvä koorinaatiston osa alkaa origosta ja on 284 yksikköä leveä ja 142 yksikköä korkea. Kuvan fyysinen koko on asetettu heti rivillä 1 `<svg>`-elementin `width`- ja `height`-attribuuteilla.

SVG:n XML-pohjaisuudesta seuraa, että se sisältää paljon toisteisia merkkijonoja. Tämä merkitsee, että se pakkautuu hyvin ja häviöttömästi. W3C:n suositukseen kuuluu, että gzip-pakattu SVG-tiedosto varustetaan tiedostotunnisteella **.svgz**. Tällöin muun muassa selaimet osaavat käsitellä tiedostoa oikein edellyttäen, että palvelin lähettää tiedoston mukana asianmukaisen sisältötyyppiotsakkeen. Tämä saattaa vaatia vielä nykyäänkin ylimääräistä konfigurointia palvelinpuolella. Toisteinen visuaalinen sisältö on jo SVG:ssä itsessäänkin mahdollista ”pakata” `<use>`-, `<pattern>`- ja `<symbol>`-elementtien avulla.

SVG-tiedosto voi ottaa kantaa omaan rasterointiprosessiinsa. Tämä tapahtuu `<svg>`-juurielementin `shape-render`-attribuutilla, joka voi saada neljä eri arvoa: `auto`, `optimizeSpeed`, `crispEdges` ja `geometricPrecision`. Attribuutin avulla grafiikan julkaisija tai kehittäjä voi ehdottaa selaimelle, priorisoidaanko hahmonnusprosessissa jotakin tiettyä ominaisuutta, kuten hahmonnusnopeutta tai -laatua.

2.5 SVG verkkosivuteknologiana

SVG on ollut olemassa jo vuodesta 2001 lähtien, ja vaikka jotkin selaimet ovat tukeneet sen esittämistä jo vuodesta 2008 (Firefox, versio 2), se on tulossa osaksi standardien verkkosivukehitysteknologioiden työkalupakkia vasta HTML5:n myötä. Käytännössä tuki on alunperin ollut SVG:n käyttämiseen staattisen kuvasisällön esittämiseksi `<img>`, `<embed>` ja `<object>` elementtien avulla [7], mikä riittää ratkaisemaan rasterigrafiikan ongelman vaihtelevien näyttökokojen kanssa. Esimerkki tällaisesta SVG-kuvan käytöstä on esitetty merkintäkoodina listauksessa 2.2.

```
<div>
  <img src="sampleimage.svg"
       alt="An SVG image linked with an img tag" />
</div>
```

Listaus 2.2 *SVG-kuvan asettaminen sivulle `<img>`-elementillä.*

Skaalautuvuus ei kuitenkaan ole ainoa etu, jonka SVG tuo verkkosivukehitykseen. HTML5:n myötä SVG-merkintäkoodia on nykyään mahdollista liittää suoraan HTML-merkinnän mukaan ilman `<img>`-elementtiä [6]. Tästä esimerkkinä on seuraava merkintäkoodi:

```
<div><!-- HTML here! -->
  <svg width="200px" height="100px">
    <!-- Suddenly, wild SVG appears! -->
    <desc>An SVG added inline in an HTML DOM.</desc>
    <rect x="10" y="20" width="50" height="30" />
  </svg>
</div>
```

Näin SVG grafiikka tulee osaksi dokumentin DOM-rakennetta (*Document Object Model*) eli sen elementtien välistä hierarkiaa. Siten myös SVG:tä voidaan manipuloida JavaScriptillä ja tyylittää CSS:llä aivan siinä missä HTML:ääkin [6]. Tämä seikka mahdollistaa dynaamisen grafiikan toteuttamisen SVG:llä, eli SVG on vaihtoehto myös HTML5:n piirtoalustaelementille.

2.5.1 CSS ja SVG

HTML:n kehityksen myötä merkittävä osa sen esitystapaan vaikuttavista elementeistä on todettu vanhentuneiksi, koska CSS tarjoaa monipuolisemman ja käytännöllisemmän tavan niiden toteuttamiseen [28]. Vanhentuneita elementtejä ovat esimerkiksi `<font>` ja `<center>`. HTML-dokumentin ulkoasun asettaminen CSS:n avulla toteuttaa vastuiden erottelumallin, mikä parantaa sisällön ylläpidettävyyttä. Tämä mahdollistaa muun muassa dokumentin paremman siirrettävyyden, koska itse informaatiosisältö on erillään ulkoasumäärittelyistä. Lisäksi ulkoasun muokkaaminen on CSS:n avulla mahdollista ilman, että alkuperäiseen HTML-dokumenttiin tarvitsee tehdä muutoksia.

SVG on tyylitettävissä CSS:llä, joten nämä edut koskevat myös sitä. SVG:n avulla voidaan esittää visuaalistakin informaatiota – kuten kaavioita ja diagrammeja – mahdollistaen sen ulkoasun määrittämisen erillisten tyylitiedostojen avulla. Jos esimerkiksi kaavioita sisältävä verkkopalvelu on muutettava noudattamaan uutta graafista ohjeistoa, myös kaavioiden ulkoasun muutos onnistuu pelkkiä tyylitiedostoja muokkaamalla. Tämä ei ole mahdollista rasterigrafiikkapohjaisissa ratkaisuissa. CSS3:n mediakyselyiden, siirtymien ja animaatioiden ansiosta SVG:tä voidaan hyödyntää myös erilaisten käyttöliittymäelementtien toteuttamiseen pelkkää HTML:ää paremmin.

2.5.2 SMIL ja SVG

Animoitu grafiikka verkkosivuilla on perinteisesti tehty animoitujen GIF-tiedostojen avulla. GIF-formaatille on laaja selaintuki, mutta sen haittapuolena on 256 kappaleeseen rajoittuva väripaletti ja kaupallisuus [1], [29]. Paremman värisyvyyden tarjoavasta PNG:stä on jo kehitetty myös animaatioita tukeva formaatti, APNG [30]. SVG:n manipuloitavuus CSS:llä ja JavaScriptillä mahdollistaa animaatioiden toteuttamisen myös SVG:llä, joten myös animoituun sisältöön on tarjolla vektoripohjainen vaihtoehto. SVG:n animoimiseen on saatavilla myös kolmas kieli: SMIL eli *Synchronized Multimedia Integration Language* [31]. SMIL on SVG:n tavoin W3C:n julkaisema XML-pohjainen standardi. SMIL-merkintää voi yhdistää SVG-merkintään, jolloin dokumentin sisällä voidaan kuvata monipuolisia animaatioita monimutkaisinkin ajastuksin deklaratiivisesti.

Esimerkkinä SMIL:in käytöstä on listaus 2.3, jossa rivit 1 ja 11 muodostavat SVG:n `<circle>`-elementin. Tuon ympyrän sisälle on asetettu SMIL:in `<animate>`-elementti, joka määrittelee ympyrälle animaation. Animaatiossa ympyrän säde kasvaa arvoon 50 ja palautuu sitten takaisin alkuperäiseen arvoon 37,5 (asetettu rivillä 5). Rivillä 9 on asetettu animaation ajoitusattribuutit `begin` ja `dur`. Tässä tapauksessa animaatio alkaa, kun jokin toinen animaatio tunnisteella `exhale` on päättynyt ja kestää 2 sekuntia. [31]

```
1 <circle r="37.5">
    <animate id="inhale"
3     attributeName="r"
      attributeType="XML"
5     values="37.5; 50; 37.5"
      calcMode="spline"
7     keyTimes="0;0.5;1"
      keySplines=".5 0 .5 1; .5 0 .5 1 "
9     begin="exhale.end" dur="2s"
      fill="freeze" repeatCount="indefinite" />
11 </circle>
```

Listaus 2.3 *Esimerkki SMIL:illä animoidusta SVG-elementistä, jossa SVG:n `<circle>`-elementti on asetettu muuttamaan sädettään edestakaisin.*

SMIL tukee ajastusattribuuteissaan jopa käyttäjän suorittamia tapahtumia, joten animaation voi määritellä käynnistymään vaikkapa jotain tiettyä elementtiä klikattaessa. Siinä, missä JavaScript edellyttää HTML-dokumenttiin suoraan upotettua SVG-merkintää, SMILin avulla animoitu SVG toimii animoidusti myös silloin, kun sitä käytetään normaalin kuvan tavoin `<img>`-elementillä. [31]

2.6 Aiempi kirjallisuus SVG:stä verkkosivuteknologiana

Huolimatta ilmestyessään saamastaan positiivisesta huomiosta ja odotuksista [20], [32] SVG:tä on tällä vuosikymmenellä käsitelty verkkosivunäkökulmasta kirjallisuudessa ja tieteellisissä julkaisuissa varsin vähän. Sitä sivutaan lääketieteen tai kartografian sovellusten toteutusteknologiana toistuvasti [10], [33]–[36], mutta ajantasaista ja sovellusalariippumatonta materiaalia SVG:stä ei juuri ole. Myös verkkosivukehitystä käsittelevässä kirjallisuudessa SVG usein korkeintaan mainitaan [7], [8], [29], jolloin sen mahdollisuudet pysyvät tuntemattomina.

Vaikka tuki SVG:lle löytyy nykyään kaikista tärkeimmistä selaimista [5], kehittäjien keskuudessa tuo kiinnostus on jäänyt maltilliseksi: SVG:stä ja sen periaatteista ollaan tietoisia, mutta sen varsinaisia hyötyjä ei välttämättä tiedosteta. Siksi kehittäjät – jopa suunnittelijat – usein valitsevat ennemmin tutun rasterigrafiikan ja ratkaisevat päätelaitekirjon ongelman muilla tavoin.

Eräässä vuoden 2015 diplomityössä toteutettiin lääketieteellistä dataa kuvaajin visualisoiva verkkosovellus. Työssä valittiin piirtoa varten rasterigrafiikkapohjainen `<canvas>`-ratkaisu SVG:n sijaan pelkästä olettamuksesta, että SVG olisi suorituskyvyltään heikompi [10]. Julkaisussa *Compression and Optimization of Web-Contents* puolestaan todetaan, että SVG olisi liitännäispohjainen, laitteisto- tai käyttöjärjestelmäsidonnainen ja ennemmin kiertokeino kuin varsinainen kuvaformaatti animaatiokontekstissa [12].

Myös Tero Taipaleen responsiivista verkkosivukehitystä käsittelevä diplomityö mainitsee SVG:n ratkaisuna vaihteleviin näyttötarkkuuksiin, mutta listaa teknologian haittapuoliksi suuren tiedostokoon, grafiikan toteutuksen työläyden ja ohjelmistojen heikon saatavuuden [11]. Hän myös vihjaa, että SVG:ksi konvertointi olisi työläs, mutta välttämätön prosessi. Tosiasiassa valtaosa verkkosivustokäyttöön luotavasta grafiikasta luodaan tai voitaisiin luoda alkeismuotojen ja polkujen avulla, joten sen voisi viedä suoraan SVG:ksi. Tällöin rasterisointia ei tapahdu missään kohtaa työstöprosessia ellei sitä katsota tarpeelliseksi jostain erityisestä syystä. Julkaisu vuodelta 2012 puolestaan kertoo, että suoritinten kehittyessä on energiataloudellisempaa suosia suoritinkäyttöä sen sijaan, että ladattaisiin enemmän dataa langattomasti verkosta [37]. Julkaisu vuodelta 2016 toteaa, että graafinen materiaali on oleellinen osa verkkoselaamisen kaistankäytöstä ja erityisesti mobiilikäyttäjien kannalta kehittäjien on huomioitava ja mahdollisuuksien mukaan optimoida tarvittavan grafiikan kaistankäyttö [12]. Kyseinen julkaisu kuitenkin rajoittuu käsittelemään pelkästään rasterigrafiikkaformaatteja, joten SVG:n vertailu samoilla kriteereillä on tarpeen.

3. GRAFIIKKALAJIEN VERTAILUMENETELMÄT

Edellinen luku loi kattavan katsausksen grafiikkalajeihin ja niihin SVG:n ominaisuuksiin, jotka tekevät siitä potentiaalisesti käyttökelpoisen teknologian verkkosivukehityksen saralla. Luku eritteli myös sitä, miten SVG:tä on käsitelty kirjallisuudessa ja julkaisuissa viime vuosina. Kirjallisuudesta löytyneet väittämät muodostavat ryhmän näkökohtia, joiden pohjalta SVG:n käytännöllisyys verkkosivuteknologiana määräytyy. Väittämät olivat seuraavat:

1. SVG on huonosti tuettua web-käyttöä ajatellen [7].
2. SVG on tiedostokooltaan suurempaa, kuin rasteriformaatit [11].
3. SVG on `<canvas>`-elementtiä heikompi suorituskyvyltään [10].
4. SVG on kilpailijoitaan huonommin soveltuva animaatioiden tekemiseen [12].
5. SVG on liitännäispohjainen, laitteisto- tai käyttöjärjestelmäsidonnainen [12].
6. SVG on työlästä tuottaa [11].

Kaikki väittämät ottavat jollain tavalla kantaa tämän työn tutkimuskysymykseen: voiko SVG korvata rasterigrafiikan verkkosivukehityksessä. Jotta näin olisi, SVG:n pitäisi soveltua valokuvamateriaalin (JPG), digitaalisen piirrosgrafiikan (PNG), animaatioiden (GIF) sekä dynaamisen grafiikan (`<canvas>`) esittämiseen. Formaatin käytännöllisyys edellyttää ennen kaikkea riittävää tukea selaimilta sekä kohtuullista siirtokaistan käyttöä, jotta latausajat säilyvät maltillisina heikkotasoisemmillakin yhteyksillä. Lisäetua on vähäisestä toteutustyömäärästä sekä kohtuullisesta suorituskykyvaatimuksesta, jotta sisältö toistuu sujuvasti myös rajoittuneemmilla päätelaitteilla. Tässä luvussa kuvataan tutkimusmenetelmät, joilla selvitetään, miten hyvin SVG soveltuu rasteriformaattien käyttökohteisiin. Tutkimusmenetelmät kattavat yllä esitetyistä väitteistä väitteet 1-4. Väitteen 5 todenperäisyys selviää tutkimuksen ohella ja väitettä 6 käsittelee luku 5. Testausmentelminä ova staattisen grafiikan testikuvat, selaintuen testitaulukot ja dynaamisen grafiikan testisovellus.

3.1 Hahmontuminen ja tiedostokoko

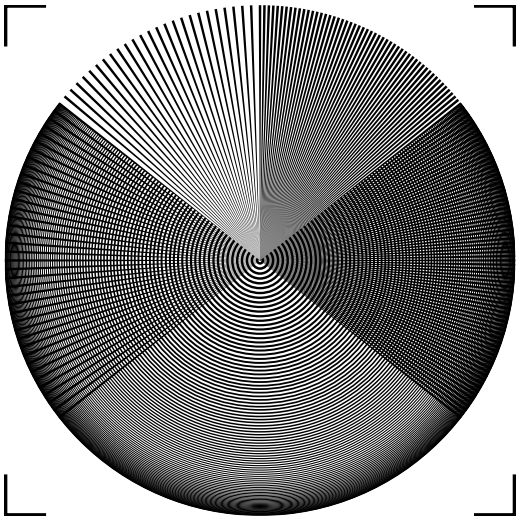
Ensimmäiseksi vertaillaan kuvaformaattien visuaalista laatua sekä tiedostokokoa. Tätä varten luodaan vertailtavat testikuvat, jotka vastaavat tyypillisiä käyttökohteita molemmille staattista rasterigrafiikkaa esittäville tiedostoformaateille. Testikuvina on siis kaksi piirrosgrafiikkakuvaa ja kaksi valokuvaa. Testiä varten luodaan verkkosivu, johon liitetään kustakin testikuvasta yhteensä 16 eri versiota. Versioiden välillä vaihdellaan tiedostoformaattia, kokoa ja laatua.

Testikuvasta tehdään neljä eri vektoripohjaista versiota: yksi käyttämään kutakin `shape-render`-attribuutin arvoa. Testikuvasta tehdään lisäksi 12 rasteriversiota: neljä PNG-tiedostoa ja kahdeksan JPG-tiedostoa. Molemmista tiedostoformaateista tehdään neljä erikokoista tiedostoa, joiden ulottuvuudet ovat seuraavat: 200 pikseliä, 400 pikseliä, 600 pikseliä ja 800 pikseliä. Testiasetelmassa jokainen koko kuitenkin asetetaan testisivulle skaalattuna 600 pikselin leveyteen CSS-säännöllä. Näin voidaan verrata sekä ylöspäin että alaspäin skaalattua rasterigrafiikkaa ja myös skaalaamatonta grafiikkaa. 200 pikselin levyisen kuvan skaalaaminen 600 pikselin levyiseksi vastaa kolminkertaista skaalauskerrointa, joka oli suurin taulukossa 2.1 esiintyvä skaalauskerroin.

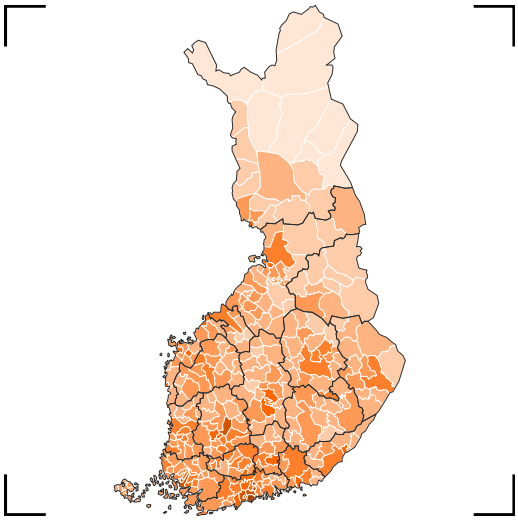
Koska PNG on häviöttömästi pakattu, pakkaustaso ei vaikuta sen ulkonäköön. JPG:n häviöllisessä pakkausalgoritmissa on sen sijaan useita lopputuloksen ulkonäköön vaikuttavia parametreja, joten testikuvasta tehdään edellä listatut kokoversiot kahdella eri pakkausasetuksella. Asetuksista käytetään tässä nimityksiä *korkea laatu* ja *matala laatu*. Nimet viittaavat rasterointiin käytettävän *Affinity Designer* -suunnitteluohjelmiston tiedostoviennin laatuasetukseen. Liitteessä A on esimerkkinä kuvankaappaus testisivusta, jossa tutkittavana on kuva 3.1(b).

Testikuvina käytetään Nicole Asunin ja Andrea Giachettin TESTIMAGES-testikuvaarkiston [38], [39] materiaalia. Materiaali on rasterigrafiikkaa, joten testikuvista on tuotettava vertailtavaksi vektoripohjaiset versiot. Ensimmäinen testikuva 3.1(a) on nimeltään 'Ympyrä', ja se on toteutettu SVG:nä mukaillen TESTIMAGES-arkiston materiaalia. Kuvan yksityiskohtaisuus lähestyy ääretöntä, kun sen kiilakuvio etenee kohti keskustaa. Kuvio sopii siksi rasteroinnin näytteistyksen laadun testaamiseen. Kuvan XML-merkintä on esitetty liitteessä B.

'Ympyrä' on hyvin epätyypillinen kuva ja poikkeaa oleellisesti kuvamateriaalista, jota tavallisesti käytetään verkkosivuilla. Siksi toinen kuva poimitaan testiin verkkosivukäytöstä. Kyseinen testikuva 3.1(b) on kaavio, joka esittää Suomen väestökehitystä kunnittain. Kuvasta käytetään nimitystä 'Suomi'. Vertailua varten kuvan XML-rakenne on optimoitu poistamalla siitä tarpeettomat elementit ja attribuutit.



(a) Testikuva: Ympyrä.



(b) Testikuva: Suomi (mukailtu lähteestä [40]).

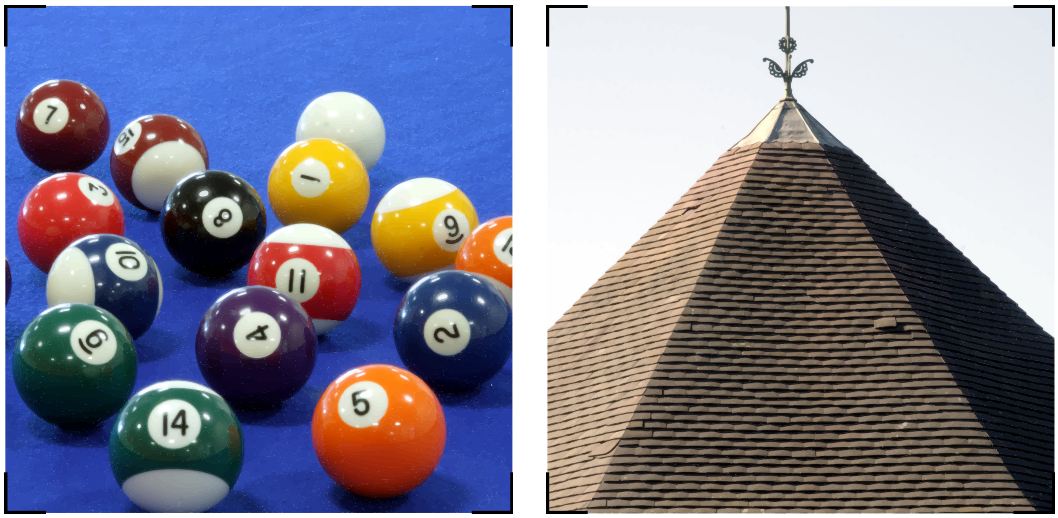
(c) Testikuva: Biljardi.

(d) Testikuva: Torni.

Kuva 3.1 *Hahmonnuksen testaamiseen käytettävät testikuvat.*

Lopuksi TESTIMAGES-arkistosta poimitaan kaksi eri tyyppistä valokuvaa. Testivalokuva 3.1(c) eli 'Biljardi' on sisällöltään värikäs ja sisältää laajoja sävyiltään tasaisesti muuttuvia alueita. Sillä voidaan testata värisyvyyden toistumista, joka saattaa olla hankalaa toisintaa vektorigrafiikan keinoin. Testivalokuva 3.1(d), 'Torni', on väriavaruudeltaan suppea, mutta sisältää paljon toistuvia ja teräviä yksityiskohtia. Vektorointialgoritmi saattaa tuottaa 'Biljardi'-kuvaa paremman lopputuloksen, koska vektoroinnin puutteet eivät välttämättä erotu selkeästi yksityiskohtien seasta.

Valokuvat voisi esittää SVG:nä yksinkertaisesti upottamalla ne `<image>`-elementillä, mutta todelliseksi vektorigrafiikaksi muuntaminen edellyttää *vektorointialgoritmin*



(a) Vektoroitu testikuva: Biljardi. (b) Vektoroitu testikuva: Torni.

Kuva 3.2 *Adobe Illustratorin avulla vektoroidut versiot rasteripohjaisista hahmonnuksen testikuvista.*

käyttöä. Valokuvat muunnetaan tätä tutkimusta varten vektorigrafiikaksi käyttämällä Adobe Illustrator -ohjelmiston vektorointitoimintoa. Testattavien valokuvien vektoroidut versiot ovat esitettynä kuvassa 3.2.

Testikuvien hahmontumisen laatua arvioidaan ensisijaisesti silmämääräisesti, koska verkkosivukontekstissakin grafiikaa käytetään vain ihmissilmää varten. Tarvittaessa esimerkiksi tulosteiden eroavuus voidaan osoittaa matemaattisesti generoitujen erotuskuvien avulla. Testissä vertailukohteena toimii 600 pikselin levyinen PNG-versio, koska se on häviötön ja valmiiksi skalaattu esityskokoon kuvankäsittelyohjelmalla. Siten selaimen ei tarvitse käsitellä sen sisältöä ollenkaan. Näin voidaan verrata selainten rasterointi- ja interpolaatioalgoritmien laatua kuvankäsittelyohjelman vastaaviin.

Tiedostokoon vertailemiseksi tutkittavista testikuvista versioineen taulukoidaan tiedostokoot. Materiaalia muokataan kuitenkin siltä osin, että vektoripohjaisista kuvista tehdään vain kaksi eri versiota: SVG ja gzip-pakattu SVGZ. Tiedostokoon kannalta kuva 3.1(a) on mielenkiintoinen, sillä se rakentuu osittain rekursiivisesta kloonaamisesta. Siinä esiintyvät kiilat ovat identtisiä, joten riittää, että vain yksi `<polygon>`-elementti määritellään ja tyylitetään. Tämä yksi voidaan seuraavaksi kloonata `<use>`-elementillä ja tulos ryhmittää `<g>`-elementillä. Syntyvä ryhmä voidaan edelleen kloonata `<use>`-elementillä ja jälleen ryhmitellä, kunnes tulos sisältää tarpeeksi monta kloonia. Rakenne käy hyvin ilmi liitteestä B, jossa kyseisen testikuvan merkintäkoodi on esitetty. Kloonausta ei voi soveltaa kuvan 3.1(a) renkaisiin,

koska niiden tyylimääritteet vaihtelevat.

