Johanna Pelkonen

SVG – UUSI VANHA VEKTORIFORMAATTI

Opinnäytetyö
Tietojenkäsittely

5.2.2010

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

Opinnäytetyön päivämäärä
5.2.2010

Tekijä(t)
Johanna Pelkonen

Koulutusohjelma ja suuntautuminen
Tietojenkäsittely

Nimeke
SVG – Uusi vanha vektoriformaatti

Tiivistelmä

Flash on tunnettu vektoriformaatti, mutta SVG vielä tuntematon, vaikka W3C standardisoikin sen jo lähes kymmenen vuotta sitten. Opinnäytetyön tavoitteenna onkin esitellä SVG niin hyvin, että lukijat voivat ohjelmoida omia SVG-sovelluksia opinnäytetyön pohjalta. Opinnäytetyön toisena tavoitteena on vertailla SVG:tä ja Flashia keskenään, sillä molemmilla voi tehdä selainpohjaisia, vuorovaikutteisia sovelluksia. Opinnäytetyön toimeksiantaja on Enkora Oy.

Opinnäytetyössä esitellään SVG-dokumentin rakenne ja kaikki keskeisimmät elementit. Opinnäytetyössä on koodiesimerkkejä elementtien luonnista ja pieni opas SVG:n käsittelemisestä JavaScriptillä sekä animoinnista SMIL-kielellä.

Esimerkkisovellus tehtiin SVG:llä. Esimerkkisovelluksen tapauksessa SVG oli tarkoituksenmukaisempi kuin Flash, sillä sovelluksen täytyy pystyä helposti kommunikoimaan muiden sovellusten kanssa. SVG:llä kommunikointi ei ole ongelma, sillä se perustuu XML-kieleen. Sovellus on selainpohjainen työkalu pohjapiirrosten luomiseen. Sillä voidaan luoda huoneita ja sijoittaa kameroita ja ovia haluttuihin paikkoihin. Sovellus liittyy Enkoran muihin kulunvalvontasovelluksiin ja nopeuttaa heidän työprosessejaan.

SVG ja Flash ovat loppujen lopuksi melko samanlaisia, sillä kumpaakin voi sekä ohjelmoida että piirtää. Suurin ero näiden kahden vektoriformaatin välillä on niiden tallennusmuoto: SVG tallennetaan ihmisellekin luettavana tekstinä, mutta Flash tallennetaan vain koneen luettavissa olevaan binäärimuotoon. Vaikka opinnäytetyön tavoite on vertailla SVG:tä ja Flashia, ei voida sanoa kumpi formaateista on parempi. Molemmilla formaateilla on hyvät