3.2 Selaintuki

Verkkosivuteknologian tarkoituksenmukaisuuden ja maturiteetin tärkein mittari on selaintuki. Tässä osiossa tutkitaan syvällisemmin eri selainten tukea SVG:n käytölle. Tukea kartoitetaan lähtien selaimien kyvystä hahmontaa SVG-kuvia liitettynä HTML-dokumenttiin eri tavoilla: `<img>`-, `<object>`-, `<embed>`- ja `<iframe>`-elementeillä, suoraan `<svg>`-elementillä HTML-merkinnän sekaan upotettua SVG-merkintäkoodia sekä CSS:llä elementin taustakuvaksi asetettua SVG:tä. Sen jälkeen selvitetään tukea SVG:n eri ominaisuuksille, kuten suotimet, syväys ja maskit. Myös tuki CSS- ja JavaScript-yhteiskäytölle sekä SMIL:lle selvitetään. Vertailun vuoksi testataan samalla tuki kahdelle tuoreemmalle SVG:n kilpailijalle: `<canvas>`-elementille ja APNG -tiedostoformaatille.

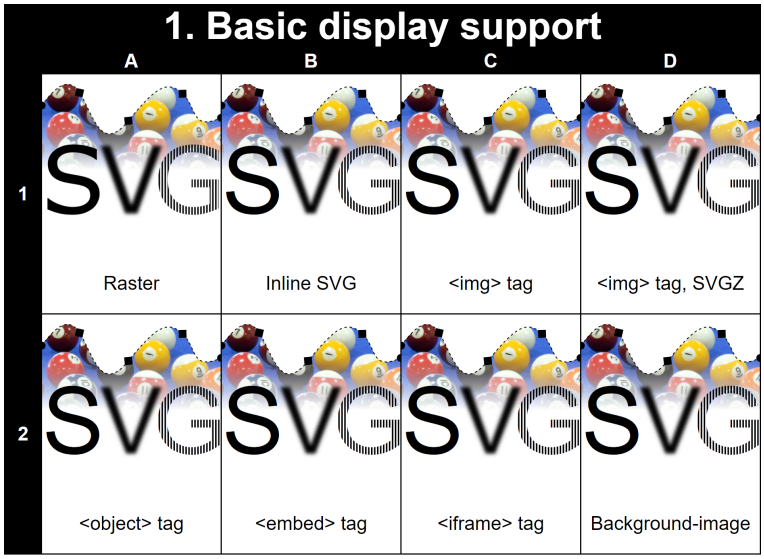
Testattavat selaimet versioineen ovat Chrome 62, Firefox 56, Safari 11, Edge 15 ja Internet Explorer 11. Chromesta ja Safarista käytetään sekä työpöytä- että mobiiliversioita kehittäjiensä omilla käyttöjärjestelmillä. Myös yhteensopivuus aiempien Internet Explorer -versioiden kanssa selvitetään hyödyntäen IE11:n emulointitoimintoa, jolla voidaan emuloida verkkosivun hahmonnusta IE:n versioilla 10-7 sekä versiolla 5.

Verkkokehitystekniikoiden selaintuen selvittämiseen käytetään yleensä palvelua *Can I Use?* [41]. Valitettavasti se esittää teknologioiden tuen vain yleisellä tasolla: palvelu esimerkiksi kertoo kyllä, että Chrome tukee sekä SVG:n että CSS:n suotimia HTML-elementeille [42], mutta palvelusta ei löydy tietoa siitä, tukeeko Chrome CSS:n suotimia SVG:n elementeille. Siksi tätä työtä varten luodaan oma menetelmä selaintuen testaamiseksi.

SVG:n perusominaisuuksien testaamiseksi luodaan sarja taulukoita, jonka jokainen solu sisältää yhden testattavan SVG:n ominaisuuden. Näin voidaan kerralla nähdä, kuinka hyvin testattava selain noudattaa standardin mukaista toimintaa kyseisen ominaisuuden kohdalla. Soluihin viittaamiseksi sarakkeet on nimetty aakkosten mukaan A:sta G:hen ja rivit numeroitu 1:stä 9:aan. Taulukoiden sisältö on esitetty yksityiskohtaisesti liitteessä C, mutta seuraavaksi sisällöt listataan yleistasoisemmin.

Ensimmäinen taulukko (kuva 3.3) sisältää 7 eri tapaa sisällyttää SVG:tä verkkosivuun. Taulukon ensimmäinen solu sisältää rasteroidun version siitä, miltä tuloksen pitäisi näyttää. Loput solut ovat järjestyksessä rivinsisäisesti upotettu SVG-merkintä, tavalliseen `<img>`-elementtiin asetettu SVG-tiedosto, sama pakatulla SVGZ-

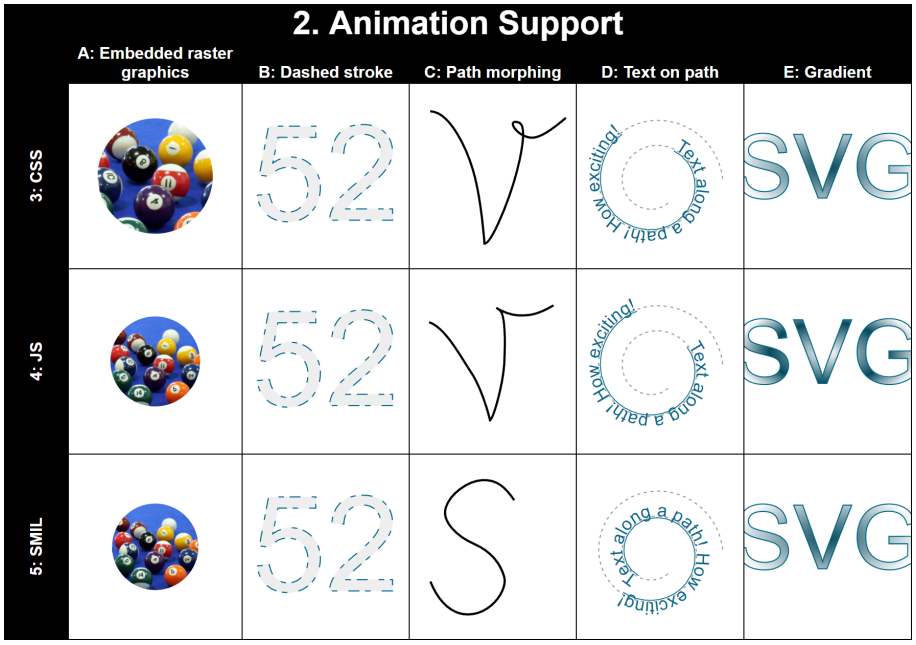
tiedostolla, `<object>`-elementti, `<embed>`-elementti, `<iframe>`-elementti ja lopuksi SVG-kuva asetettuna elementin taustakuvaksi CSS:n `background-image`-säännöllä. Testattavana oleva kuva käyttää useita SVG:n eri ominaisuuksia, jotta samalla voitaisiin varmistua testattavan selaimen kyvystä hahmontaa monimutkaisempaakin SVG:tä oikein. Kuvassa on käytetty katkoviivaa (`dash array`), reunamerkkejä (`<marker>`), leikkauspolkua (`<clipPath>`), peittoa (`<mask>`), upotettua rasterigrafiikkaa (`<image>`), tekstiä (`<text>`), suotimia (`<filter>`) ja kuviotäyttöä (`<pattern>`).



Kuva 3.3 *Selaintuen testitaulukko 1: SVG:n upottaminen sivulle. Taulukon ensimmäinen solu sisältää PNG-version kuvasta ja muihin soluihin on upotettu SVG-versio eri upotustavoilla.*

Toinen taulukko (kuva 3.4) sisältää eri animaatioita toteutettuna CSS:llä, JavaScriptillä sekä SMIL:llä. Animaatiot on sijoiteltu viiteen sarakkeeseen ja kullakin rivillä on käytössä yksi animointimenetelmä. Ensimmäinen testianimaatio on SVG:hen `<image>`-elementillä upotettu rasterikuva biljardipalloista, jotka esiintyivät jo testikuvassa 3.1(c). Kuvaa rajataan `<clipPath>`-elementtiä käyttäen ympyrällä, joka muuttaa kokoaan edestakaisin. Myös itse kuvan koko vaihtelee, mutta vaihteluväli on rajaavaa ympyrää suurempi. Toinen testianimaatio on katkoviiva, joka kiertää vaihtuvaa tekstisisältöä niin kutsuttuja marssivia muurahaisia muistuttavalla tavalla. Tässä testataan, vaikuttaako vaihtuva sisältö animaation toimivuuteen. Kolmas testattava animaatio on polku, joka muuttaa muotoaan muodostaen järjestyksessä kirjaimet S, V ja G toistuvasti. Neljäs testianimaatio sisältää spiraalin muotoiselle polulle asetetun tekstin, joka liukuu polulla edestakaisin. Viimeinen testianimaatio sisältää täyttötyylinä (*fill*) olevan väriliu'un, jonka keskiankkuri liikkuu edestakaisin. Tämä saa liu'un ikäänkuin aaltoilemaan.

Kolmannessa taulukossa (kuva 3.5) testataan sekalaisia toimintoja. Animoitujen



Kuva 3.4 *Tilannekuva selaintuen testitaulukosta 2: Animointimenetelmät. Todellisuudessa kuvat liikkuvat. Taulukko sisältää viisi eri animaatiota toteutettuna kolmella eri animointimenetelmällä.*

solujen toteutustekniikka on merkittynä solun oikeassa yläkulmassa. Tässä taulukossa SVG:n manipulointi JavaScriptillä tehdään hyödyntäen Adoben Snap.svg-kirjastoa [43], joka on kehitetty tarjoamaan jQueryn kaltainen rajapinta SVG:n manipuloimiseen. Taulukolla pyritään luomaan yleistasoinen läpileikkaus testattavan selaimen tuesta SVG:n ominaisuuksille. Taulukon toimiva versio löytyy osoitteesta http://jjsp.fi/di/compatibility/.

Taulukon solut A6-E6 sisältävät animointia, suotimia, peittoa sekä leikkausta erikseen ja yhdistellen. Niissä pohjana toimii JavaScriptin avulla dynaamisesti sisällöltään vaihtuva `<text>`-elementti. Seuraavat viisi solua, F6, G6 sekä A7-C7 sisältävät elementtejä, joille on asetettu koko CSS:n avulla eri tavoilla. Soluissa varioidaan myös `viewBox`-attribuuttia, jonka tulee vaikuttaa sisällön hahmontumiskokoon. Solut D7 ja E7 sisältävät HTML:n kanssa yhteisten attribuuttien animointia CSS:llä.

F7 sisältää APNG-tiedostona tallennetun animaation ja G7 puolestaan yksinkertaisen `<canvas>`-testin. Näin voidaan testata selaintukea samalla myös SVG:n kanssa kilpaileville tekniikoille. Solut A8-D8 sisältävät animoituja polkuja, koska polkukaaren muodon muuttumisen animoiminen on huomattavasti muita testattuja attribuutteja mutkikaampi toteuttaa. Muotoa muuttavaa polkua käytetään leikkuupolkuna ja peittona soluja C6 ja D6 vastaavalla tavalla.

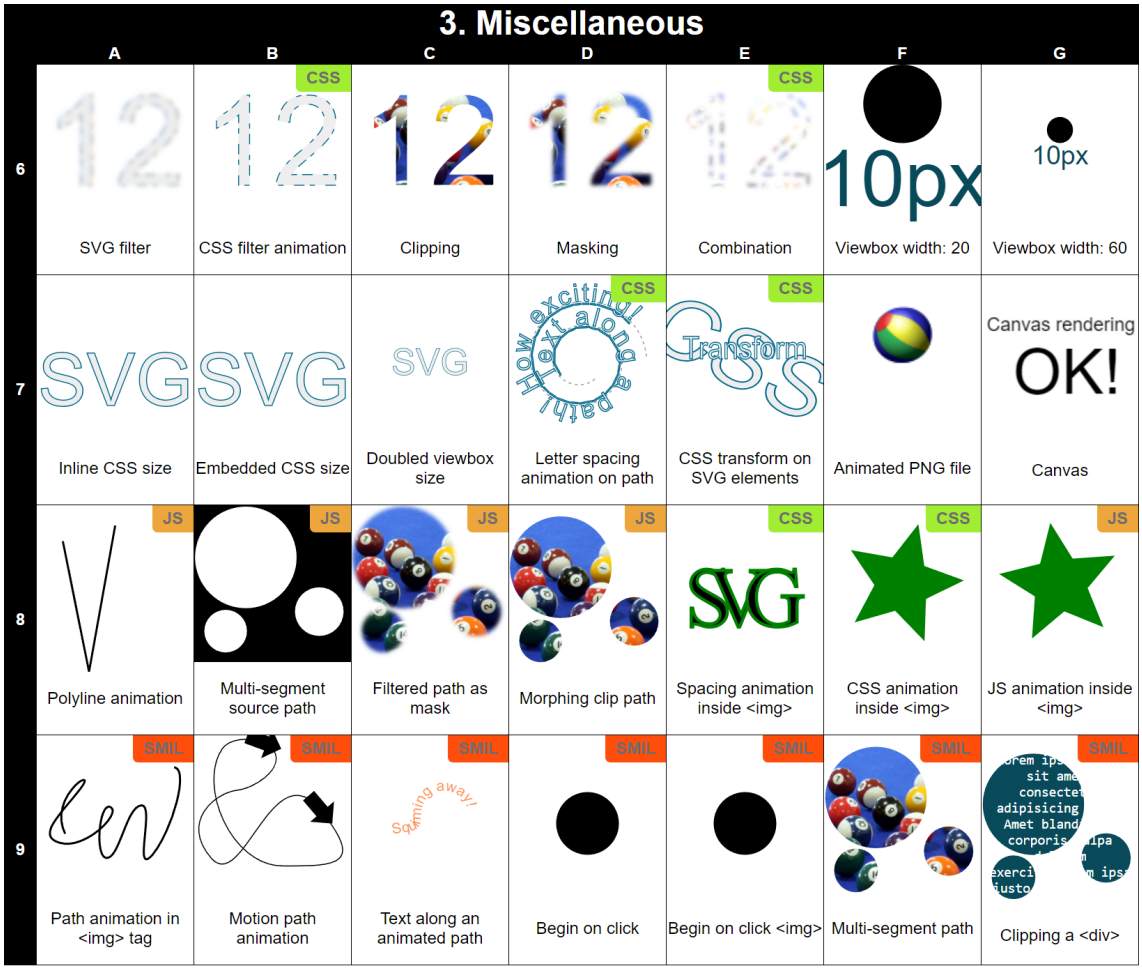
Solut E8-G8, A9-C9 sekä E9 sisältävät `<img>`-elementin avulla liitettyä, animoitua SVG:tä. Eri soluissa on käytetty eri animointitapaa, jolloin voidaan kartoittaa selainten tukea SVG:n animointiin ilman, että SVG:n DOM liitetään HTML:n DOM:in osaksi. Solu G8 on animoitu JavaScriptillä, ja koska JavaScriptin ajaminen näin olisi tietoturvariski, animaation ei kuulu toimia tässä. D9 ja E9 sisältävät SMIL-animaation, joka on asetettu alkamaan käyttäjän klikkauksesta. Jälkimmäisessä solussa SVG on liitetty `<img>`-elementillä, ja näin voidaan selvittää, välittyykö käyttäjän tapahtuma myös linkitettyyn kuvaan. Lopuissa soluissa F9 ja G9 käytetään solussa B8 käytettyä polkua, mutta tässä se on animoitu muuttamaan muotoaan SMIL:in avulla. Solussa G9 tuota polkua käytetään leikkaamaan normaalia HTML:n `<div>`-elementtiä.

Taulukot avataan testattavilla selaimilla, ja jokaisen solun kohdalla arvioidaan asianosaisen SVG:n ominaisuuden toimimista kyseisellä selaimella. Sitä, kuinka hyvin selain tukee kutakin ominaisuutta, arvioidaan kolmiportaisella jaottelulla: täysi, puutteellinen tai puuttuva tuki. Näiden tasojen esittämiseksi käytetään värikoodausta, jossa vihreä vastaa täyttä tukea, keltainen puutteellista ja punainen puuttuvaa tukea.

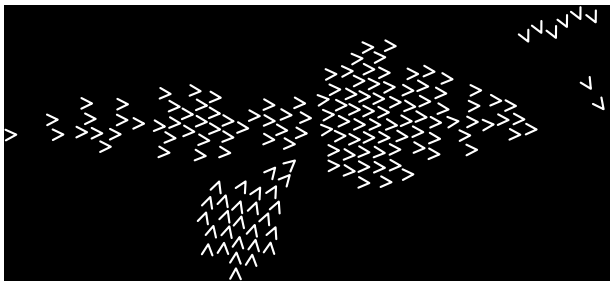
3.3 Dynaaminen grafiikka

Kuten tämän luvun alussa todettiin, aiheesta aiemmin tehty tutkimus osoittaa `<canvas>`-elementtiä pidettävän SVG:tä varmempana ratkaisuna dynaamisen grafiikan generointiin. Tässä osiossa vertaillaan sitä JavaScriptillä manipuloituun SVG:hen. Mentelmien vertailemista varten kehittiin sovellus, joka tuottaa selaimen esitettäväksi aktiivista sisältöä. Se toteutettiin niin, että varsinainen hahmonnus suoritetaan erillisessä komponentissa. Tätä komponenttia varten on määritetty rajapinta, jota sovelluslogiikka hyödyntää ilman, että komponentin sisäinen toteutus vaikuttaa muun sovelluksen toimintaan. Näin voimme toteuttaa komponentista kaksi eri versiota: yksi, joka hyödyntää SVG:tä ja toinen, joka hyödyntää `<canvas>`-elementtiä. Sovellus toteutetaan TypeScript-kielellä, sillä se tukee eksplisiittistä modulaarisuutta ja soveltuu siksi komponenttien toteuttamiseen hyvin. Vertailussa huomioidaan hahmonnuskomponenttien koodin selkeys ja niiden suorituskykyä mitataan eri selainten suorituskykyprofilointitoiminnoilla. Vertailulla pyritään selvittämään, mitkä tekijät ovat merkitseviä teknologian valinnassa, kun kyse on dynaamisesta grafiikasta.

Dynaamisen grafiikan tutkimiseksi toteutettava sovellus on versio Craig Reynoldsin Boids-algoritmista [44]. Algoritmi on valittu siksi, että sillä saa kohtalaisen helposti generoitua mutkikasta hahmonnettavaa selaimelle (kuva 3.6). Algoritmin ohjaamat

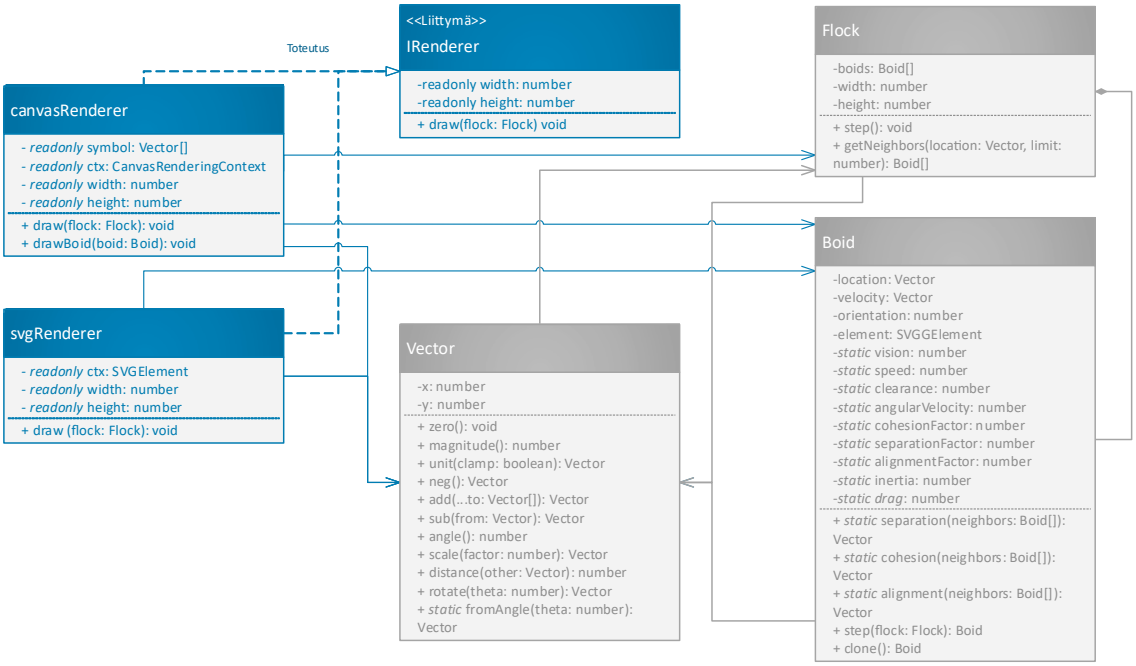


Kuva 3.5 *Selaintuen testitaulukko 3: Sekalaisia SVG:n ominaisuuksia. Animoiduissa soluissa animointimenetelmä on merkitty solun oikeaan yläkulmaan. Taulukolla voidaan nopeasti selvittää, kuinka paljon hajontaa SVG:n tuessa on eri selainten välillä.*



Kuva 3.6 *Boids-algoritmisovelluksen kuvankaappaus. Kuvassa valkoisella näkyvät boidit pyrkivät liikkumaan suoraan, mutta asettuvat parvimuodostelmaan päätyessään toistensa lähelle.*

objektit eli *boidit* liikkuvat, pyörivät ja niiden määrää säätämällä näkymän kompleksisuutta voidaan varioida. Algoritmin yhden askeleen asymptoottinen suoritusaika on boidien lukumäärään nähden $O(n^2)$, mutta sovelluksen komponenttirakenteen



Kuva 3.7 *Boids-algoritmisovelluksen luokkakaavio. Korostettuna hahmonnustoiminnallisuus eli hahmonnuskomponentin rajapinta sekä sen toteuttavat luokat.*

vuoksi voimme tutkia kohdistetusti hahmonnuskomponentin toimintaa. Vertailtavuutta edesauttaa myös se, että hahmonnuskomponentit kytkeytyvät muuten keskenään identtiseen toteutukseen kuten nähdään kuvasta 3.7. Hahmonnuskomponenttien eri versioiden lähdekoodit on listattu liitteessä D ja toimiva versio löytyy osoitteesta http://jjsp.fi/di/boids/.

Hahmonnuskomponentin rajapintakuvaukseen kuuluu kaksi metodia: `draw()` ja `constructor()`. Jälkimmäinen luo ja lisää DOM-hierarkiaan tarvittavat elementit sekä alustaa ne hahmonnusta varten. Ensimmäinen piirtää parven senhetkisen tilan asianmukaisella hahmonnusmenetelmällä. SVG-hahmonnuksen alustaminen edellyttää `<svg>`-elementin lisäämistä DOM-hierarkiaan. Elementtiin on myös lisättävä yksi boidi valmiiksi `<polyline>`-elementillä. SVG:n `<use>`-elementtiä hyödyntäen tämä esimääritelty boidi voidaan monistaa haluttuja muunnoksia soveltaen minne tahansa kuvan alueelle. Tämä hahmonnuskomponentti kykenee alustuksen jälkeen yksinkertaisesti siirtämään ja kiertämään objektit tarvittaviin sijainteihin ja asentoihin. `<canvas>`-elementtiin perustuvan komponentin sitä vastoin tarvitsee alustamistoimenpiteenä vain lisätä `<canvas>`-elementti DOM-hierarkiaan ja säilöä viite luotuun elementtiin. Algoritmin askelten hahmonnus tapahtuu pyyhkimällä joka askeleella koko piirtoalue ja piirtämällä sen jälkeen jokainen boidi uudelleen.

Sovelluksen suorituskykyä arvioidaan Chromen ja Firefoxin kehittäjätyökalujen avul-

la. Mittarina käytetään keskimääräistä ruudunpäivitysnopeutta 10 sekunnin ajanjaksolla. Mittauksen yksikkönä on FPS eli 'ruutua sekunnissa' (*Frames Per Second*). Mittaus suoritetaan molemmilla selaimilla käyttäen molempia hahmonnuskomponentteja ja vaihdellen simuloitavien boidien määrää 50 ja 300 kappaleen välillä 50 kappaleen askelissa. Testi suoritetaan kahdella erilaisella tietokoneella, joista toinen on suorituskyvyltään rajallinen kannettava tietokone ja toinen on pelikäyttöön tarkoitettu, nopealla näytönohjaimella varustettu pöytäkone. Tarkemmat tiedot testilaitteista on esitetty taulukossa 3.1.

Taulukko 3.1 *Dynaamisen grafiikan testilaitteiden tekniset tiedot.*

Ominaisuus	**Laite 1**	**Laite 2**
Suoritin	Intel Core i7-2600K	Intel Core i7-5600U
Taajuus	3,4 GHz	2,6 GHz
Keskusmuisti	16 Gt	16 Gt
Grafiikkapiiri	NVidia GeForce GTX 970	Intel HD Graphics 5500
Näyttömuisti	4059 Mt	128 Mt
Jaettu muisti	8144 Mt	8039 Mt

4. VERTAILUJEN TULOKSET JA PÄÄTELMÄT

Tässä luvussa esitetään luvun 3 mukaisten testien tulokset taulukoiden ja kuvaajien avulla soveltuvilta osin. Tuloksista esitetään myös sanallinen koonti ja analyysi.

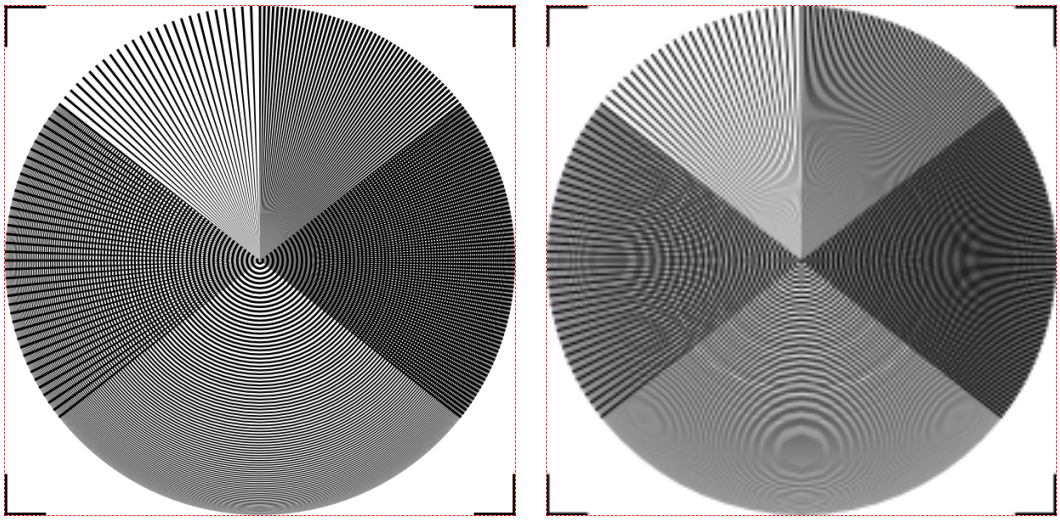
4.1 Hahmontuminen ja tiedostokoko

Tiedoston visuaalinen laatu ja tiedostokoko ovat grafiikkaformaatin tärkeimmät ominaisuudet ja usein toistensa kääntöpuolia: pienikokoinen tiedosto on tyypillisesti heikompi laadultaan ja hyvälaatuinen kuva vaatii paljon tilaa. Tässä aliluvussa verrataan molempia ristiin ja analysoidaan, pitääkö periaate paikkansa luvussa 3 esitetyille testikuville.

4.1.1 Testikuva 'Ympyrä'

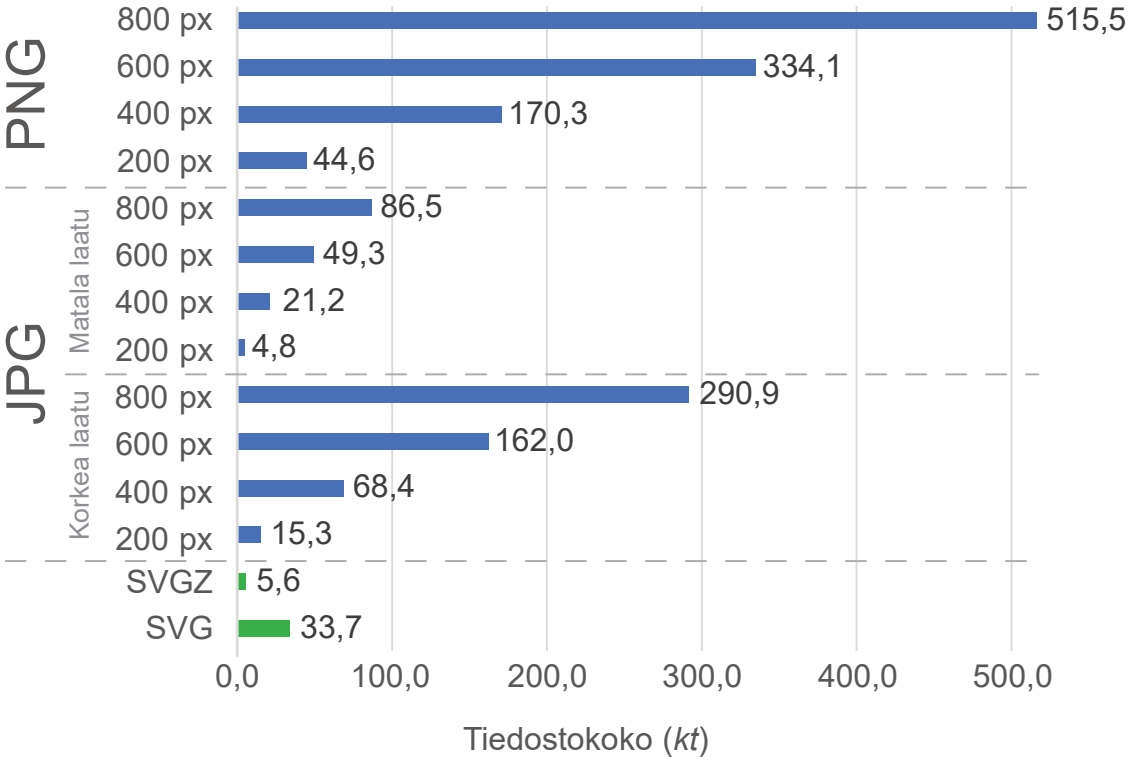
Testikuva 'Ympyrä' (kuva 3.1(a)) osoittautui erittäin tehokkaaksi testiksi selainten hahmonnuskyvylle. Tutkitaan ensin rasterigrafiikan skaalautumista. Kuvassa 4.1(a) on testikuva esitystarkkuudessa (eli 600 pikseliä), joten selaimen ei ole tarvinnut soveltaa siihen minkäänlaista interpolaatiota. Kuva 4.1(b) puolestaan on resoluutioltaan esitystarkkuudesta vain kolmannes. Siksi selain joutuu skaalaamaan sen kolminkertaiseksi, jotta kuva saataisiin fyysisesti oikean kokoiseksi. Skaalattu kuva näyttää sumealta ja pikselöityneeltä, vaikka skaalaus vastaa vain työpöytäselaimelle kehitetyn kuvan esittämistä älypuhelimen ruudulla. Kuvasta 4.2 käy ilmi, että 600 pikselin levyinen versio on tiedostokooltaan 334,1 kilotavua ja skaalausta vaatinut 200 pikseliä leveä versio on 44,6 kilotavun kokoinen tiedosto. Kokoero versioiden välillä on siis yli seitsenkertainen. Alkuperäinen SVG-versio sitä vastoin hahmontuu aina esitystarkkuudella ja tiedostokoko on vakio eri esitysresoluutioiden välillä. Tässä tapauksessa SVG-tiedosto vie PNG:tä vähemmän tilaa: 33,7 kilotavua ja pakattuna 5,6 kilotavua.

Jokaisessa selaimessa kuvaan ilmestyi elliptisiä hahmoja, joita kuva 4.3 esittää. Todellisuudessa kuvassa on vain samankeskeisiä ympyröitä (korostettu sinisellä). Samat



(a) Ei skaalausta (esitystarkkuus) (b) Skaalattuna kolminkertaiseksi

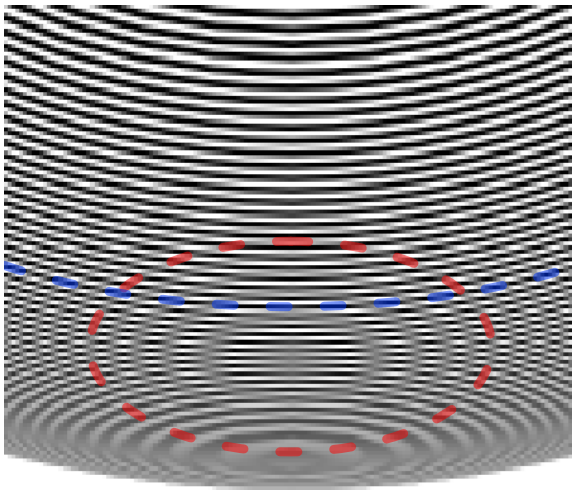
Kuva 4.1 *Skaalauksen vaikutus rasterikuvan laatuun.*



Kuva 4.2 *Testikuvan 'Ympyrä' tiedostokoot eri formaateissa. Suurikokoisimmat versiot ovat PNG-tiedostoja. Vastaavan resoluution JPG-versiot ovat oleellisesti pienempiä, mutta pakkausalgoritmi onkin häviöllinen. Tässä tapauksessa SVG- ja SVGZ-tiedostot ovat huomattavan pienikokoisia.*

kuviot ilmenevät tosin myös kuvankäsittelyohjelman tuottamassa PNG-versiossa, joten lienee kohtuutonta odottaa internetselainten toimivan tässä suhteessa paremmin.

Kyseessä on niin kutsuttu Moiré-kuvio, joka johtuu todennäköisesti rasterointialgoritmin aiheuttamasta laskostumisesta kun rasteroitava grafiikka on liian yksityiskohtaista. Rasteroinnin tuloksen pitäisi tässä tilanteessa edetä tasaväriseksi, kuten nähdään tapahtuvan kuvankäsittelyohjelman rasteroimassa kuvassa 4.4(a). Mikään testatuista selaimista ei toiminut näin.

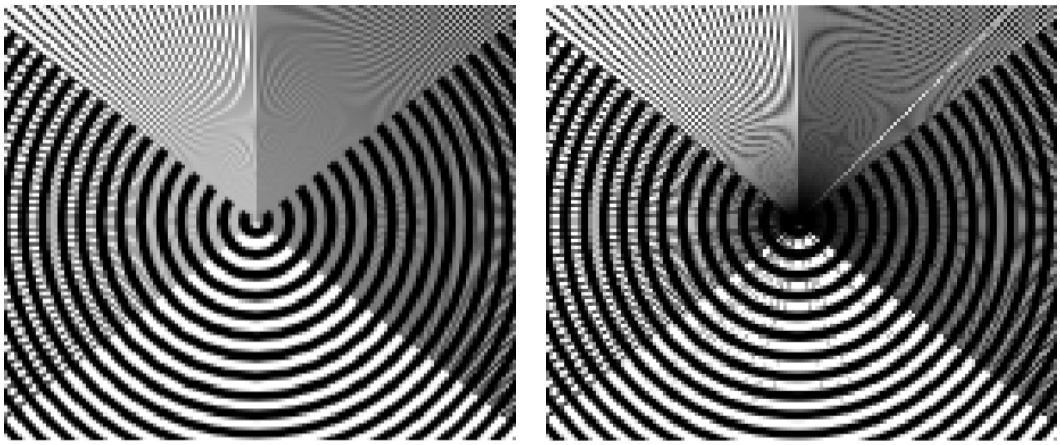


Kuva 4.3 *Testikuvan 'Ympyrä' rasteroinnissa ilmeneviä Moiré-kuvioita (korostettu punaisella). Todellisuudessa kuvassa on vain samankeskeisiä ympyröitä (korostettu sinisellä). Kuva saattaa olla tulostunut eri tavalla tässä dokumentissa, mutta alkuperäinen kuva on katsottavissa osoitteessa http://jjsp.fi/di/filesize/radial.*

Toinen havainto on, että Chromella kuvaan muodostuu virheellisiä artefakteja kuvan keskiosassa. Artefaktit näkyvät kuvassa 4.4(b), kun sitä verrataan kuvankäsittelyohjelman rasteroinnin tulokseen kuvassa 4.4(a). Chromen hahmonteessa keskipisteestä lähtee valkoinen viiru kohti oikeaa ylänurkkaa. Samoin keskustasta lähtee useita tummia juovia ympyrän alasektoria kohti, vaikka ainoa kuviointi tässä osassa pitäisi olla mustat ympyräkaaret.

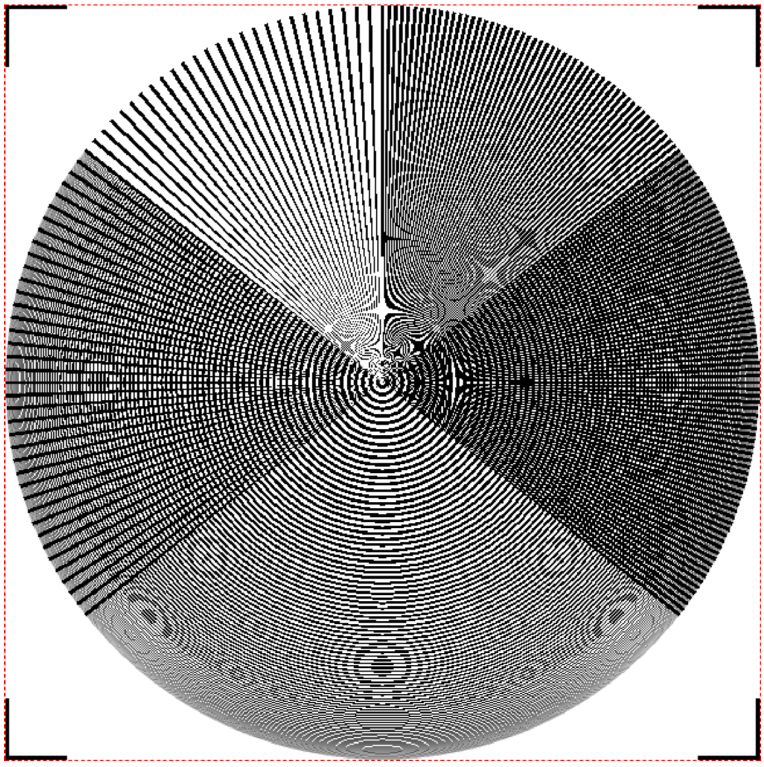
Kun testikuvan `shape-render`-attribuutti oli arvossa `optimizeSpeed`, Chromen hahmonnustulos oli kuvan 4.5 mukainen. Kuvassa havaitaan merkittävä visuaalisen laadun heikkeneminen. Pikseliruudukkoon nähden vinossa olevat sekä kaarevat viivat näyttävät kantikkailta ja Moiré-kuviota esiintyy lähes koko kuvan alueella. Kuvassa esiintyy pelkkiä täysin mustia ja täysin valkoisia pikseleitä, mikä viittaa siihen, että rasteroinnissa ei ole käytetty ollenkaan laskostumista hillitsevää ylinäytteistystä. Tulos oli vastaava myös `shape-rendering="crispEdges"`-arvolla ja Firefoxilla. Testatuista selaimista Edge on ainut, joka ei huomioi kyseistä asetusta, vaan hahmontaa kaikki neljä SVG-versiota identtisesti.

Muitakin selainten välisiä eroja löytyi. Firefox ja Edge hahmontavat hyvälaatuiset



(a) Alkuperäinen PNG-versio (b) SVG-versio Chromen rasteroimana

Kuva 4.4 *Suurennos testikuvan 'Ympyrä' keskiosasta ja Chromen hahmontama tulos. Chrome hahmontaa virheellisesti kuvan keskipisteestä ulospäin eteneviä viivamaisia artefakteja.*



Kuva 4.5 *Testikuva 'Ympyrä' rasteroituna nopeus etusijalla. Kuvassa esiintyy voimakasta Moiré-ilmiötä, koska rasteroinnissa ei käytetä prosessia hidastavaa ylinäytteistystä.*

SVG:t ja esitystarkkuutta suuremmat PNG-tiedostot vain hieman toisistaan poiketen, mutta Chrome poikkeaa näiden kuvien kohdalla enemmän. Sen sijaan esitystarkkuudella tallennetut ja sitä pienemmät rasterikuvat tulostuivat identtisesti

(a) PNG-pakattu (b) JPG-pakattu

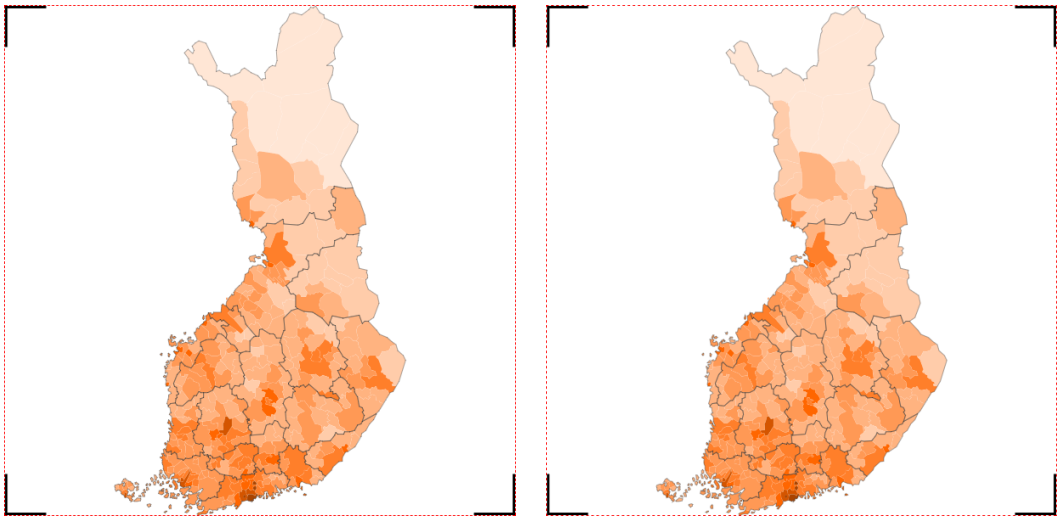
Kuva 4.6 *Vertailu häviöttömän PNG-pakkauksen ja häviöllisen JPG-pakkauksen välillä. Terävä kontrasti valkoisen taustan ja mustan kiilakuvion rajalla aiheuttaa sen, että tasaiselle valkoiselle taustalle muodostuu irrallisia tummempia pikseleitä. Kuvattuna on korkealaatuisesti pakattu kuva, ja artefaktit pahenevat pakkaustasoa nostettaessa.*

kaikilla testatuilla selaimilla, mikä voidaan katsoa rasterigrafiikan eduksi. Tulos on hieman yllättävä, ja viittaa siihen, että selaimet käyttävät yhteistä toteutusta rasterigrafiikan ylöspäin interpoloinnille.

Testikuva 'Ympyrä' osoittaa hyvin, miksi JPG ei sovellu piirrosgrafiikan tallentamiseen. Syy näkyy kuvasta 4.6, jossa vertaillaan häviöttömästi pakattua PNG:tä ja häviöllisesti pakattua JPG:tä. JPG:nä tallennetussa kuvassa 4.6(b) näkyy tummia laikkuja mustan kiilakuvion ympärillä siinä, missä kuvassa 4.6(a) näkyy tasaista valkoista. Pakkausartefaktit erottuvat selkeästi juuri kohdissa, joissa terävää kontrastia ympäröi tasaisen värinen alue. Valokuvissa ja vastaavissa tällaisia alueita ei ole ja pakkausartefaktit hukkuvat kuvan yksityiskohtien sekaan. Kuvasta 4.2 nähdään, että vertailtava PNG-tiedosto on kooltaan 334,1 kilotavua ja vastaava JPG-tiedosto vain 162,0 kilotavua.

4.1.2 Testikuva 'Suomi'

Testikuva 'Ympyrä' ei sovellu mallintamaan tavanomaisen verkkosivugrafiikan hahmontumista, mutta sen paljastamat käyttäytymiserot selainten välillä ovat mielenkiintoisia. Kuva 4.7 osoittaa, että tyypillisempi SVG-pohjainen kuva 'Suomi' hahmontuu selainten välillä yhdenmukaisemmin. Ainoastaan kuntarajoja esittävien polkujen pääte- ja taitepisteisiin näyttää Chromella ilmestyvän pieniä mustia täpliä,



(a) Chrome (b) Firefox

Kuva 4.7 *Vertailu tyypillisen SVG-kuvan hahmontumisessa Chromen ja Firefoxin välillä. Kuvat ovat lähes identtisiä, eikä eroja huomaa vierekkäin vertaamalla.*

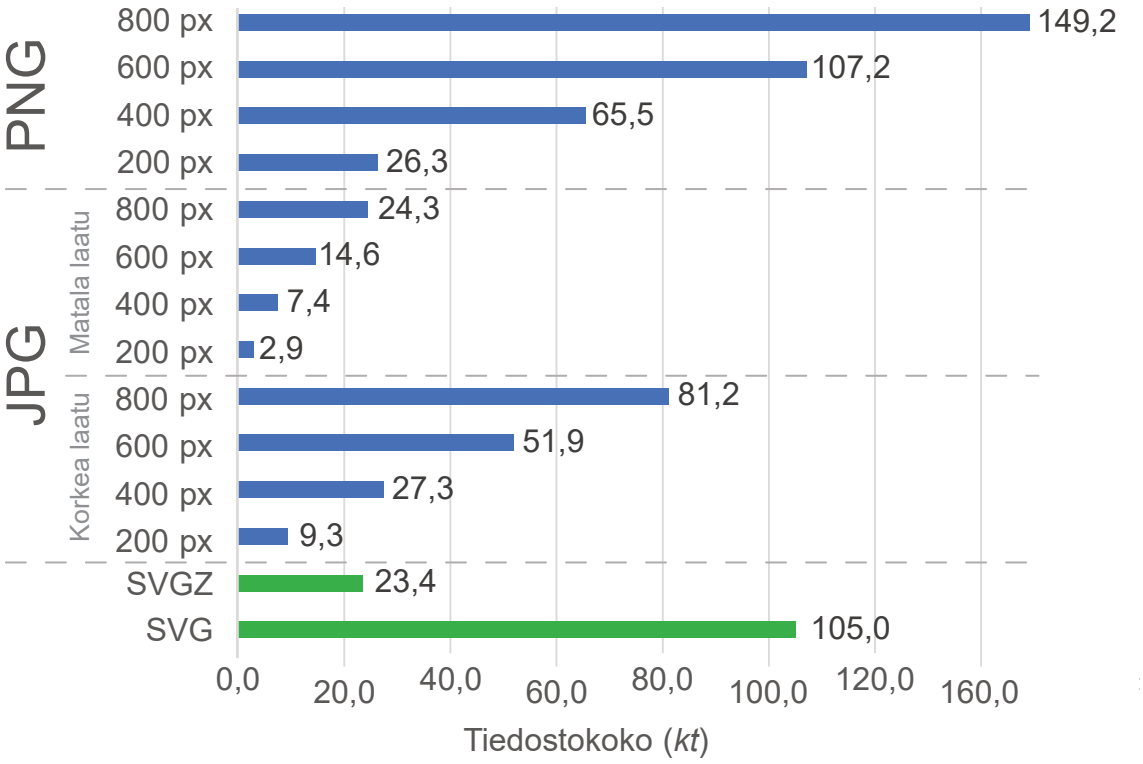
mutta eron huomaa ainoastaan tarkasti katsoen. Ero on niin vähäinen, että selainten välinen hahmontumisero ei näyttäisi olevan syy olla käyttämättä SVG:tä.

Kun verrataan testikuvan 'Suomi' (3.1(b)) PNG-versioita esitysresoluutiossa ja sitä suuremmassa resoluutiossa, huomataan, että ylöspäin skaalaamisen lisäksi myös alaspäin skaalaus heikentää grafiikan visuaalista laatua. Tämä käy ilmi kuvasta 4.8, jossa alaspäin skaalatussa kuvassa esiintyvä reunaviiva on sumeampaa kuin skaalaamattomassa. Tämän vuoksi grafiikan toimittaminen liian tarkkanakaan ei ratkaise rasterigrafiikan laatuongelmia täysin, vaan kuvanlaadun maksimoimiseksi materiaalista pitäisi aina olla saatavilla oikean tarkkuuden versio. Tämä löydös korostaa SVG:n käytännöllisyyttä rasterigrafiikkaan nähden.

Jos siis kuvista on oltava versiot useammalle eri skaalaustasolle, tarvitaan palvelimella enemmän tilaa eri versioille. Ainakin tätä menetelmää käyttävä Retina.JS-kirjasto [17] myös noutaa jokaisesta sivuston kuvasta alkuperäisen lisäksi korkeatarkkuuksisen version, mikä lisää oleellisesti selaimen kaistankäyttöä. Kuvassa 4.9 on kaavioituna testikuvan eri versioiden tiedostokoot, ja siitä nähdään, että eri kokoiset PNG-tiedostot vievät yhteensä 348,2 kilotavua. Pakkaamaton SVG vie tilaa hieman vähemmän, kuin esitystarkkuuksinen PNG. Toisaalta pakattu SVGZ vie vähemmän, kuin pienin PNG-tiedosto. Tulos on hieman yllättävä siksi, että kyseessä on kohtalaisen yksityiskohtainen kuva ilman, että siinä juurikaan on visuaalisesti toisteista sisältöä.

(a) Esitysresoluutio (b) Alaspäin skalaattu

Kuva 4.8 *Vertailu skaalaamattoman ja alaspäin skaalatun PNG-materiaalin välillä. Kuvankaappaus on otettu Chromesta.*



Kuva 4.9 *Testikuvan 'Suomi' tiedostokoot eri formaateissa. SVG on kohtalaisen kokoista verrattuna PNG:hen, joka on piirrosgrafiikkaan parhaiten soveltuva rasterigrafiikkaformaatti.*

Tulokset osoittavat, että SVG on piirrosgrafiikan tallentamisessa tiedostokooltaan vastaavaa tasoa, kuin PNG. Pakattu SVG on molemmissa testikuvissa pienempi tiedosto kuin mikään tutkituista PNG-versioista. Testikuvan 'Ympyrä' kohdalla SVGZ

tiedosto oli jopa vain 8 prosenttia pienimmän PNG-version koosta, mutta tämä johtuu kuvan sisältämän toisteisen grafiikan kloonattavuudesta. Käytännössä näin hyvin optimoitavia tapauksia tuskin syntyy usein. Tuloksista huomataan myös, että rasterigrafiikan skaalaaminen heikentää kuvan laatua huolimatta siitä, tehdäänkö skaalaus ylös vai alas päin. Tiedostokoon ja hahmontumislaadun perusteella SVG kykenee hyvin korvaamaan PNG:n, ja koska SVG on resoluutioriippumatonta, se soveltuu tyypillisiin PNG:n käyttökohteisiin jopa PNG:tä paremmin.

4.2 Testivalokuvat

Loput kaksi testikuvaa olivat valokuvia, joten piirrosgrafiikkaa vastaavan vertailuprosessin mahdollistamiseksi kuvista oli tehtävä vektoriversiot. Tätä varten on käytettävä vektorointityökalua, joka tässä tapauksessa oli Adobe Illustrator -piirto-ohjelmisto.

Silmämääräisesti vertailtuna biljardikuva 3.1(c) ja sen pohjalta vektoroitu kuva 3.2(a) näyttävät hyvin samanlaisilta, eli laadukas vektorointi ei ainakaan oleellisesti heikennä kuvan laatua. Vertailu tornin kuvien 3.1(d) ja 3.2(b) välillä tukee havaintoa. Vektoroitujen kuvien skaalaaminen kuitenkin paljastaa rasteri- ja vektoriversioiden erot. Kuvasta 4.10 nähdään, millainen tulos Chromen rasterigrafiikan interpoloinnilla on verrattuna vektoroidun version skaalaamisen tulokseen. Vertailusta havaitaan, että skaalaus tuo esille vektorointialgoritmin nauhoittumisongelman: esimerkiksi biljardipallosta heijastuvat valaisimet muodostavat pehmeän valkoisen läikän rasterikuvaan, mutta vektorikuvassa heijastukset esiintyvät selkeästi asteittaisina laikkuina. Myös numeron 15 erottaminen on hankalampaa vektorikuvasta.

Tornissa selkeää nauhoittumista ei synny, kuten nähdään kuvasta 4.11(b). Sen sijaan havaitaan, että pienen sävyvaihtelun alueet tulostuvat tasaisen värisinä. Tätä ilmiötä kutsutaan posterisoitumiseksi [1], ja sen lopputulos muistuttaa enemmän öljyvärimaalausta kuin valokuvaa. Tämäkin puute paljastuu vasta suurennoksessa, mutta ilmiö ei ole yhtä häiritsevä, kuin nauhoittuminen.

Kuten todettu, valokuvamateriaalissa vektorigrafiikan luomiseksi tarvittavaa metatietoa ei ole saatavilla. Ilman lisäinformaatiota vektorointialgoritmikaan ei kykene päättelemään kuvan semanttista sisältöä. Ainakin tällä hetkellä vektorointialgoritmit yksinkertaisesti kynnystävät kuvan ja luovat tuloksen reunaviivoja seuraavan polun. Tämä toistetaan eri kynnysarvoilla ja erikseen eri väreille, jolloin saadaan koostettua vektoroitu täysivärikuva. Prosessissa jokaista värisävykaistaa vastaa yksi polku. Koska eri värejä voi olla miljoonia yhdessä kuvassa, visuaalisesti riittävä

(a) Alkuperäinen (b) Vektoroitu

Kuva 4.10 *Testikuvan 'Biljardi' alkuperäisen ja vektoroidun version viisinkertainen suurennos. Vektoroidussa kuvassa on korostettuna nauhoittumisilmiön aiheuttamia reunoja.*

(a) Alkuperäinen (b) Vektoroitu

Kuva 4.11 *Testikuvan 'Torni' alkuperäisen ja vektoroidun version kolminkertainen suurennos.*

yksityiskohtaisuus vaatisi valtavan määrän polkuja, ja siksi tulos viekin valtavasti tallennustilaa.

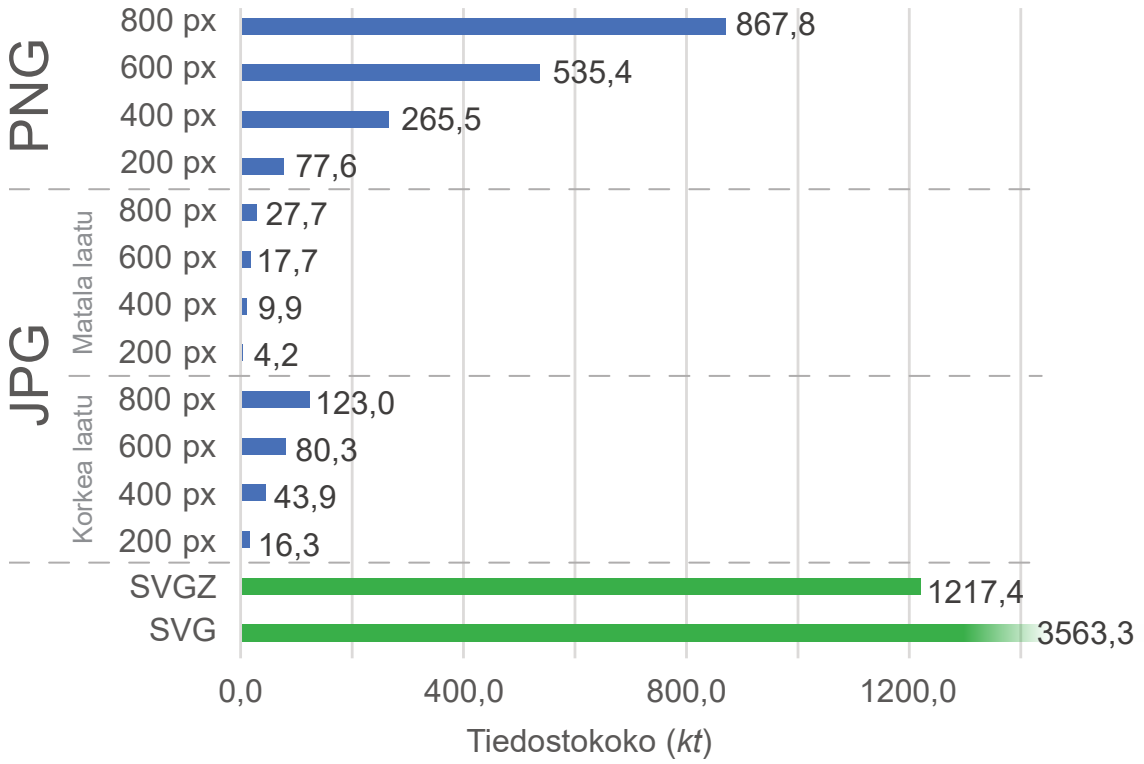
Huomattava tiedostokoko käykin ilmi kuvasta 4.12. Testikuva 'Biljardi' vie 800 pikselin levyisenä PNG:nä 867,8 kilotavua tilaa, mutta vektorointi nostaa tiedostokoon 3 563,3 kilotavuun. Testikuva 'Torni' on PNG:nä vastaavasti 778,7 kilotavua ja vektoroituna 3 493,2 kilotavua. Vektoroidut kuvatiedostot ovat siis yli nelinkertaisia tiedostokoo'iltaan. Pakkaaminen tosin tiputtaa kokovaatimukset kuville arvoihin 1 217,4 kilotavua ja 1 167,9 kilotavua järjestyksessä, mutta kokoa jää edelleen rasteriformaattia enemmän.

Tulokset osoittavat, että vektorointialgoritmit eivät ratkaise rasterigrafiikan skaalautumattomuusongelmaa. Vektoroitu rasterikuva saattaa skaalattuna näyttää jopa heikkolaatuisemmalta, kuin jos skaalataan alkuperäistä rasterikuvaa asianmukaisella interpolaatiomenetelmällä. Kun vielä huomioidaan vektoroidun materiaalin suurempi tilantarve, voidaan tehdä johtopäätös, että SVG ei sovellu valokuvien tai muun fotorealistisen materiaalin tallentamiseen eikä esittämiseen. PNG:n häviötön pakkausalgoritmi taas jättää tiedostokoon JPG:tä suuremmaksi, vaikka JPG:n häviöllinen pakkaus ei välttämättä heikennä valokuvien visuaalista laatua havaittavasti. JPG on tässä käytössä siksi edelleen varteenotettavin tiedostoformaatti.

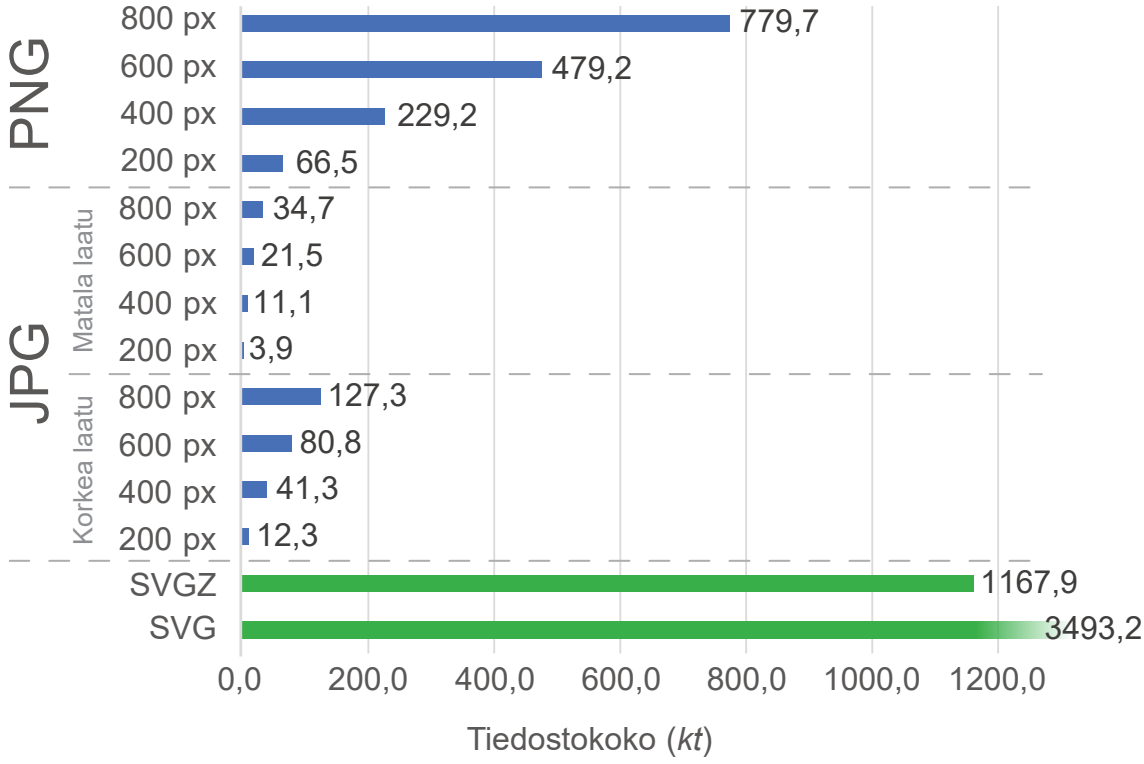
4.3 Tiedostokoon merkitys verkkosivuilla

Tiedostokoko heijastuu verkkosivukontekstissa kaistankäyttöön, johon W3C:n julkaisu *Mobile Web Application Best Practices* ottaa kantaa [45]. Dokumentin luvussa 3.4 eritellään optimaalisen verkon käytön kannalta oleellisia toimintamalleja. Luvun pääasiallinen sanoma on vähentää palvelinpyyntöjen määrää, vaikka se lisäisi pyyntöjen kokoa. Tämä puoltaa sitä, että grafiikka toteutettaisiin HTML-merkintään upotettuna SVG:nä, jolloin dokumentin kuvamateriaali ei edellytä erillisiä kutsuja. Luvun alaisuudessa olevat kohdat täsmentävät suositusta, ja näistä SVG:tä koskettavat alakohdat ovat 3.4.1, 3.4.7 ja 3.4.11.

Kohta 3.4.1 suosittelee käyttämään komposiittikuvia erillisten kuvien sijaan dokumentin visuaalisen ilmeen ehostamiseen. Saman ohjeistuksen antaa myös Jakob Nielsen kirjassaan *WWW-suunnittelu* [46]. Käytännössä menetelmä tarkoittaa, että komposiittikuvasta kohdistetaan näkyviin oikea kohta CSS-asemoinnilla. Menetelmä onnistuu SVG:lläkin, mutta DOM-rakenteeseen suoraan liitettävät kuvat voidaan toteuttaa myös hyödyntämällä SVG:n `<use>`-elementtiä. Määrittelemällä kaikki graafiset objektit yhdessä `<svg>`-elementissä `<defs>`-elementin sisällä ja sitten viittaamalla niihin muista `<svg>`-elementeistä `<use>`-elementtien avulla voidaan kerran mää-



(a) Testikuva 'Biljardi'



(b) Testikuva 'Torni'

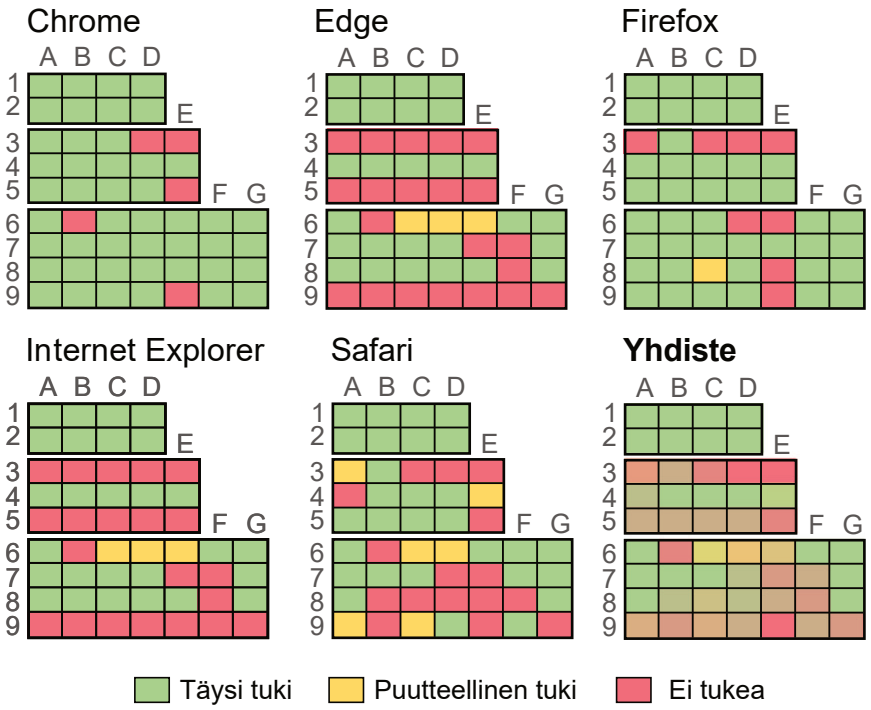
Kuva 4.12 *Hahmonnuksen testaamiseen käytettyjen rasteripohjaisten testikuvien tiedostokoot. Vektorointialgoritmin generoimat SVG-versiot ovat huomattavan kookkaita.*

riteltyä objektia käyttää suoraan muissa sijainneissa. `<use>`-elementtimenetelmä on selkeämpi, koska oikean kuvaleikkeen poimiminen on eksplisiittistä. Komposiittikuvan asemointien asettaminen ja laskeminen ovat siihen nähden työläitä ja syntyvästä koodista on vaikea tulkita, mitä kuvaleikettä milloinkin esitetään.

Edelleen kohta 3.4.7 ohjeistaa upottamaan taustakuvina käytettävän rasterigrafiikan upottamista CSS-tiedostoihin niin sanottuina data-URL:na. Tämänkin voidaan katsoa puoltavan SVG:n upottamista suoraan HTML:n mukaan, sillä toimenpide ajaa saman asian. Toisaalta kohta 3.4.11 kehottaa pitämään DOM-rakenteen maltillisen kokoisena päätelaitteen rajallisen muistin vuoksi, mutta tämä ei liene enää ongelma nykypäivänä. Moderneissa laitteissa on enemmän muistia, mutta mobiiliverkko on toisinaan edelleen laadultaan puutteellinen.

4.4 Selaintuki

Testattujen selainten SVG:n tuen taso on esitetty kuvassa 4.13. Taulukkojen solujen sijainnit vastaavat käytettyjen testitaulukoiden sijainteja, ja solun väritys kertoo, kuinka hyvin kyseinen selain tukee kyseistä ominaisuutta. Vihreä tarkoittaa täyttä tukea, keltainen puutteellista tukea ja punainen puuttuvaa tukea.



Kuva 4.13 *Valtaselainten uusimpien versioiden tuki testatuille SVG-ominaisuuksille. Rivit ja sarakkeet vastaavat testitaulukoiden rivejä ja sarakkeita (katso liite C).*

Kuvasta 4.13 huomataan, että kaikissa taulukoissa ensimmäiset kaksi riviä ovat kokonaan vihreitä. Tämä merkitsee, että tuki staattisen SVG:n esittämiseen löytyy kaikista moderneista testatuista selaimista. Löydös on linjassa Can I Use? -palvelun tietojen kanssa [5]. SVG:n eri ominaisuuksien syvällisempi testaus aiheutti kuitenkin huomattavasti hajontaa eri selainten välillä. Selkein puute on, että Microsoftin selaimet eivät tue SMIL:iä ollenkaan, mikä näkyy rivin 5 punaisuutena. Riviltä 3 puolestaan nähdään, että Microsoftin selaimet eivät myöskään tue CSS-animaatioita niille SVG:n attribuuteille, joita HTML:ssä ei ole [1]. Solut D7 ja E8 toimivat, koska niissä animoitu attribuutti on myös HTML-elementeiltä löytyvä `letter-spacing`. Myös Firefoxin tuki CSS:llä animoimiselle näyttää rajalliselta rivin 3 perusteella.

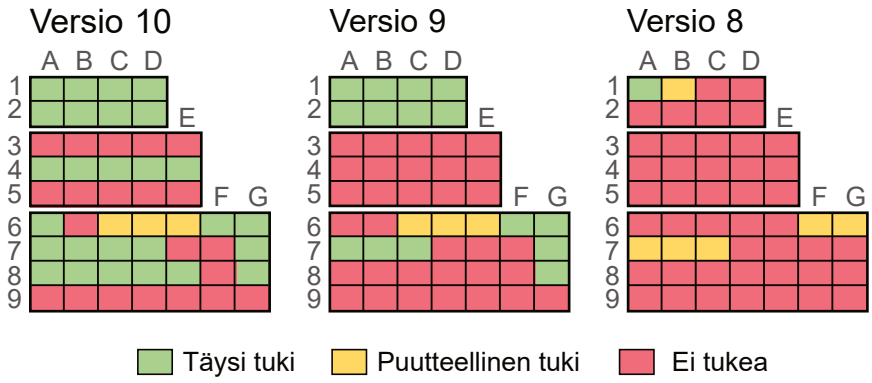
SMIL:in kyky reagoida käyttäjän ärsykkeisiin toimi kaikissa SMIL:iä tukevissa selaimissa, jos SVG on upotettuna HTML-merkintään. Tämä käy ilmi solusta D9, jossa ympyrän pulssianimaation kuuluu käynnistyä klikattaessa. Vastaava grafiikka liitettynä `<img>`-elementillä solussa E9 puolestaan ei toiminut missään selaimessa, joten käyttäjän syötteet eivät välity `<img>`-elementin läpi grafiikalle.

JavaScriptillä animoidut solut A4-E4 sekä A8-A9 toimivat parhaiten kaikista animoiduista soluista, mikä viittaa siihen, että JavaScript on selaintuen kannalta luotettavin tapa animoida SVG:tä. Tuki löytyy testin perusteella kaikista SVG:tä tukevista selaimista paitsi IE9:stä. Mielenkiintoinen ilmiö tosin on, että maskitestin perusteella SVG:n `<defs>`-osiossa määritetty maskina toimivan `<text>`-elementin sisällön muutos päivittyy sivulle vain selaimen joutuessa hahmontamaan kyseisen kohdan sivusta muista syistä. Näin tapahtuu esimerkiksi silloin, kuin selainikkuna pienennetään tai sen kokoa muutetaan niin, että kohta päätyy ikkunan alan ulkopuolelle. Jos kohta tulee uudelleen näkyviin, päivitetty arvo piirtyy dokumenttiin. Jostain syystä leikkauspolkutestin (solu C6) arvo ei päivity tässäkään tilanteessa. Tämä käyttäytyminen ilmeni Edgessä sekä Safarissa.

SVG:n kilpailevien teknologioiden selaintukea kuvaavat solut F7 ja G7 antavat tietoa siitä, onko selaintuki peruste suosia SVG:tä tai kilpailijoitaan. Solu F7 sisältää APNG-tiedoston, joka on vaihtoehtoinen tapa esittää animoitua sisältöä verkkosivulla. F7 on punaisena Microsoftin selaimia esittävissä taulukoissa ja vihreä muissa, joten sillä on JavaScriptillä animoitua SVG:tä heikompi tuki moderneissa selaimissa. Solun G7 `<canvas>`-elementti puolestaan toimi kaikissa selaimissa ja jopa IE9:ssä, joten selaintuki dynaamiselle grafiikalle on aavistuksen SVG:tä parempi `<canvas>`-elementillä.

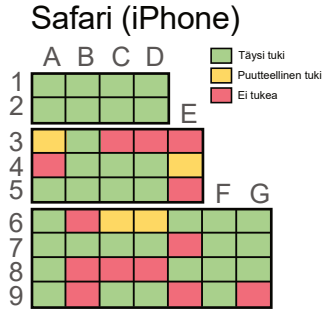
[1] Esimerkiksi `width`-attribuutti löytyy sekä HTML:n `<div>`-elementistä että SVG:n `<rect>`-elementistä, mutta attribuutti `cx` löytyy vain SVG:stä

Kuvassa 4.13 on esitetty pelkästään Internet Explorerin uusin versio 11. Koska Internet Explorer on surullisenkuuluisa hitaasta päivitystahdistaan, testattiin myös sen aiempien versioiden tuki samoille ominaisuuksille. Tulokset on esitetty kuvassa 4.14. Havaitaan, että SVG:n tuki heikkenee merkittävästi vasta versiossa 9, mutta siinäkin staattinen SVG on hyvin tuettu. Versio 8 sen sijaan tukee tämän testin perusteella enää pelkästään SVG:n `<text>`-elementtiä. Toisaalta testin arvo on kyseenalainen, sillä Microsoft on muutenkin lopettanut tuen taulukon 4.14 selainversioille vuoden 2016 alussa [47]. Statcounter-palvelun mukaan IE:n osuus maailmanlaajuisesta verkkosivuliikenteestä on lokakuussa 2017 vain 3,74 % ja siitä 3,11 prosenttiyksikköä on IE:n uusimmalla versiolla [4]. IE:n vanhentuneita versioita käytetään siis niin vähän, ettei SVG:n käyttöä kannata sen vuoksi välttää.



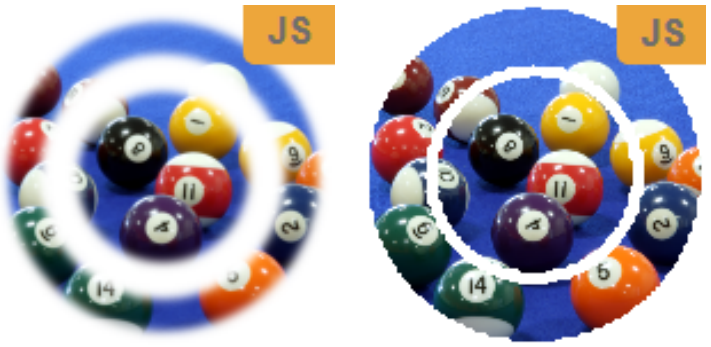
Kuva 4.14 *Internet Explorerin vahnempien versioiden tuki SVG:n ominaisuuksille. Tuki SVG:n esittämiselle löytyy vielä versiosta 9, mutta versio 8 kykenee näyttämään enää SVG:n* `<text>`*-elementtien sisällöt.*

Mobiiliselaimista Androidin Chrome vastasi työpöytäversiotaan, mutta iPhone 6S:llä Safarin tuloste poikkesi työpöytäversiostaan hieman iPhonen eduksi. Tämä huomataan verrattaessa kuvaa 4.13 kuvaan 4.15, joka esittää iPhone 6S:n Safarin tuloksia.



Kuva 4.15 *iPhone 6S:n Safarin tuki testatuille SVG:n ominaisuuksille.*

SVG:n suotimet paljastavat eroja selainten välillä. Chrome ei tue CSS-suotimien



(a) Chrome (b) Firefox

Kuva 4.16 *Testitaulukon solun C8 toiminta Chromessa ja Firefoxissa. Firefox ei tuota pehmeää reunaa kuvalle kuin kuuluisi, vaan tulos on roisoinen ja terävä.*

käyttöä SVG:n elementeille, vaan rajoittaa kehittäjän käyttämään SVG:n omia suodinelementtejä. SVG:n suotimet ovat erittäin monipuolisia, mutta niitä ei voi CSS-suotimista poiketen animoida. Kuvassa 4.16 näkyvä testitaulukon solu C8 osoittaa, että suotimella muokatun muodon käyttö maskina tuottaa virheellisen lopputuloksen Firefoxilla: maskina toimivien renkaiden reunojen tulisi olla pehmeitä, mutta maskin pikselit ovat joko täysin läpinäkyviä tai täysin peittäviä. Myös maskeja testaavat solut D6 ja E6 tuottavat ongelmia Firefoxilla, sillä ne eivät tulostu ollenkaan.

Kaiken kaikkiaan selaintukea testattiin kohtalaisen monipuolisesti, mutta tutkimus jäi silti suunnitellusti yleistasoiseksi. Jo nyt saavutetut löydökset osoittivat, että SVG:n tuki on monelta osin keskeneräistä selaimissa. Tukea on siis syytä seurata ja tutkia syvällisemminkin, jotta selaintuen puutteet voitaisiin kartoittaa täsmällisemmin.

Yleisellä tasolla testiominaisuuksia parhaiten tukivat Chrome ja Firefox. Ne tukevat suurinta osaa testatuista CSS- ja SMIL-pohjaisista ominaisuuksista. Kuvassa 4.13 viimeisenä oleva yhdistetaulukko osoittaa, että kaikissa selaimissa on hyvä tuki SVG:n esittämiseen ja JavaScript-pohjaiseen animointiin. Näiltä osin SVG sopii siis tuotantokäyttöön.

Muut SVG:n animointiin soveltuvat tekniikat CSS ja SMIL eivät ole kattavasti tuettuja eri selaimilla, joten niiden käyttö ei ole suotavaa muuten kuin kosmeettisissa sovelluskohteissa. CSS:n ja SMIL:in avulla animaatiot onkin erityisen käytännöllistä liittää juuri käyttäjän syötteisiin, jolloin niitä voidaan hyödyntää käyttäjäkokemuksen parantamisessa. Asiaan palataan luvussa 5.

4.5 Dynaaminen grafiikka

Dynaamista, eli selaimessa generoitua grafiikkaa testattiin toteuttamalla Boids-algoritmi sekä <canvas>- että <svg>-pohjaisella hahmonnuskomponentilla. Tutkitaan aluksi eri hahmonnuskomponenttien toteutusta. <canvas>-pohjaisen toteutuksen yhden boidin piirron koodi on esitetty listauksessa 4.1. Kuten nähdään, boidi on piirrettävä alusta asti uudelleen, koska uuden kehyksen piirtämiseksi edellinen on pyyhittävä tyhjäksi. Piirtokäskyjen koordinaatit on laskettava erikseen (rivit 4-7) ja uusi boidi syntyy piirtoalustalle viiva kerrallaan (rivit 11-18).

```
1  let absolutePoints: Vector[] = [];

3  // Rotate the corner points according to the boid orientation
   for (let point of this.symbol) {
5   absolutePoints.push(
     point.rotate(boid.orientation).add(boid.location)
7   );
   }
9
   // The draw commands
11 this.ctx.strokeStyle = "white";
   this.ctx.lineWidth = 2;
13 this.ctx.beginPath();
   this.ctx.moveTo(absolutePoints[0].x, absolutePoints[0].y);
15 this.ctx.lineTo(absolutePoints[1].x, absolutePoints[1].y);
   this.ctx.lineTo(absolutePoints[2].x, absolutePoints[2].y);
17 this.ctx.stroke();
   this.ctx.closePath();
```

Listaus 4.1 *Boidin piirto <canvas>:lla. Jokainen boidi on piirrettävä viiva kerrallaan uudestaan piirtoalustalle.*

SVG-toteutuksen yhden boidin sijainnin päivittäminen on esitetty listauksessa 4.2. Toteutuksen koordinaatistojen siistimiseksi jokainen boidi on omassa <g>-elementissään ja itse boidi on tosiasiassa <use>-elementti. <g>-elementti määrittää boidin sijainnin kun taas asento asetetaan suoraan <use>-elementille. <canvas>-toteutukseen verrattuna SVG:llä kyseessä on todella boidin siirto eikä vanhan hävittäminen ja uuden piirtäminen oikeaan kohtaan ja oikeassa asennossa. Tässä toteutuksessa selain huolehtii siirtymän piirtämisestä heti, kun se on koodissa asetettu.

```
boid.element.setAttribute("transform",
 `translate(${String(boid.location.x)}
            ${String(boid.location.y)})`);
let $use = boid.element.firstElementChild;
$use.setAttribute("transform",
 "rotate(" + (boid.orientation * 180 / Math.PI) + ")")
```

Listaus 4.2 *SVG-pohjaisen boidin sijainnin ja asennon päivittäminen. Uudet arvot sijoitetaan elementtien* `transform`*-attribuutteihin.*

Boids-algoritmin toteutusten vertailu osoittaa, että `<canvas>`-elementin käyttäminen dynaamisen grafiikan generointiin ja manipulointiin on varsin matalan tason ratkaisu. Uuden kehyksen piirtäminen sillä edellyttää edellisen kehyksen pyyhkimisen ja jokaisen objektin manuaalisen piirtämisen uudelleen piirtoalustalle. SVG:llä sitä vastoin riittää, kun jokaiselle objektille merkitään uusi sijainti ja asento, jolloin selain huolehtii kehyksen uudelleenpiirtämisestä. Kun taas tutkitaan hahmontimien rakennusmetodeja, niin SVG vaatii paljon enemmän alustustoimenpiteitä, kuin `<canvas>`. Nämä alustustoimenpiteet tosin voitaisiin tehdä valmiiksi dokumenttiin joko palvelimella tai staattisesti sivua kehitettäessä. `<canvas>`-elementin sisällön asettaminen valmiiksi palvelinpuolella ilman JavaScriptiä ei ole mahdollista.

SVG:n kohdalla myös mukautuminen erilaisille näyttöresoluutioille ei edellytä lisää koodattavaa, koska SVG:n sisäinen koordinaatisto ei ole sidoksissa näytön resoluutioon. SVG-kuvan koordinaatiston näkyvä osa voidaan määritellä mielivaltaisesti `<svg>`-elementin `viewBox`-attribuutilla. Kuvan fyysinen koko näytöllä määritellään erikseen CSS:llä tai `width` ja `height` attribuuteilla. Jos koko määritellään suhteellisena, selain huolehtii skaalaamisesta automaattisesti.

Dynaamisen grafiikan kohdalla vertailtiin SVG:n ja *<canvas>*-elementin suorituskykyä Chromen versiolla 61 ja Firefoxiin versiolla 56. Tulokset testilaitteelle 1 on esitetty taulukossa 4.1 ja testilaitteelle 2 taulukossa 4.2. Tulokset kertovat, että simuloitavien boidien määrä saa ruudunpäivitystaajuuden Chromella tippumaan alle 60 FPS rajan 200 kappaleen kohdalla. Firefoxilla päivitystaajuus laskee joka askeleella, mutta merkittävän alhaiseksi se laskee molemmilla vasta 250 kappaleessa. Edge ja Internet Explorer eivät olleet tutkittavissa tällä menetelmällä, koska niissä suorituskyvyn arviointitoiminto itse pudottaa päivitystaajuutta huomattavasti. Vertailukelpoista tulosta ei ole mahdollista saada, mutta silmämääräisen arvion perusteella voidaan olettaa, että yleinen käyttäytyminen olisi vastaavanlaista.

Chromen tarkempi suorituskykyraportti molempien hahmonnuskomponenttien toiminnasta löytyy liitteestä E. Siitä nähdään pullonkaula, jossa selain suorittaa vain prosessia *Recalculate style*. Tämä osoittaa, että SVG:n heikompi suorituskyky joh-

Taulukko 4.1 *Piirtoalustan ja SVG:n suorituskykyerot testilaitteella 1.*

Boidien määrä	Ruututaajuus (*FPS*)			
	Chrome		Firefox	
	`<canvas>`	`<svg>`	`<canvas>`	`<svg>`
50	60	60	59	53
100	60	60	59	51
150	60	60	59	45
200	60	35	58	39
250	53	27	54	26
300	44	24	38	22

Taulukko 4.2 *Kangaselementin ja SVG:n suorituskykyerot testilaitteella 2.*

Boidien määrä	Ruututaajuus (*FPS*)			
	Chrome		Firefox	
	`<canvas>`	`<svg>`	`<canvas>`	`<svg>`
50	60	59	58	55
100	60	57	59	49
150	54	42	59	47
200	43	30	52	34
250	39	25	41	28
300	38	21	33	25

tuu JavaScript-pohjaisen animoinnin edellyttämästä DOM-manipulaatiosta: koska boidin muuttunut sijainti asetetaan boidia vastaavan `<use>`-elementin `style`-attribuutin avulla, selain joutuu laskemaan tyylin vaikutukset uudestaan ennen, kuin uusi näkymä voidaan piirtää. Tämä viivästyttää kehyksen piirtoa. Jos kehittäjän ei tarvitsisi animoida näkymää JavaScriptillä ruutu kerrallaan, suorituskyky olisi luultavasti huomattavasti parempi. Siinä tapauksessa selain voi ohjata piirtokäskyt suoraan suorittimelle tai jopa grafiikkapiirille ilman, että se vaatii JavaScriptin tulkitsemista tai DOM-manipulaatiota.

W3C:n julkaisema dokumentti *Mobile Web Application Best Practices* ottaa kantaa dynaamisen grafiikan toteuttamiseen mobiililaitteilla kohdassa 3.7.1. Siinä todetaan, että jos nopeus on tärkeää, `<canvas>` saattaa olla tehokkaampi. Toisaalta, koska `<canvas>` luo litteän bittikartan, se on luontaisesti huonosti saavutettavaa, eikä sitä siksi tulisi käyttää ainoana tapana esittää tietoa. Siksi kohdassa suositellaan käyttämään SVG:tä, mikäli kuvan sisältöä on manipuloitava ja kangasta, mikäli pelkkä dynaaminen luonti riittää [45]. Samansuuntaiset ohjeet löytyvät kirjasta *Introducing HTML5* [9].

5. SVG:N ERITYISPIIRTEET VERKKOSIVUTEKNIIKKANA

SVG on verkkosivuteknologiana erityinen grafiikkaformaatti. Sitä voidaan muun muassa käyttää verkkosivulla sekä itsenäisenä tiedostona että verkkosivun rakenteeseen liitettynä. Tässä mielessä SVG on paitsi grafiikkaformaatti myös dokumenttiformaatti. Siksi sen käyttämisessä on huomioitava muutamia näkökohtia, jotka eivät koske työskentelyä rasterigrafiikan parissa. Tämä luku täydentää edeltävien lukujen löydöksiä listaamalla oleellisimmat erityisnäkökohdat, joista verkkosivukehittäjän on syytä olla tietoinen kehittäessään verkkosivuja SVG:tä hyödyntäen. Luvussa pyritään löytämään vastaus tämän diplomityön toiseen tutkimuskysymykseen: mitä erityispiirteitä SVG:ssä on verkkosivutekniikkana? Käsiteltävät erityispiirteet perustuvat osittain työn taustatutkimuksessa esiintulleisiin kirjallisuuslähteisiin ja osittain kirjoittajan ammatillisiin havaintoihin verkkosivusuunnittelijana ja -kehittäjänä.

5.1 Yhdistegrafiikka

Perinteinen tapa täydentää valokuvamateriaalia piirroksin on ollut lisätä piirrosmateriaali suoraan valokuvaan ja tallentaa tulos jossain rasterigrafiikkaformaatissa. Myös tähän menetelmään liittyy metatiedon menetys ja häviöllistä pakkausta käytettäessä myös piirrosmateriaalin laadun heikkeneminen. Kuten luvussa 4 huomattiin, SVG:llä ei kannata yrittää esittää jatkuva-aikaista grafiikkamateriaalia. Sen sijaan tällainen grafiikkamateriaali on mahdollista upottaa SVG-kuvaan `<image>`-elementin avulla. Liitettävä kuva voidaan linkittää SVG:hen URL:ia käyttäen, eli selain voi noutaa sen jostain muusta sijainnista. Toinen vaihtoehto on upottaa se SVG-kuvan mukaan data-URL:n avulla.

Näin voidaan yhdistää molempien grafiikkalajien edut yhteen kuvatiedostoon. Piirrosmateriaali säilyy skaalautuvana, teksti säilyy tekstinä, kuvaan voidaan soveltaa SVG:n suotimia, valokuvaa voidaan rajata vektorikuvioilla ja tämä kaikki on selaimessa dynaamisesti muokattavissa sekä animoitavissa. Rasterigrafiikan skaalaus jää esittävän ohjelman interpolaatioalgoritmin varaan, mutta luvussa 4 tulos havaittiin paremmaksi, kuin vektoroimalla tuotettu kuva. Yhdistegrafiikan skaalautuvuutta

(a) Rasterigrafiikka (b) Yhdistegrafiikka

Kuva 5.1 *Vertailu pelkän rasterigrafiikan ja yhdistegrafiikan ylöspäin skaalauksen välillä. Rasterigrafiikka on kauttaaltaan pikselöitynyt, mutta yhdistegrafiikassa vektoripohjaiset elementit säilyttävät terävät reunat.*

havainnollistaa kuva 5.1, joka on alunperin julkaistu 200 pikseliä leveänä ja korkeana. Rasteriversiossa ylöspäinskaalauksen haitat vaikuttavat koko kuvaan, mutta yhdistegrafiikassa vain valokuvamateriaalia joudutaan interpoloimaan. Teksti, oranssi tausta ja rasterikuvaa rajaava hammasratas ovat reunoiltaan teräviä.

5.2 Tietoturva

Tämän luvun johdannossa mainittiin, että SVG on myös dokumenttiformaatti. Tietoturvamielessä SVG onkin tärkeää sellaisena mieltää, sillä formaattiin kuuluu mahdollisuus sisällyttää JavaScriptiä. Seikka on syytä huomioida erityisesti, jos käsitellään ulkopuolisesta lähteestä peräisin olevaa SVG:tä. Tästä esimerkkinä on tapaus, jossa henkilö avasi linkistä SVG-kuvan. Tähän SVG:hen upotettu JavaScript ohjasi selaimen sivulle, joka yritti asentaa Chromeen haitallisen liitännäisen. Liitännäinen ei näkynyt selaimen käyttöliittymässä, mutta se lähetti käyttäjän Facebook-tilillä roskapäivityksiä ja lisäksi poisti Facebookin dokumenttirakenteesta päivitysten muokkaus- ja poistopainikkeet.

Tietoturvan kannalta SVG onkin kiistatta rasteriformaatteja arveluttavampaa. Mario Heiderich et al. on kirjoittanut aiheesta julkaisun vuonna 2011 [48]. Sen mukaan selainvalmistajat kohtelevat SVG:tä pelkkänä grafiikkaformaattina ylenkatsoen seikan, että formaatti voi sisältää haitallista JavaScriptiä, Flash-sovelluksen tai muuta ajettavaa koodia. Tämä merkitsee, että selainten suojatoimet esimerkiksi XSS-

hyökkäyksiä vastaan eivät välttämättä kohdistu SVG-tiedostoihin. Julkaisun mukaan tietoturvariskin kannalta ei ole merkitystä käsitelläänkö SVG:tä `<img>`- tai `<object>`-elementin kautta, CSS-taustakuvana vai HTML5-dokumenttiin suoraan lisättynä. Riski on olemassa jopa silloin, kun selainta käytetään pelkästään SVG:n katseluun. Julkaisu kiinnittää myös huomion siihen, että sähköpostisovelluksetkin käsittelevät HTML:ää ja voivat siten olla alttiina SVG:tä hyödyntäville hyökkäyksille.

Tietoturvariskien vuoksi esimerkiksi suosittu sisällönhallintajärjestelmä ja blogialusta WordPress ei oletuksena salli SVG-tiedostojen lisäämistä mediakirjastoonsa [49]. Oletettavasti valtaosa julkaisun paljastamista haavoittuvuuksista on nykyään korjattu ja selainkehittäjien suhtautuminen SVG:hen on tältä osin parantunut. Silti on syytä suhtautua edelleen varauksella esimerkiksi sivuston käyttäjien koneilta peräisin oleviin SVG-tiedostoihin tai linkkeihin, kuten ylempänä esitelty tuore tapaus osoittaa. Onneksi tänä päivänä verkkosivukehittäjän on mahdollista käyttää suodatustyökaluja varmistamaan, että SVG ei sisällä arveluttavaa koodia. Yksi tällainen työkalu on Darryll Doylen kehittämä *svg-sanitizer* [50].

5.3 Grafiikan tuottaminen

Grafiikkalajin vaihtaminen rasterigrafiikasta vektorigrafiikkaan on luonnollisesti merkittävä muutos henkilölle, jonka tehtävä on tuottaa kyseinen grafiikka. Tämä aliluku erittelee SVG:n erityisominaisuuksia *verkkosivusuunnittelijan* näkökulmasta. Työn taustatukimusmateriaalissa olikin yksi suunnittelijan diplomityö, jossa todettiin SVG:n olevan hankalaa tuottaa [11]. Tämä aliluku ottaa kantaa kyseiseen väitteeseen.

Adoben Photoshop on vakiinnuttanut asemansa graafisen suunnittelun johtavana työkaluna. Sitä on sopeuduttu käyttämään myös verkkosivugrafiikan tuottamiseen [11], vaikka kyseessä on valokuvien manipuloimiseen ja rasterigrafiikan tuottamiseen tarkoitettu työkalu. Verkkosivustojen ulkoasut ovat kuitenkin edenneet viistetyistä, varjostetuista ja heijastavista ikoneista jo hillitympiin ja tasavärisiin yksityiskohtiin. Siksi nimenomaan vektorigrafiikkaa tuottaviin *piirto-ohjelmiin* on mahdollista siirtyä. Grafiikan luomisprosessi on suoraviivaisempi ja lopputulos käytännössä automaattisesti tarkkuudeltaan kelvollinen mihin tahansa käyttöön. Vektorigrafiikka ei kuitenkaan tarkoita, että pelkästään yksivärisiin muotoihin tai viivapiirroksiin olisi tyydyttävä, vaan myös tyypillisimmät kuvasuotimet ja väriliu'ut kuuluvat SVG:n standardiin. Vektorigrafiikalla on siten mahdollista luoda monimutkaisiakin skaalautuvia kuvia ja käyttöliittymäelementtejä, mikäli tarpeen.

Vektorigrafiikka vastaa luonteeltaan paremmin ihmisen havainnointia ja mielellistä mallia. Jos vektorigrafiikkadokumentissa on keskellä neliö, voi piirto-ohjelmalla manipuloida tuota neliötä esimerkiksi skaalaamalla tai muokkaamalla sen väriä. Rasterigrafiikassa taas tämä semanttinen data puuttuu: ”neliö” on tosiasiassa vain värillisistä pisteistä koostuva taulukko. Neliön ominaisuuksia (kuten sivun pituutta tai täyttöväriä) ei voi muokata helposti ja työskentelyn helpottamiseksi se tulee manuaalisesti eristää omalle tasolleen. Vektorigrafiikan kohdalla tasoilla voidaan järjestellä dokumentin rakennetta, mutta niiden käyttö ei ole välttämätöntä. Kun dokumentin semanttinen rakenne vastaa kuvan visuaalista sisältöä, työkalu osaa aina kohdistaa käyttäjän syötteen oikeaan objektiin.

Tunnettuja piirto-ohjelmia ovat Adobe Illustrator, Corel Draw, Affinity Designer ja Inkscape. Näistä viimeistä lukuunottamatta kaikki ovat kaupallisia vektorigrafiikkapiirto-ohjelmia, jotka kykenevät viemään dokumentteja SVG-tiedostoiksi. Inkscape eroaa näistä paitsi ilmaisuudellaan, myös sillä, että se on nimenomaan SVG-editori. Se siis käyttää sisäisenä formaattinaan SVG:tä ja suuri osa sen toiminnoista kuvautuu suoraan SVG:n ominaisuuksiin. Lisäksi siihen on sisäänrakennettu toiminto, jolla käsiteltävän SVG dokumentin merkintäkoodia voi muokata suoraan. Tämä erikoisuus voi olla hyödyllinen erityisesti silloin, jos luotava materiaali on tarkoitus muotoilla tai manipuloida selaimessa, sillä syntyvän dokumentin rakenne on hyvin pitkälle käyttäjän määriteltävissä.

Mikään näistä sovelluksista ei kuitenkaan tuota automaattisesti SVG-dokumentteja, jotka sellaisenaan sopisivat DOM-manipulointiin verkkosivukehittäjän toimesta. Sovelluksessa tulisi suoraviivaisemmin voida määrittää elementtien tunniste- sekä luokka-attribuutit ja niiden hierarkkinen rakenne tulisi olla mahdollisimman yksinkertaista. Ongelmallisia kehittäjän kannalta ovat esimerkiksi seuraavat:

- Sekava CSS-tyylien hyödyntäminen. Manuaalinen luokkien ja tyylien hallinnointi voisi tehostaa julkaisuprosessia.
- Turhaan generoituvat `<g>`-elementit, joihin liittyy monimutkaisia alkeismuunnoksia. Lopputuloksena on useita sisäkkäisiä ja toisiinsa nähden vääristyneitä koordinaatistoja, joiden välisiä suhteita on vaikea hahmottaa.
- Resurssien turha toisteisuus, vaikka esimerkiksi kerran määritettyä väriliukua voisi käyttää useissa eri elementeissä.
- Tarpeeton `<tspan>`-elementtien käyttö tekstin asetteluun ja tyylitykseen.

Jos sovelluksella julkaistaan SVG:tä verkkosivukäyttöön, on usein tärkeää varmistaa,

että teksti viedään tiedostoon tekstinä eikä poluiksi muunnettuna. Näin tieto kuvan sisällöstä säilyy luettavana ja muokattavana, mistä voi olla hyötyä sisällön siirrettävyyden kannalta vaikkapa ruudunlukijaa käytettäessä. Mikäli sisältö kohdennetaan esitettäväksi alkuperäisellä tarkkuudella niin, että tiedoston sisäinen koordinaatisto vastaa näytön pikselikoordinaatteja, voi olla syytä pyrkiä asemoimaan elementit linjaan pikseleiden kanssa. Esimerkiksi reunaviivaltaan pikselin levyinen suorakulmio toistuu terävimmillään, jos sen reunaviivat asettuvat pikseleiden puoliväliin. Silloin kuvion reunaviivan reuna osuu täsmälleen pikseleiden rajalle. Menettelystä on etua etenkin matalan pikselitiheyden näyttölaitteilla, mutta erityisen tarkoilla näytöillä pikselit ovat muutenkin niin pieniä, ettei eroa huomaa paljain silmin.

5.4 Animoitu verkkosivusisältö

CSS3:n animaatio-ominaisuuksien myötä verkkosivujen käyttökokemusta on alettu parantaa animaatioiden avulla. Niiden käyttöön kannustavat muun muassa Jakob Nielsen kirjassaan *WWW-suunnittelu*, Googlen *Material Design*- ja Applen *Human Interface Guidelines* -suunnitteluohjeistot [46], [51], [52]. Niistä voidaan koostaa animaatioiden hyödyntämisen tarjoavan seuraavat edut:

- ”Siirtymien välisen jatkuvuuden osoittaminen” [46, s. 145]
- ”Siirtymän suunnan osoittaminen” [46, s. 145]
- ”Ajan myötä tapahtuvan muutoksen esittäminen” [46, s. 146]
- Suuremman informaatiomäärän välittäminen
- Elementtien välisten avaruudellisten ja hierarkkisten suhteiden havainnollistaminen
- Huomion kiinnittäminen joko tarpeellisiin paikkoihin tai pois tarpeettomista
- Palautteen antaminen käyttäjän syötteestä
- Käyttäjän toimen seuraamuksista vihjaaminen
- Viimeistelty vaikutelma

CSS tarjoaa nykyään erittäin käytännöllisen tavan soveltaa animaatioita verkkosivuilla, koska se mahdollistaa sekä animaatioiden määrittämisen että niiden liittämisen käyttäjän syötteisiin deklaratiivisesti. SVG:n ansiosta animaatioita voidaan

soveltaa listattujen etujen saavuttamiseksi myös graafiseen sisältöön. Näin kehittäjä voi hyödyntää huomattavasti rikkaampia animaatioita verkkosivujen käyttöliittymän toteuttamiseksi. Valitettavasti – kuten on havaittu luvussa 4 – tällä hetkellä SVG:n animoimiseen on tarjolla useita eri teknologioista, joista yksikään ei selkeästi ole paras ratkaisu. Erottavia tekijöitä ovat suorituskyky, selaintuki, paradigma ja monipuolisuus. Seuraavaksi käydään tarkemmin läpi nuo teknologiat: CSS, JavaScript ja SMIL. Osa esitetyistä tiedoista on löydöksiä luvusta 4 ja osa perustuu asianosaisten tekniikoiden standardeihin (CSS [53] ja SMIL [31]).

5.4.1 SVG:n animointi CSS:llä

HTML-elementtien animointi CSS:llä tuli mahdolliseksi CSS3:n myötä. Menetelmä on deklaratiivinen ja mahdollistaa laitteistokiihdytyksen, mutta CSS:ssä on kielenä tiettyjä puutteita SVG:n käsittelyssä. SVG:hen kuuluu valtava määrä erilaisia attribuutteja, joiden animoitavuudesta olisi hyötyä. CSS valitettavasti kattaa näistä vain osan, ja Microsoftin selaimet eivät tue minkään SVG:n oman attribuutin animoimista.

CSS:n animaatiomoduuli on tarkoitettu lähinnä yksittäisten elementtien animointiin, minkä vuoksi se ei tarjoa mahdollisuutta suurempien animaatiokokonaisuuksien hallinnointiin. Tukea animaatioiden ketujukselle tai muunlaiselle keskinäiselle ajastamiselle ei ole, vaan kehittäjän on tehtävä nämä manuaalisesti hyödyntäen muita teknologioita.

5.4.2 SVG:n animointi JavaScriptillä

Jos SVG:n DOM on paljastettuna kehittäjälle, sitä voi manipuloida luotettavasti ja monipuolisesti JavaScriptillä. Tätä seikkaa voidaan hyödyntää myös animaatioiden toteuttamiseksi SVG:n avulla. Menetelmä on kuitenkin matalan abstraktiotason ratkaisu, koska kehittäjän on huolehdittava jokaisen animoitavan objektin attribuuttien päivittämisestä jokaista animaation kehystä varten. DOM-manipulointi merkitsee myös merkittävää lisäkustannusta suorituskykynäkokulmasta.

Luonnollisesti myös JavaScript-tuki on vaatimus ja DOM:in paljastaminen poissulkee animaation toiminnan upotettuna `<img>`-elementin avulla. JavaScript ei ole deklaratiivinen kieli, joten animaaatioiden toteuttaminen sillä on vähemmän luontevaa kuin muilla SVG:n animointimenetelmillä. Toki kirjastojen käyttö abstrahoi tehtävää, mutta pohjimmiltaan skriptikielellä on aina koostettava jokainen animaation ruutu erikseen.

5.4.3 SVG:n animointi SMILillä

SMIL on ollut olemassa jo ennen SVG:tä, mutta sen toisessa versiossa keskityttiin parantamaan sen yhteistoimintaa muiden XML-pohjaisten tekniikoiden kanssa. Kyseinen versio mahdollisti myös SVG-elementtien animoimisen ja se julkaistiinkin samana vuonna SVG:n ensimmäisen version kanssa [31]. SVG on siis ollut animoitavissa aina, mutta tekniikan varsinainen hyödyntäminen on jäänyt häviävän vähäiseksi ainakin verkkosivukontekstissa [54]. SMIL:in katsotaan olevan hyvin rajatun sovellusalan ratkaisu ja sen tuki on työläs toteuttaa. Vaikka sillä on periaatteessa mahdollista animoida myös HTML-elementtejä, se ei kuitenkaan tuo merkittävästi lisää ominaisuuksia verrattuna CSS:ään ja JavaScriptiin nähden. HTML:n yhteydessä se olisi vain turha ylimääräinen kieli.

SVG:n animoimiseen SMIL on verraton, koska se on deklaratiivinen, laitteistokiihdytettävä ja sen avulla animoitu SVG on itsenäinen komponentti ilman ulkoisia riippuvuuksia. Se toimii animoituna `<img>`-elementissä GIF:in tavoin. Se myös tukee animaatioiden ketjutusta ja jopa kykenee huomioimaan käyttäjätapahtumia animaatioiden ohjauksessa.

Haittapuolena on täysin puuttuva tuki Microsoftin selaimissa. Firefoxissa ja Chromessa se on tuettu tällä hetkellä hyvin, mutta tulevaisuus on ainakin Chromen osalta epävarmaa: SMIL-animaatiot on jo asetettu vältettäviksi [54], mutta tämä päätös jäädytettiin korvaavien teknologioiden puuttumisen vuoksi ainakin siihen asti, kunnes seuraavaksi esiteltävä *Web Animations* -standardi valmistuu [55].

5.4.4 Web Animations

Animointitekniikoiden määrän voidaan katsoa levinneen vielä laajemmaksi, jos lasketaan erikseen kaikki JavaScriptillä toteutetut animointikirjastot. Tilanne on kehittynyt siihen pisteeseen, että W3C on aloittanut työskentelyn animointiteknologioiden yhtenäistämiseksi uudella standardilla. Se on nimeltään *Web Animations*, ja sen tavoite on määritellä selkeä ja yhtenäinen rajapinta niin HTML- kuin SVG-sisällönkin animoinnille [56]. Rajapinnan avulla selainvalmistajat voivat toteuttaa rajapintakutsut hyödyntäen laitteistokiihdytystä, mutta menetelmä on silti sidottu JavaScriptin paradigmaan ja -tukeen. Lähtökohtaisesti Web Animations ei siten myöskään toimine `<img>`-elementillä upotettuun grafiikkaan, vaikkakin tämän mahdollistava poikkeusjärjestely voisi olla teoriassa mahdollinen. Standardi on vielä siinä määrin kesken, että tämän tason yksityiskohtia on mahdollista ainoastaan spekuloida.

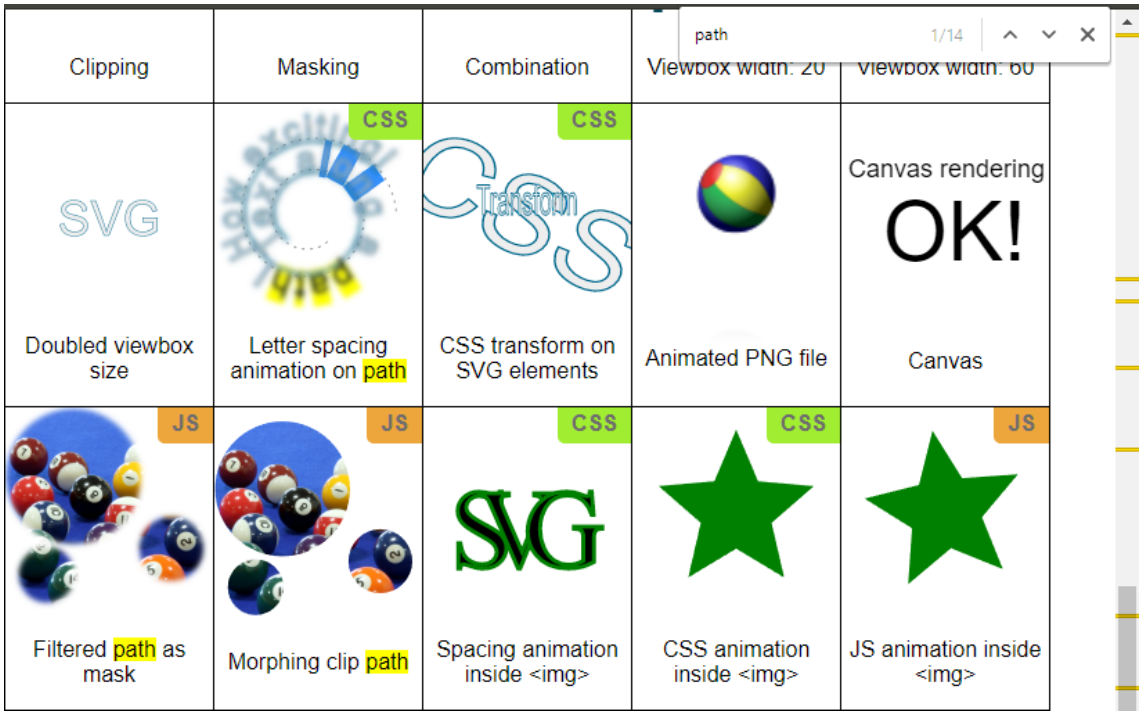
Standardin korkeamman tason versioihin tulee kuulumaan myös tuki animaatioiden ryhmittelyyn ja ketjuttamiseen, ja teoriassa jokainen animointimenetelmä voisi hyödyntää samaa taustatoteutusta Web Animations -moduulin kanssa [57]. Tämä voisi myös olla hyvä uutinen SMIL:in tuelle tulevaisuudessa.

5.5 Käyttäjäkokemus ja siirrettävyys

Vektorigrafiikan resoluutioriippumattomuus ja dynaaminen muokattavuus ovat ominaisuuksia, jotka tarjoavat rasterigrafiikkaan nähden uusia mahdollisuuksia myös käyttäjäkokemuksen ja siirrettävyyden parantamiseen. Teoksessa *WWW-suunnittelu* onkin kokonainen aliluku otsikolla ”Irti resoluutiosta.” Hän huomauttaa, että kehittäjä ei voi tietää, minkä kokoisella näytöllä sivua tullaan selaamaan, ja siksi sivu tulisi toteuttaa resoluutiosta riippumattomaksi. Esimerkkinä hän mainitsee, että erityisesti ikoneihin tulee kiinnittää huomiota niiden riittävän näkyvyyden ja erottuvuuden varmistamiseksi myös yli 100 DPI:n tarkkuuden näytöillä [46]. Tämä on erityisen tärkeää, mikäli ikoneihin tai muuhun graafiseen sisältöön kuuluu tekstiä. Nielsen huomauttaa myös, että tekstin upottaminen grafiikoihin ei ole muutenkaan suositeltavaa, koska ”se hidastaa siirtonopeutta ja vaikeuttaa käyttöliittymän kääntämistä muille kielille.” [46, s. 30] SVG:n tapauksessa ongelma ei ole niin merkittävä, sillä tekstidata säilyy tekstimuotoisena SVG:n merkintäkoodissa. Näin sen sisältö voidaan vaihtaa ohjelmallisesti selaimessa tai jopa palvelinpuolella. Toisaalta SVG:n `<switch>`-elementin ansiosta grafiikkaan voi valmiiksi sisällyttää tekstisisällön eri kielillä, joista selain valitsee oikean [26]. Lisäksi kaistankäyttö ei nykypäivänä ole merkittävä ongelma, ja vaikka olisikin, olemme todenneet SVG:n olevan tiedostokooltaan varsin kompaktia.

Teksti-informaation säilyminen SVG:ssä parantaa käytettävyyttä myös sikäli, että se säilyy käyttäjän maalattavana ja kopioitavana. Tämä pitää paikkansa vaikka teksti voi olla animoitua, suodattimella käsiteltyä tai asetettuna mutkikkaalle polulle. Myös selaimen etsintätoiminto voi löytää osumat grafiikan sisältäkin, kuten nähdään kuvassa 5.2.

Edelleen kirjassaan Nielsen kiinnittää huomiota verkkosivujen tulostamiseen. Tulostin on päätelaitteena näyttöihin nähden korkeampiresoluutioinen, mutta kuva-alaltaan kapeampi. Tämäkään ei ole ongelma SVG:n kanssa, vaan vektorigrafiikka tulostuu myös paperille aina terävänä siinä, missä rasterigrafiikka olisi suttuisimmillaan. Tosin kuva 5.3 osoittaa, että testikuva ’Ympyrä’ tulostuu paperille paremman näköisenä rasteripohjaisesta PNG-tiedostosta. Tämä kuitenkin johtuu kyseisen kuvan huomattavasta yksityiskohtaisuudesta, eikä tulostimen rasterointialgoritmi mukaudu näin poikkeavaan materiaaliin. Tulostusjäljen perusteella voi jopa olla, että

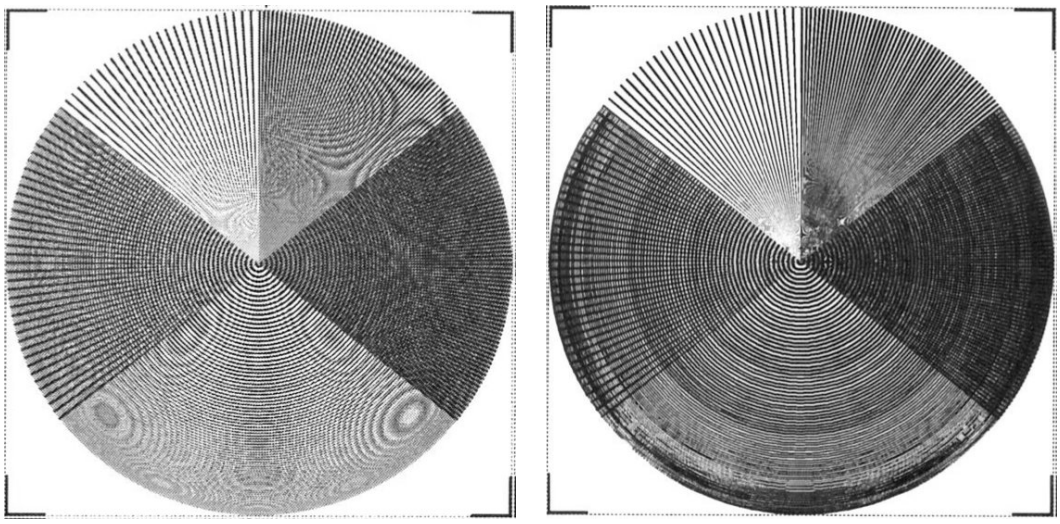


Kuva 5.2 *SVG-kuvan tekstisisältö säilyy etsittävänä ja maalattavana. Chromella jopa vierityspalkkiin ilmestyvät löydökset liikkuvat animaatioiden tahtiin.*

tulostimissa luotetaan pelkästään suureen pikselitiheyteen (tässä käytetty tulostin on pikselitiheydeltään 600 DPI), eikä niissä käytetä ylinäytteistystä ollenkaan. Vastaava havainto tehtiin kahdella eri laitteella.

Tämän perusteella tulostusmateriaalina on suotavaa käyttää mahdollisuuksien mukaan valmiiksi rasteroitua materiaalia, mikäli materiaali on poikkeuksellisen yksityiskohtaista ja tulostusjäljen johdonmukaisuus on tärkeää. Puutteita havaittiin vain kuvalla, joka luotiin varta vasten rasterointijäljen tutkimiseksi. Todellisuudessa tulostimien tarkkuus riittää tavanomaiselle kuvamateriaalille.

Siirrettävyyttä eri tyyppisille laitteille ja medioille edesauttaa CSS:n mediakyselyiden (*media query*) käyttö SVG:n tyylittämisessä, jolloin myös grafiikka voidaan toteuttaa mukautumaan erilaisille medioille. Tekstiselaimille ja ruudunlukijoille SVG:hen on mahdollista liittää metatietoa elementeillä `<title>`, `<desc>` ja `<role>` [58]. Näin julkaisija voi asettaa kuvaan valmiin sanallisen kuvauksen sen sisällöstä ja määrittää myös kuvan roolin. Roolin määrittelemällä julkaisija voi kertoa, onko kuva pelkästään koriste vai esittääkö se oleellista informaatiota. Kuvaajat ovatkin tyypillinen käyttökohde SVG:lle verkkosivuilla. Vielä pidemmälle vietynä XML:n selkotekstipohjaisuus voisi mahdollistaa SVG:n varsinaisen rakenteen ja sisällön analysoinnin siten, että ruudunlukija tulkitsee ja kuvailee suoraan SVG:n rakennetta [20].



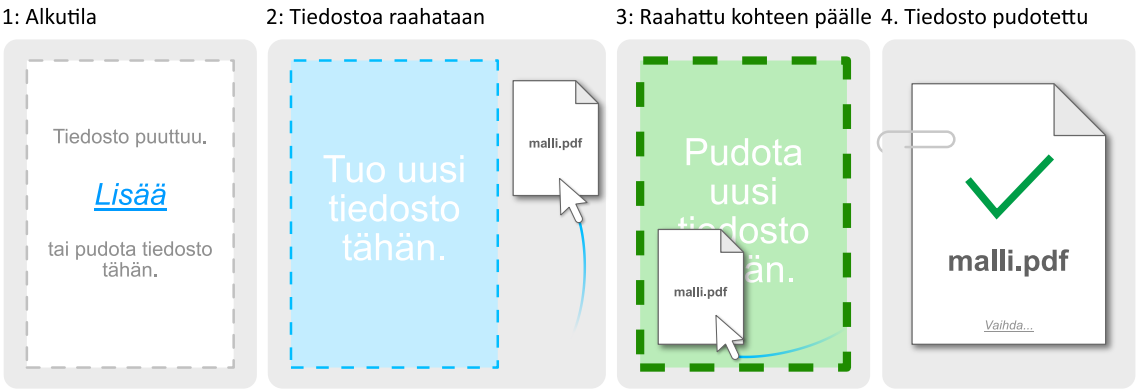
(a) Tuloste PNG-tiedostosta. (b) Tuloste SVG-tiedostosta.

Kuva 5.3 *Testikuvan 'Ympyrä' paperituloste PNG-tiedostosta ja SVG-tiedostosta.*

SVG:n toiminta JavaScriptin, CSS:n ja SMIL:in kanssa tekee siitä oivallisen teknologian mikrointeraktioiden toteuttamiseen. Mikrointeraktiot ovat pieniä prosesseja osana esimerkiksi sovelluksen tai laitteen suurempaa ominaisuutta. Yksi sähköpostisovelluksen mikrointeraktio on uuden, tyhjän sähköpostin luominen painiketta painamalla. Yksi älypuhelimen mikrointeraktio on sen näytön lukitseminen painamalla fyysistä nappia laitteen kyljessä. Kirjassaan *Microinteractions* Dan Saffer toteaa, että mikrointeraktioiden huolellinen suunnittelu saa tuotteen siedetystä rakastetuksi (*"From tolerated to treasured"*). Safferin mukaan hyvä mikrointeraktio huolehtii neljästä osasta: *laukaisin*, *säännöt*, *palaute* sekä *tilat ja silmukat*. [59]

Mikrointeraktion osista käyttöliittymässä näkyvimmät ovat laukaisin, palaute sekä tilat, ja näistä jokaisen toteuttamiseen SVG tarjoaa muita teknologioita paremmat mahdollisuudet. SVG:llä voidaan toteuttaa pelkkää HTML:ää monipuolisempia käyttöliittymäelementtejä, koska HTML ei sovellu puhtaan kuvamateriaalin kuvaamiseen. `<canvas>`-elementin avulla voidaan piirtää monipuolista graafista materiaalia dynaamisesti, mutta se soveltuu huonosti käyttäjäsyötteen vastaanottamiseen. Erillisten kuvien tai leikekarttojen (*sprite sheet*) käyttäminen tekee siirtymäanimaatioista hankalia tai käytännössä mahdottomia toteuttaa, jos elementin tulee mukautua erilaisiin käyttäjäsyötteisiin toisistaan riippumatta.

Kuva 5.4 havainnollistaa, miten mikrointeraktioita voi ehostaa SVG:n avulla. Siinä on esitetty mikrointeraktio, jossa käyttäjä raahaa liitetiedostoa ladattavaksi palvelimeen. Ensimmäisessä kohdassa näkyy pelkästään affordanssi, joka viestii käyttäjälle, että ladattavan tiedoston voi joko valita klikkaamalla tai raahaa-pudota



Kuva 5.4 *Esimerkki mikrointeraktion tehostamisesta SVG:n avulla.*

-menetelmällä. Kohdassa kaksi käyttäjä on aloittanut tiedoston raahaamisen, joten klikkausvaihtoehto voidaan poistaa näkyvistä ja raahauksen maali korostuu ohjaten käyttäjää. Korostus on mahdollista SVG:n `stroke`- ja `fill`-attribuutteja säätämällä joko CSS:llä tai JavaScriptillä. Kolmannessa kohdassa käyttäjä on raahannut tiedoston maalin päälle, muttei ole vielä pudottanut sitä. Tässä vaiheessa maali voisi korostua edelleen: väri voisi vaihtua hyväksyväksi vihreäksi, reuna voisi paksuuntua ja vaikka lähteä pyörimään. Korostus voidaan toteuttaa muuttamalla maalin `fill`-, `stroke-width`- sekä `stroke-dashoffset`-attribuutteja. Kun käyttäjä sitten päästää hiiren painikkeen irti, maali voisi korvaantua liitetiedostoa esittävällä grafiikalla. Tiedoston todellinen nimi voidaan asettaa näkyviin `<text>`-elementillä ja prosessin päättymistä voisi korostaa vielä lisäämällä paperiliitin animoidusti tiedoston päälle. Lopputulos näkyy kuvan kohdassa 4.

6. YHTEENVETO JA JOHTOPÄÄTÖKSET

Tämän diplomityön tarkoitus oli selvittää seuraavat asiat:

1. Kykeneekö SVG korvaamaan verkkosivukehityksessä vallalla olevat rasterigrafiikkaformaatit?
2. Mitä rasterigrafiikasta puuttuvia erityispiirteitä SVG:ssä on verkkosivutekniikkana?

Työtä pohjustettiin kirjallisuuskatsauksella siitä, miten SVG soveltuu verkkosivukäyttöön. Katsauksella löydettiin seuraavat väittämät SVG:stä:

1. SVG on huonosti tuettua web-käyttöä ajatellen [7].
2. SVG on tiedostokooltaan suurempaa, kuin rasteriformaatit [11].
3. SVG on `<canvas>`-elementtiä heikompi suorituskyvyltään [10].
4. SVG on kilpailijoitaan huonommin soveltuva animaatioiden tekemiseen [12].
5. SVG on liitännäispohjainen, laitteisto- tai käyttöjärjestelmäsidonnainen [12].
6. SVG on työlästä tuottaa [11].

Työssä vertailtiin SVG:tä rasteripohjaisiin kuvaformaatteihin ja -tekniikoihin. Tutkitut formaatit ja tekniikat olivat JPG, PNG, GIF ja `<canvas>`. Soveltuvuuden mittareina olivat materiaalin visuaalinen laatu, tiedostokoko, selaintuki ja suorituskykyvaatimus. Vertailu staattisen grafiikan (eli formaatit JPG ja PNG) kanssa suoritettiin luomalla neljästä eri testikuvasta 16 eri versiota vaihdellen versioiden kokoa ja pakkausasetuksia. Eri versioiden visuaalista laatua arvioitiin silmämääräisesti, niiden tiedostokoot taulukoitiin ja tiedostokoon vastaavuutta versioiden laatuun analysoitiin sanallisesti. SVG:n soveltuvuus animaatioiden toteuttamiseksi arvioitiin ensisijaisesti selaintuen ja teknologioiden monipuolisuuden kannalta verrattuna GIF:iin. Lopulta dynaaminen grafiikan generointi tutkittiin JavaScriptillä ohjatun

SVG:n ja HTML5:n uuden `<canvas>`-elementin välillä. Vertailussa huomioitiin ensisijaisesti suorituskyky ja toteutuksen helppous.

Tässä luvussa on lyhyesti koottuna työn löydökset osa-alueittain ja analysointia siitä, poikkeavatko löydökset taustatutkimuksesta löydetyistä väittämistä. Lisäksi pohditaan, miten hyvin työ onnistui vastaamaan lähtökohtana olleisiin tutkimuskysymyksiin. Luvussa pohditaan myös, mihin SVG ja sovellusalue on kehittymässä lähitulevaisuudessa. Lopuksi eritellään, miten tämän työn tutkimusta voitaisiin syventää.

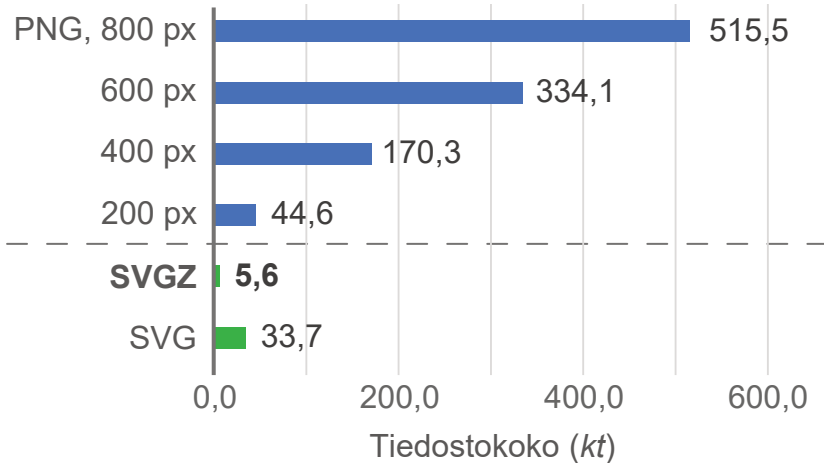
Heti alkuun voidaan kumota kaksi väitettä. Ensinnäkin kaikki tämän työn testit oli suoritettavissa selainympäristöissä eri alustoilla ilman asennettavia liitännäisiä, joten väite 5 ei pidä paikkaansa. Toisekseen aliluvussa 5.3 esitettiin rasteroinnin olevan tarpeeton toimenpide piirrosgrafiikan julkaisemisessa, joten vektoripohjaisen SVG:n julkaiseminen on teknisesti rasterigrafiikkaa yksinkertaisempaa. Tämä kumoaa väitteen 6.

6.1 SVG:n soveltuvuus staattisen grafiikan esittämiseen

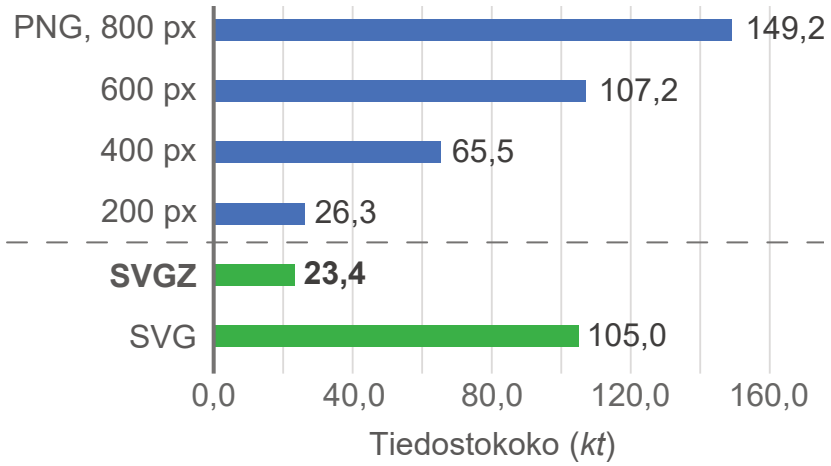
Rasterigrafiikka on kiinteäresoluutioista [1]. Resoluutiosidonnaisuuden haitat kävivät selkeästi ilmi luvussa 4.1, jossa havaittiin, että kaikenlainen rasterigrafiikan skaalaus heikentää kuvan yksityiskohtaisuutta. Toisaalta aliluvussa 4.1.1 todettiin, että poikkeuksellisen yksityiskohtainen SVG-kuva hahmontuu eri selainten välillä epäsäännönmukaisesti, mutta aliluvun 4.1.2 perusteella ongelma ei koske tavanomaisempia kuvia. Alkujaan vektorigrafiikkana tallennetuissa kuvissa kuvanlaatu on kaikkiaan parempi SVG-formaatissa.

Kuva 6.1 osoittaa, että SVG-formaatti on tiedostokooltaan joko samaa luokkaa tai pienempää, kuin häviötön kilpaileva rasteriformaatti PNG. Molemmissa piirrosgrafiikan testikuvissa esitystarkkuudessa (600 pikseliä) oleva PNG oli suurempi tiedosto, kuin SVG. Kun SVG pakattiin SVGZ-tiedostoksi, tiedostokoko pieneni PNG:tä selkeästi pienemmäksi. Tämä kumoaa taustatutkimuksesta koostetun väitteen 2, jonka mukaan SVG veisi paljon tilaa.

Piirrosgrafiikan tapauksessa lopputulos on selvä SVG:n eduksi. Valokuvamateriaali sen sijaan on ongelmallista. Aliluvussa 4.2 todettiin, valokuvamateriaalin esittäminen vektorigrafiikkana ei tee materiaalista skaalautuvaa, mutta vie huomattavasti alkuperäistä enemmän tallennustilaa. Skaalatun rasterigrafiikan interpolointi tuottaa visuaalisesti laadukkaamman lopputuloksen. Tämän seurauksena valokuvamateriaali on järkevää tallentaa ja toimittaa JPG-formaatissa, koska sen pakkausalgo-



(a) Testikuva 'Ympyrä'

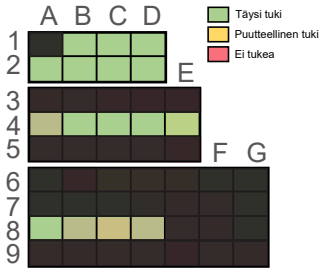


(b) Testikuva 'Suomi'

Kuva 6.1 *Piirrosgrafiikan tiedostokokovertailu testikuvista 'Ympyrä' ja 'Suomi'. Kaaviot on siistitty kuvista 4.2 ja 4.9. Molemmissa tapauksissa pienin tiedosto on SVGZ.*

ritmi on tehokas ja maltillisilla pakkaussuhteilla heikentää valokuvamateriaalin näkyvää laatua vain vähän. Jos on tarpeen täydentää rasterigrafiikkaa piirtografiikalla, kannattaa hyödyntää SVG:n `<image>`-elementtiä rasterigrafiikan liittämiseen SVG:n mukaan. Näin piirtomateriaali ei kärsi JPG:n häviöllisestä pakkausalgoritmista eikä skaalauksesta ja sen semanttinen informaatio säilyy, kuten todettiin aliluvussa 5.1.

Selaintuki staattiselle SVG:lle paljastui kuvassa 6.2, jossa asianosaiset rivit 1 ja 2 osoittavat selaintuen olevan kattava kaikilla moderneilla selaimilla. Kuvasta 4.14 nähdään, että tuki staattisen SVG:n esittämiseen ylettyy Internet Explorerin versioon 9 asti. Kirjallisuudesta kävi kuitenkin ilmi, että Microsoft on lakkauttanut tuen vanhemmille IE:n versioille ja niiden markkinaosuuskin maailmanlaajuisesta selauskäytöstä on vain 0,65 %. Selaintuki on siis varsin hyvällä tasolla ja väite 1 on todettu vääräksi.



Kuva 6.2 *Kuvan 4.13 Yhdiste-taulukon SVG:n esittämistukea ja JavaScriptillä animoinnin tukea esittävät solut. Kuvan selite löytyy liitteestä C*

Staattisen grafiikan tutkimuksen lopputulos on, että SVG sopii PNG:tä paremmin piirrosgrafiikan tallentamiseen ja esittämiseen tiedostokoon, visuaalisen laadun ja käytännöllisyyden kannalta. Selaintuen ei todettu olevan rajoite staattisen grafiikan tapauksessa. Valokuvien tallentamiseen JPG on edelleen paras formaatti, koska vektorigrafiikalla ei voi saavuttaa vastaavaa visuaalista laatua ja vektoroitu materiaali vie huomattavasti enemmän muistia. Jos graafisessa materiaalissa on tarpeen yhdistää piirrosgrafiikkaa valokuvamateriaaliin, kannattaa hyödyntää SVG:n `<image>`-elementtiä, jolloin sekä rasteri- että vektorigrafiikan edut ovat hyödynnettävissä.

Staattisen grafiikan tutkimiseen käytetty testiaineisto oli melko rajallista, ja tämän osa-alueen löydökset eivät ole yleistettävissä kaikkeen graafiseen materiaaliin. Erityisesti tiedostokoon tutkimiseen olisi ollut hyvä käyttää laajempaa testiaineistoa ja koostaa tuloksista tunnusluvut. Löydökset ovat kuitenkin suuntaa antavia ja havaintojen perusteella voidaan jo tehdä valistuneita olettamuksia siitä, miten hyvin SVG sopisi toteutustekniikaksi verkkosivuprojekteihin. Tieteellisen faktan tuottamiseksi testimateriaalin laajuutta ja monipuolisuutta tulisi kasvattaa oleellisesti.

6.2 Animoitu piirrosgrafiikka

Animaatioihin käytettyjen GIF:in ja APNG:n kohdalla tilanne ei ole aivan yhtä selkeä, kuin staattisen grafiikan kanssa. Staattisen materiaalin tallentamiseen PNG on joka suhteessa GIF:iä parempi formaatti (ja SVG puolestaan PNG:tä parempi aiemman mukaan), mutta animoidun sisällön kanssa haittoja alkaa ilmetä enemmän. HTML-elementtien animointiin on nykyään kattava tuki CSS:llä ja JavaScriptillä, mutta etenkin CSS:n soveltuvuus SVG-sisällön animoimiseen on puutteellinen selaintuen ja kielen ominaisuuksien vuoksi. Näin kävi ilmi aliluvuista 4.4 ja 5.4. Selaintukea osoittavan testitaulukon animointia koskevat rivit on esitetty kuvassa 6.3, joka osoittaa JavaScriptin olevan luotettavin tapa animoida SVG:tä. JavaScript ei valitettavasti mahdollista animoidun sisällön upottamista GIF:in tavoin yksinkertaisesti `<img>`-elementtiin, vaan koko SVG-merkintä on upotettava HTML-merkinnän

mukaan.

Kuva 6.3 *Kuvan 4.13 Yhdiste-taulukon SVG:n animoinnin tukea esittävät solut. JavaScriptin rivi on lähes täysin vihreä, eli sillä toteutetut animaatiot toimivat hyvin eri selaimilla. Kuvan selite löytyy liitteestä C*

SMIL sitä vastoin on tarkoitettu nimenomaan XML-pohjaisten dokumenttien animaatioiden toteuttamiseen ja kykenee teknisesti tarjoamaan GIF:iä huomattavasti käytännöllisemmän tavan toteuttaa animoitua sisältöä verkkosivuilla. Sen lisäksi, että itse kuvamateriaali on semanttisesti kuvailtua vektorigrafiikkaa, myös animaatiot on kuvattu semanttisella tavalla. Tämä mahdollistaa helpomman ylläpidettävyyden ja sisällön toiston optimoinnin laitteistokiihdytyksen avulla selaimessa. Haittapuolena on, ettei Microsoft tue SMIL:iä ollenkaan selaimissaan ja Chromesta tuki tullee poistumaan tulevaisuudessa [54]. Web Animations saattaa kuitenkin kehittyä korvaavaksi teknologiaksi, jonka selaintuki olisi myös todennäköisesti kattava [56].

Johtopäätöksenä voidaan todeta, että rivinsisäisenä `<svg>`:n animoiminen JavaScriptillä on GIF:iä parempi ratkaisu monipuolisuudeltaan ja teknisiltä ominaisuuksiltaan. Jos Microsoftin selaimia ei tarvitse tukea, SMIL on erittäin suositeltava tapa sen monipuolisuuden ja yksinkertaisuuden ansiosta. Jos animaation on toimittava kaikilla selaimilla `<img>`-elementissä, GIF on ainoa vaihtoehto. Väitteeseen 4 ei siis löytynyt selvää yleispätevää vastausta.

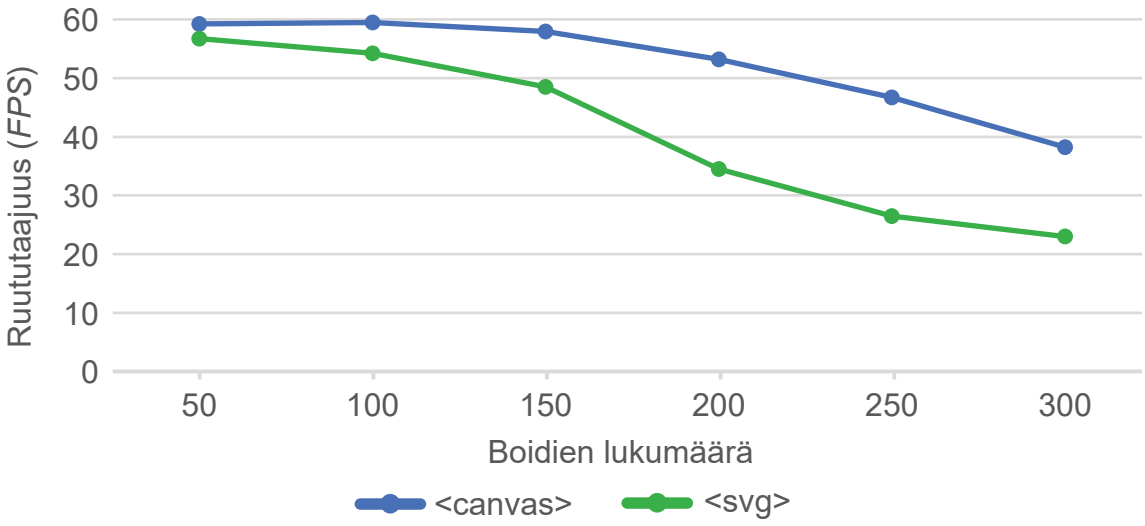
Jo tässä työssä käytetyn testiaineiston perusteella johtopäätös vaihtelevasta selaintuesta on hyvin perusteltu. Siltä osin työ oli onnistunut. Tutkimusta JavaScriptin toimintavarmuudesta voisi kuitenkin täydentää vielä erilaisilla testisovelluksilla, jotta virheen mahdollisuus pienenisi.

6.3 Dynaaminen grafiikka ja suorituskyky

Testattaessa selaintukea aliluvussa 3.2 kävi ilmi, että SVG:n suorituskykyvaatimukset voivat nousta liian suuriksi, mikäli sisältöön kuuluu paljon animoituja objekteja. Vertailutaulukoiden 3.3, 3.4 ja 3.5 avaaminen älypuhelimen selaimella pudottaa näkymän ruudunpäivitystaajuuden tasolle, joka ei ole käyttäjäkokemuksen näkökulmasta millään muotoa hyväksyttävä. Suorituskyky muodostui ongelmaksi myös testattaessa Boids-algoritmia työpöytätietokoneilla, kun boidien lukumäärää nostettiin

tarpeeksi. Kuvaan 6.4 on koottu keskiarvot Chromen ja Firefoxin ruudunpäivitystaajuuksista molemmilla testilaitteilla taulukoista 4.1 ja 4.2. Kuvasta nähdään, että päivitystaajuus tippui alle 40 FPS rajan, kun boideja piirrettiin `<canvas>`-elementillä 300 kappaletta. Vastaava lukema SVG:tä hyödyntäen oli 200 kappaletta.

Tyypillisessä verkkosivukehityksessä näin suuri määrä liikkuvia asioita ei ole tyypillistä. Poikkeuksen voivat muodostaa esimerkiksi selainpelit. Jos kuitenkin puhutaan tavallisen verkkosivun ehostukseen käytetyistä animaatioista, jotka käynnistyvät käyttäjän syötteestä yksi kerrallaan, niin suorituskyky ei muodostune ongelmaksi moderneilla älypuhelimilla puhumattakaan työpöytäselaimista. Väite 3 pitää siis periaatteessa paikkansa, mutta on merkitykseltään vähäinen.



Kuva 6.4 *Mitattujen ruututaajuuksien keskiarvot kangaselementille ja SVG:lle.*

Tässä valossa suorituskyky ei ole peruste käyttämättä SVG:tä mikäli kyseessä ei ole huomattavan raskas sovelluskohde. Esimerkiksi kuvaajien toteuttamiseen SVG:n suorituskyky on hyvin riittävä ja se tarjoaa piirtoalustaa monipuolisemman rajapinnan suunnittelijoille ja kehittäjille. SVG:tä käytettäessä tyylitys voidaan hoitaa CSS:llä ja kuvaajan elementteihin voidaan kytkeä tapahtumankäsittelijät yhtä kätevästi kuin HTML-elementteihinkin. Lisäksi SVG on luontaisesti responsiivista ja se voidaan haluttaessa generoida valmiiksi palvelimella.

Suorituskykyä tutkittiin tässä työssä vain yhdellä testisovelluksella. Dynaamisen grafiikan tuottamiseen hyödynnettiin pelkästään JavaScriptiä, vaikka osa animaatioista olisi ollut mahdollista toteuttaa muillakin tekniikoilla. Tällä olisi todennäköisesti ollut vaikutus suorituskykymittausten lopputuloksiin. Myös vuorovaikutteisuuden lisääminen testisovelluksen ominaisuuksiin olisi saattanut tehdä toteutuksen monimutkaisuuden eron tekniikoiden välillä merkittävämmäksi.

6.4 Johtopäätökset

Lopputulokset vertailuista rasterigrafiikan kanssa on koostettu taulukkoon 6.1.

Taulukko 6.1 *SVG:n soveltuvuus tyypillisten rasterigrafiikkaformaattien korvaajaksi.*

Formaatti	Käyttökohde	Lopputulos
PNG	Digitaalinen piirrosgrafiikka	SVG soveltuu oleellisesti rasterigrafiikkaa paremmin.
GIF	Animoitu piirrosgrafiikka	SVG soveltuu tietyin ehdoin rasterigrafiikkaa paremmin.
JPG	Valokuvamateriaali	SVG ei sovellu; JPG on laadukkaamman näköistä ja kompaktimpaa. JPG-kuvaa voidaan kuitenkin käyttää SVG-dokumentin sisällä.
`<canvas>`	Dynaaminen grafiikka	SVG soveltuu interaktiiviseen ja monimutkaisuudeltaan kohtalaiseen käyttöön rasterigrafiikkaa paremmin.

SVG on hyvin käytännöllinen formaatti verkkosivukehityksessä, mutta dokumenttiluonteensa vuoksi siihen ei voi täysin suhtautua pelkkänä grafiikkaformaattina. SVG:n mahdollisuuksia täydentävät erityisesti sen yhteensopivuus CSS:n ja JavaScriptin kanssa. Kääntöpuolena SVG-dokumentti voi sisältää ajettavaa JavaScriptiä, joka saattaa altistaa verkkosivun tietoturvauhille. Riski on olemassa tosin vain, mikäli sivustolla esitetään käyttäjiltä peräisin olevaa materiaalia.

6.5 Tulevaisuus

Kaikilla grafiikkalajien osa-alueilla on paraikaa kehitteillä teknologioita, jotka saattavat muuttaa tämän työn lopputuloksia. Rasterigrafiikkarintamallakin tapahtuu kehitystä edelleen. Google kehittää verkkosivukäyttöön uutta rasterigrafiikkaformaattia, joka yhdistäisi muiden grafiikkaformaattien edut yhdeksi monipuoliseksi formaatiksi. Formaatti on nimetty *WebP*:ksi. WebP käyttää muista tutkituista rasteriformaateista tehokkaampaa pakkausalgoritmia, ja sen kohdalla on mahdollista valita, käytetäänkö häviöllistä vai häviötöntä pakkausta. Se tukee myös animaatioita ja läpinäkyvyyttä toisin, kuin JPG. [12]

W3C puolestaan on kehittämässä SVG:n seuraavaa versiota, SVG 2:a. Sen version on tarkoitus parantaa SVG:n siirrettävyyttä entisestään lisäämällä enemmän sisältöä kuvailevia elementtejä ja attribuutteja. Pyrkimyksenä on myös parantaa SVG:n toimintaa muiden teknologioiden ja merkkauskielten kanssa [60].

Liikkuvan grafiikan puolella SMIL:in tulevaisuus näyttää epävarmalta. Microsoft on tehnyt selväksi, ettei se aio tukea SMIL:iä selaimissaan. Google on päättänyt julistaa SVG:n animoinnin SMILiä hyödyntäen vältettäväksi tekniikaksi [54], joten kyseinen tuki on tulevaisuudessa todennäköisesti poistumassa Chromestakin. Google kuitenkin jäädytti julistuksen myöhemmin kritiikin vuoksi todeten, että korvaavat teknologiat eivät ole vielä tarpeeksi kehittyneitä [55]. Google kehottaa käyttämään korvaavina teknologioina JavaScriptiä, CSS:ää ja Web animaatioita, vaikka tällä hetkellä ainoastaan SMIL mahdollistaa animaatioiden rakenteen ja synkronoinnin deklaratiivisesti. Esitetyt korvaavat teknologiat lisäävät riippuvuuksia ja vaativat JavaScriptin tukea.

Edge ei tule tukemaan myöskään CSS-animaatioita SVG-elementeille [61]. Tämä jättää JavaScriptin ainoaksi selaintueltaan ja ominaisuuksiltaan kattavaksi tavaksi animoida SVG:tä.

6.6 Jatkotutkimus

Testikuva 3.1(a) paljasti, että kaikkien testattujen selainten sekä PNG-versioiden konvertointiin käytetyn Affinity Designerin rasterointialgoritmit ovat alttiita laskostumiselle. Tämä viittaa siihen, että ne perustuvat *näytteistykseen*. Näytteistyksessä rasteroitavasta grafiikasta tutkitaan diskreettejä ja äärettömän pieniä pisteitä, joiden kohdalta kunkin pikselin väriarvo määräytyy. Menetelmästä kehittyneempi versio on ylinäytteistys, jossa yhtä pikseliä kohden lasketaan useampi arvo, joiden keskiarvo määrää lopullisen pikselin värin. Teoriassa laskostumisen eliminoiva menetelmä olisi laskea analyyttisesti, kuinka suuren osan minkäkin värinen objekti kattaa pikselin pinta-alasta.

Suorituskykymittausta voisi laajentaa lisäämällä testisovellusten määrää ja täydentämällä sitä myös muilla animointitekniikoilla. Testattu JavaScript-pohjainen sovellus vastaa esimerkiksi selainpeleihin liittyviä käyttötapauksia, mutta SMIL- ja CSS-pohjaisissa animaatioissa suorituskykyprofiili olisi todennäköisesti hyvin erilainen SVG:n hyväksi. Tässä työssä suoritettu testi ei myöskään hyödyntänyt SVG:n suorituskykyattribuutteja. Niillä voidaan vihjata esittävälle sovelluukselle, halutaanko hahmonnuksessa suosia suorituskykyä vai visuaalista laatua. Merkitys jäisi kuitenkin vähäiseksi JavaScriptin rinnalla, mutta deklaratiivisten animointimenetelmien kanssa vaikutus saattaa olla merkitsevä. Samalla voisi tutkia myös interaktiivisen grafiikan toteutuksen monimutkaisuutta SVG:n ja `<canvas>`in välillä.

Tällä hetkellä käytössä olevat rasterigrafiikan vektorointialgoritmit perustuvat yksinkertaiseen kvantisointiin ja reunantunnistukseen. Jatkuvaluontoisen grafiikan esit-

tämiseen mentelmä soveltuu kehnosti, koska liukuvien värienvaihteluiden mallintaminen näin vaatii valtavasti polkuja ja siten tulos vie paljon muistia. Tämä kävi ilmi kuvasta 4.12. Lisäksi täysin terävät polut aiheuttavat sen, että lopputulos nauhoittuu, mikä käy ilmi kuvasta 4.10(b). Jos algoritmit hyödyntäisivät esimerkiksi SVG:n pehmennyssuodinta ja väriliukuja vektoroinnissa, niin tulos voisi olla parempi ja kompaktimpi. Alkuperäistä rasterikuvaa yksityiskohtaisempi vektoroinnin tulos ei voi olla, mutta vektoroidun kuvan terävät reunat pysyisivät terävinä ja pehmeät pehmeinä.

Tässä työssä viitattiin SVG:n tietoturvaa käsittelevään tutkimukseen, joka julkaistiin vuonna 2011 [48]. Nykytilanne olisi syytä kartoittaa tuoreella tutkimuksella, sillä – kuten luvusta 2 opimmme – Flash toimii historiallisena esimerkkinä tietoturvan merkityksestä teknologian suosioon.

LÄHTEET

[1] N. Chapman ja J. Chapman, *Digital multimedia*, English, 3rd. Chichester: John Wiley & Sons, 2009, ISBN: 9780470512166.

[2] J. D. Eisenberg, *SVG Essentials*, English. Sebastopol: O'Reilly & Associates, Inc., 2002, ISBN: 0596002238.

[3] *statcounter*. Saatavissa: http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet#monthly-200901-201710 (viitattu 08. 11. 2017).

[4] *statcounter*. Saatavissa: http://gs.statcounter.com/browser-market-share/desktop/worldwide#monthly-200901-201710 (viitattu 08. 11. 2017).

[5] A. Deveria, *Can I use SVG?* Saatavissa: https://caniuse.com/#search=svg/ (viitattu 03. 11. 2017).

[6] *HTML5: A vocabulary and associated APIs for HTML and XHTML*, W3C, 2014. Saatavissa: http://www.w3.org/TR/html5 (viitattu 14. 04. 2017).

[7] J. K. Korpela, *HTML5 - Uudet ominaisuudet*, Finnish. Porvoo: Docendo, 2011, ISBN: 9510381373.

[8] J. K. Korpela ja T. Linjama, *Web suunnittelu*, Finnish. Porvoo: Docendo, 2005, ISBN: 9518462437.

[9] B. Lawson ja R. Sharp, *Introducing HTML5*, English. Berkeley, California: New Riders, 2011, ISBN: 0321687299.

[10] A. Mäkinen, "Visualization of medical data using web technologies", Diplomityö, Tampereen teknillinen yliopisto, 2015.

[11] T. Taipale, "Mukautuvien käyttöliittymien mentelmät web-sovelluksissa", Diplomityö, Tampereen teknillinen yliopisto, 2015.

[12] S. Thapar, S. K. Chowdhary ja D. Bahri, "Compression and Optimization of Web-Contents", teoksessa *Proceedings of 3rd International Conference on Advanced Computing, Networking and Informatics: ICACNI 2015, Volume 2*, A. Nagar, D. P. Mohapatra ja N. Chaki, toim., New Delhi: Springer India, 2016, s. 495–505, ISBN: 978-81-322-2529-4. DOI: 10.1007/978-81-322-2529-4_52. Saatavissa: http://dx.doi.org/10.1007/978-81-322-2529-4_52.

[13] V. Keränen, N. Lamberg ja J. Penttinen, *Web-julkaiseminen & multimedia*. Porvoo: Docendo, 2006, ISBN: 9518462844.

[14] J. K. Korpela ja T. Linjama, *XHTML-käsikirja*, Finnish. Porvoo: Docendo, 2004, ISBN: 9518467382.

[15] S. Matsumoto, *Electronic display devices*, English. Chichester: Wiley, 1990, ISBN: 9780471922186;

[16] G. Reid, *PostScript language : program design*. Addison-Wesley, 1988.

[17] S. Truesdell, *Retina.js*. Saatavissa: http://imulus.github.io/retinajs/ (viitattu 12. 11. 2017).

[18] Adobe Systems Software Ireland Ltd., *Flash Player*, English. Saatavissa: http://www.adobe.com/fi/products/flashplayer.html (viitattu 29. 10. 2017).

[19] C. Musciano ja B. Kennedy, *HTML: The Definitive Guide*, English. Sebastopol: O'Reilly & Associates, Inc., 1996, ISBN: 1565921755.

[20] S. J. Vaughan-Nichols, "Will vector graphics finally make it on the Web?", English, *Computer*, vol. 34, nro 12, s. 22–24, joulukuu 2001, ISSN: 0018-9162. DOI: 10.1109/2.970549.

[21] J. Yu, *SVG in IE9 Roadmap*, 2010. Saatavissa: https://blogs.msdn.microsoft.com/ie/2010/03/18/svg-in-ie9-roadmap/ (viitattu 11. 11. 2017).

[22] S. Jobs, *Thoughts on Flash*, huhtikuu 2010. Saatavissa: http://www.apple.com/hotnews/thoughts-on-flash/ (viitattu 14. 12. 2016).

[23] Adobe Corporate Communications, *Flash, HTML5 and open web standards*, marraskuu 2015. Saatavissa: https://blogs.adobe.com/conversations/2015/11/flash-html5-and-open-web-standards.html/ (viitattu 14. 12. 2016).

[24] ——, *Flash & the future of interactive content*, heinäkuu 2017. Saatavissa: https://blogs.adobe.com/conversations/2017/07/adobe-flash-update.html (viitattu 30. 08. 2017).

[25] *About SVG*, W3C, 2004. Saatavissa: https://www.w3.org/Graphics/SVG/About.html (viitattu 04. 01. 2017).

[26] *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*, W3C, 2011. Saatavissa: http://www.w3.org/TR/SVG (viitattu 25. 10. 2016).

[27] *Extensible Markup Language (XML) 1.1 (Second Edition)*, W3C, 2006. Saatavissa: http://www.w3.org/TR/2006/REC-xml11-20060816/ (viitattu 11. 03. 2017).

[28] *HTML 4.01 Specification*, W3C, 1999. Saatavissa: https://www.w3.org/TR/html401 (viitattu 23. 08. 2017).

[29] F. Boumphrey, C. Greer, D. Raggett, J. Raggett, S. Schnitzenbaumer, T. Wugofski ja K. Kolehmainen, *Inside XHTML: Ohjelmoijan käsikirja*. Helsinki: Edita, IT Press, 2000, ISBN: 9789518262285.

[30] Stuart Parmenter and Vladimir Vukicevic and Andrew Smith, *APNG specification*. Saatavissa: https://wiki.mozilla.org/APNG_Specification (viitattu 13. 11. 2017).

[31] *Synchronized Multimedia Integration Language (SMIL 3.0)*, W3C, 2008. Saatavissa: https://www.w3.org/TR/SMIL (viitattu 25. 08. 2017).

[32] D. Duce, I. Herman ja B. Hopgood, "Web 2D Graphics File Formats.", *Computer Graphics Forum*, vol. 21, nro 1, s. 43–64, 2002, ISSN: 01677055. Saatavissa: http://search.ebscohost.com.libproxy.tut.fi/login.aspx?direct=true&db=bth&AN=6183393&site=ehost-live&scope=site.

[33] Z.-R. Peng ja C. Zhang, "The roles of geography markup language (GML), scalable vector graphics (SVG), and Web feature service (WFS) specifications in the development of Internet geographic information systems (GIS)", *Journal of Geographical Systems*, vol. 6, nro 2, s. 95–116, kesäkuu 2004, ISSN: 1435-5949. DOI: 10.1007/s10109-004-0129-0. Saatavissa: https://doi.org/10.1007/s10109-004-0129-0.

[34] A. Neumann ja A. M. Winter, "Time for SVG—towards high quality interactive web-maps", *International Cartographic Association*, 2001.

[35] M. N. Boulos, C. Russell ja M. Smith, "Web GIS in practice II: interactive SVG maps of diagnoses of sexually transmitted diseases by Primary Care Trust in London, 1997 – 2003", *International Journal of Health Geographics*, vol. 4, nro 1, s. 4, tammikuu 2005, ISSN: 1476-072X. DOI: 10.1186/1476-072X-4-4. Saatavissa: https://doi.org/10.1186/1476-072X-4-4.

[36] A. Cecconi ja M. Galanda, "Adaptive Zooming in Web Cartography", *Computer Graphics Forum*, vol. 21, nro 4, s. 787–799, 2002, ISSN: 1467-8659. DOI: 10.1111/1467-8659.00636. Saatavissa: http://dx.doi.org/10.1111/1467-8659.00636.

[37] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh ja J. P. Singh, "Who Killed My Battery?: Analyzing Mobile Browser Energy Consumption", teoksessa *Proceedings of the 21st International Conference on World Wide Web*, sarja WWW '12, Lyon, France: ACM, 2012, s. 41–50, ISBN: 9781450312295. DOI: 10.1145/2187836.2187843. Saatavissa: http://doi.acm.org/10.1145/2187836.2187843.

[38] N. Asuni ja A. Giachetti, "TESTIMAGES: A Large Data Archive For Display and Algorithm Testing", *Journal of Graphics Tools*, vol. 17, nro 4, s. 113–125, 2013. DOI: 10.1080/2165347X.2015.1024298. eprint: http://dx.doi.org/10.1080/2165347X.2015.1024298. Saatavissa: http://dx.doi.org/10.1080/2165347X.2015.1024298.

[39] ——, "TESTIMAGES: a Large-scale Archive for Testing Visual Devices and Basic Image Processing Algorithms.", teoksessa *Eurographics Italian Chapter Conference*, 2014, s. 63–70.

[40] *Väestötiheys, Suomi, 2016*, huhtikuu 2016. Saatavissa: https://commons.wikimedia.org/wiki/File:Suomi.v%C3%A4est%C3%B6tiheys.2016.svg (viitattu 21. 08. 2017).

[41] A. Deveria, *Can I use?* Saatavissa: https://caniuse.com (viitattu 13. 11. 2017).

[42] ——, *Can I use filter?* Saatavissa: https://caniuse.com/#search=filter/ (viitattu 29. 10. 2017).

[43] Adobe Web Platform, *Snap.svg*. Saatavissa: http://snapsvg.io (viitattu 24. 08. 2017).

[44] C. W. Reynolds, "Flocks, Herds and Schools: A Distributed Behavioral Model", teoksessa *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, sarja SIGGRAPH '87, New York, NY, USA: ACM, 1987, s. 25–34, ISBN: 0-89791-227-6. DOI: 10.1145/37401.37406. Saatavissa: http://doi.acm.org/10.1145/37401.37406.

[45] *Mobile Web Application Best Practices*, W3C, 2010. Saatavissa: http://www.w3.org/TR/2010/REC-mwabp-20101214 (viitattu 01. 04. 2017).

[46] J. Nielsen, *WWW suunnittelu*, Finnish. Jyväskylä: IT Press, 2000, ISBN: 9518262039.

[47] Microsoft, *Support for older versions of Internet Explorer ended*, 2016. Saatavissa: https://www.microsoft.com/en-us/windowsforbusiness/end-of-ie-support (viitattu 05. 11. 2017).

[48] M. Heiderich, T. Frosch, M. Jensen ja T. Holz, "Crouching tiger - hidden payload: security risks of scalable vectors graphics", teoksessa *Proceedings of the 18th ACM conference on Computer and communications security*, ACM, 2011, s. 239–250.

[49] *WordPress*. Saatavissa: https://wordpress.org (viitattu 14. 11. 2017).

[50] D. Doyle, *svg-sanitizer*. Saatavissa: https://github.com/darylldoyle/svg-sanitizer (viitattu 14. 11. 2017).

[51] Apple Inc., *Human Interface Guidelines*. Saatavissa: https://developer.apple.com/ios/human-interface-guidelines/ (viitattu 17. 10. 2017).

[52] Google Inc., *Material Motion*. Saatavissa: https://material.io/guidelines/motion/material-motion.html (viitattu 17. 10. 2017).

[53] *Cascading Style Sheets Level 2 Revision 2 (CSS 2.2) Specification*, W3C, 2014. Saatavissa: http://dev.w3.org/csswg/css2 (viitattu 25. 10. 2016).

[54] *Intent to deprecate: SMIL*. Saatavissa: https://groups.google.com/a/chromium.org/forum/#!searchin/blink-dev/SMIL/blink-dev/5o0yiO440LM/59rZqirUQNwJ (viitattu 24. 08. 2017).

[55] *Re: Intent to deprecate: SMIL*. Saatavissa: https://groups.google.com/a/chromium.org/forum/#!topic/blink-dev/5o0yiO440LM%5B126-150%5D (viitattu 29. 08. 2017).

[56] *Web Animations*, W3C, 2016. Saatavissa: https://www.w3.org/TR/web-animations (viitattu 11. 09. 2017).

[57] *Web Animations API*, 2017. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/API/Web_Animations_API (viitattu 05. 11. 2017).

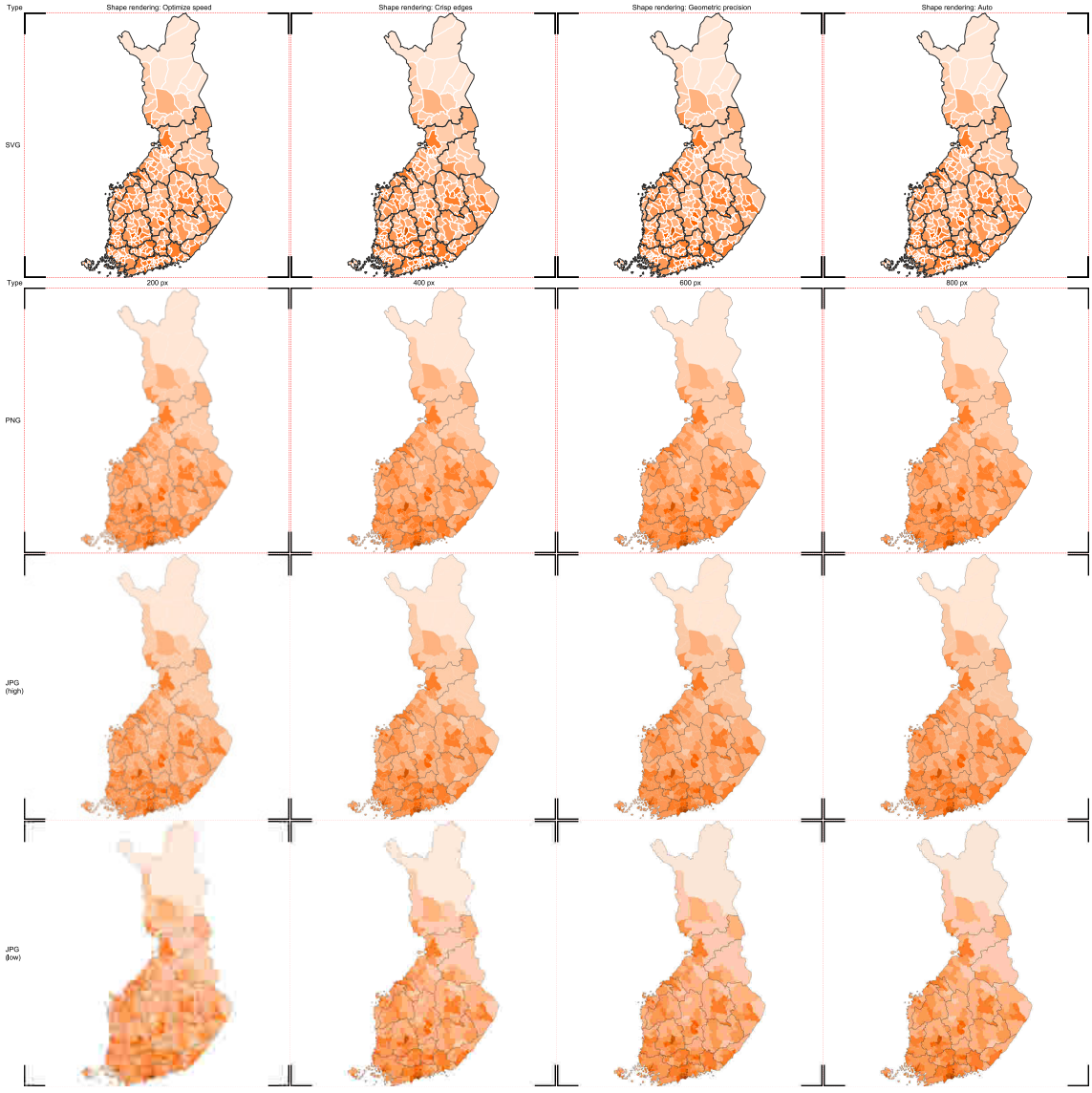
[58] L. Watson, *Using ARIA to enhance SVG accessibility*, 2013. Saatavissa: https://developer.paciellogroup.com/blog/2013/12/using-aria-enhance-svg-accessibility (viitattu 30. 08. 2017).

[59] D. Saffer, *Microinteractions*. Sebastopol: O'Reilly & Associates, Inc., 2013.

[60] *Scalable Vector Graphics (SVG) 2*, W3C, 2016. Saatavissa: https://www.w3.org/TR/SVG2 (viitattu 17. 10. 2017).

[61] O. Ibrahim, *SVG Animations not working in Edge*, 2016. Saatavissa: https://developer.microsoft.com/en-us/microsoft-edge/platform/issues/8487752/#comment-0 (viitattu 25. 09. 2017).

A. HAHMONTUMISEN TESTISIVU



B. HAHMONTUMISEN TESTIKUVAN 'YMPYRÄ' MERKINTÄKOODI

```
<svg xmlns="http://www.w3.org/2000/svg"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     version="1.1" width="800px" height="800px"
     viewBox="-100, -100, 200, 200">
  <defs>
    <clipPath id="hiddenSector">
      <polygon points="-100,-78 0,0 100,-78 100,100 -100, 100" />
    </clipPath>
  </defs>
 <g id="wedges">
    <polygon points="0,0,-0.5,-100,0.5,-100" fill="black" id="a" />
    <g id="g">
      <g id="f">
        <g id="e">
          <g id="d">
            <g id="c">
              <g id="b">
                <use class="wedge-dense" xlink:href="#a"
                     transform="rotate(1)" />
                <use class="wedge-dense" xlink:href="#a"
                     transform="rotate(2)" />
                <use class="wedge-sparse" xlink:href="#a"
                     transform="scale(-1) rotate(52)" />
              </g>
              <use xlink:href="#b" transform="rotate(2)" />
            </g>
            <use xlink:href="#c" transform="rotate(4)" />
          </g>
          <use xlink:href="#d" transform="rotate(8)" />
        </g>
        <use xlink:href="#e" transform="rotate(16)" />
      </g>
      <use xlink:href="#f" transform="rotate(32)" />
    </g>
    <use xlink:href="#g" transform="rotate(64)" />
 </g>
```

```
38  <g id="circles" stroke="black" fill="none"
        clip-path="url(#hiddenSector)">
40   <circle r="1.153" style="stroke-width: 0.99;" />
     <circle r="3.173" style="stroke-width: 0.98;" />
42   <circle r="5.13" style="stroke-width: 0.97;" />
     <!-- ... 95 circles total -->
44  </g>
  </svg>
```

Listaus B.1 *Testikuvan 'Ympyrä' merkintäkoodi.*

C. SELAINTUEN TESTITAULUKOIDEN SELITE

	A	B	C	D	E	F	G
	1: SVG:n näyttäminen						
1	Mallina toimiva PNG-kuva.	HTML-merkintään suoraan liitetty SVG-merkintä.	<img>-elementin avulla linkitetty SVG-tiedosto.	<img>-elementin avulla linkitetty SVGZ-tiedosto.			
2	<object>-elementin avulla linkitetty SVG-tiedosto.	<embed>-elementin avulla linkitetty SVG-tiedosto.	<iframe>-elementin avulla linkitetty SVG-tiedosto.	CSS:n avulla taustakuvaksi asetettu SVG-tiedosto.			
	2: Animaatiotuki eri tekniikoilla						
3: CSS	Leikkauspolulla rajatun <image>-elementin koon muutos.	JavaScriptin avulla vaihtuvan luvun reunaviivan animointi.	Muotoaan muuttava polku muodostaa vuorotellen kirjaimet S, V ja G.	Polulle asetetun tekstin siirtyminen eteen ja taaksepäin.	Täytteenä toimivan väriliu'un keskimmäinen väri siirtyy edestakaisin.		
4: JS							
5: SMIL							
	3: Sekalaisia ominaisuuksia						
6	SVG-suodattimen soveltaminen muuttuvaan tekstiin.	CSS-suotimen animointi ja soveltaminen <text>-elementtiin.	Dynaamisesti muuttuva leikkuupolku.	Dynaamisesti muuttuva, suodatettu maski sovellettuna rasterikuvalle.	Animoidun ja suodatetun reunaviivan käyttö maskina rasterikuvalle.	Kokotesti, jossa SVG-attribuutti ja CSS-sääntö ovat ristiriidassa.	Sama, kuin edellinen, mutta viewBox on kolminkertainen.
7	style-attribuutilla asetettu kirjainkoko.	Valitsimella asetettu kirjainkoko.	Sama, kuin edellinen, mutta viewBox on kaksinkertainen.	CSS:llä animoitu kirjainväli polulle asetetulla tekstillä.	Animoituja CSS-muunnoksia sovellettuna SVG:n elementteihin.	APNG-kuva.	<canvas>-elementti, johon kirjoitetaan "OK!", mikäli tuki löytyy.
8	JavaScriptin avulla muotoaan muuttava <polyline>-elementti.	JavaScriptin avulla muotoaan vaihtava moniosainen polkuelementti.	Edellisen solun polku suodatettuna maskina rasterigrafiikalle.	Edellisen solun polku leikkuupolkuna rasterigrafiikalle.	CSS:llä animoitu kirjainväli <img>-elementillä linkitetyssä SVG-kuvassa.	Animoidulla CSS-muunnoksella pyörivä monikulmio linkitettynä <img>-elementillä.	Sama, kuin edellinen, mutta animoitu JavaScriptillä. Ei kuulu toimia, koska skriptin ajaminen olisi tietoturvariski.
9	SMIL:illä toteutettu muotoa muuttava polku <img>-elementissä.	SMIL:illä toteutettu liikepolkuanimaatio <img>-elementissä.	Teksti asetettuna SMIL:in avulla muotoaan muuttavalle polulle.	Klikkauksesta käynnistyvä pulssianimaatio.	Sama, kuin edellinen, mutta <img>-elementillä linkitettynä.	SMIL:in avulla animoitu moniosainen polku leikkuupolkuna rasterigrafiikalle.	Sama, kuin edellinen, mutta polku leikkaa -elementtiä.

D. BOIDS-ALGORITMIN HAHMONNUSKOMPONENTTIEN LÄHDEKOODIT

```
class CanvasRenderer implements IRenderer {

  constructor(flock: Flock, symbol: Vector[]) {

    // Set up the canvas context
    const $canvas = document.createElement("canvas");
    this.width = flock.width;
    this.height = flock.height;
    $canvas.id = "canvas";
    $canvas.width = flock.width;
    $canvas.height = flock.height;
    this.ctx = $canvas.getContext('2d');
    this.symbol = symbol;
    document.body.appendChild($canvas);
  }

  private drawBoid(boid: Boid): void {

    let absolutePoints: Vector[] = [];

    // Orient the coordinates of the prototype boid
    for(let point of this.symbol){
      absolutePoints.push(point.rotate(boid.orientation)
                  .add(boid.location));
    }

    // Construct the boid line by line
    this.ctx.strokeStyle = "white";
    this.ctx.lineWidth = 2;
    this.ctx.beginPath();
    this.ctx.moveTo(absolutePoints[0].x, absolutePoints[0].y);
    this.ctx.lineTo(absolutePoints[1].x, absolutePoints[1].y);
    this.ctx.lineTo(absolutePoints[2].x, absolutePoints[2].y);
    this.ctx.stroke();
    this.ctx.closePath();
```

```
    }
37
    draw(flock: Flock) {
39
      // Wipe the canvas clean with a black rectangle
41    this.ctx.fillStyle = "black";
      this.ctx.fillRect(0, 0, this.width, this.height);
43
      // Invoke a method to re-draw each boid
45    for(let boid of flock.boids) {
        this.drawBoid(boid);
47    }
    }
49 }
```

Listaus D.1 *Hahmonnuskomponentti toteutettuna* `<canvas>`*-elementillä.*

```
1 class SvgRenderer implements IRenderer {

3   constructor(flock: Flock, symbol: Vector[]) {

5     // Set up the actual SVG element
      this.ctx = document.createElementNS(SvgRenderer.svgNS, "svg");
7     this.ctx.setAttribute( "viewBox"
        , "0 0 " + String(config.width) + " " + String(config.height));
9     this.ctx.setAttribute("xmlns:xlink", "http://www.w3.org/1999/xlink");

11    // Create the prototype boid element
      let $defs = document.createElementNS(SvgRenderer.svgNS, "defs");
13    let $boid = document.createElementNS(SvgRenderer.svgNS, "g");
      let $ymbol = document.createElementNS(SvgRenderer.svgNS, "polyline");
15
      $boid.setAttribute( "id", "boid");
17    let path: string = "";
      for(let point of symbol){
19      path += `${point.x},${point.y} `;
      }
21    $ymbol.setAttribute( "points", path);

23    $boid.appendChild($ymbol);
      $defs.appendChild($boid);
25    this.ctx.appendChild($defs);

27    document.body.appendChild(this.ctx);

29    // Place each boid on the view
      for(let boid of flock.boids){
31      let $boid = document.createElementNS(SvgRenderer.svgNS, "g");
```

```
        let $use = document.createElementNS(SvgRenderer.svgNS, "use");
33      $boid.setAttribute( "transform"
          , `translate(${String(boid.location.x)}
35                      ${String(boid.location.y)})` );
        $use.setAttribute("transform"
37        , `rotate(${boid.orientation * 180 / Math.PI})`);
        $boid.appendChild($use);
39      this.ctx.appendChild($boid);

41      // Link the <use> element to the boid prototype
        $use.setAttributeNS(SvgRenderer.xlinkNS, "href", "#boid");
43      boid.element = $boid;
      }
45  }

47  draw(flock: Flock): void {
      // To render a new frame, loop through each boid and update
49    // their position and orientation
      for(let boid of flock.boids){
51      boid.element.setAttribute( "transform"
          , `translate(${String(boid.location.x)}
53                      ${String(boid.location.y)})` );
        let $use = boid.element.firstElementChild;
55      $use.setAttribute( "transform"
          , "rotate(" + (boid.orientation * 180 / Math.PI) + ")")
57    }
    }
59 }
```

Listaus D.2 *Hahmonnuskomponentti toteutettuna SVG:llä.*

E. DYNAAMISEN GRAFIIKAN SUORITUSKYKYMITTAUKSET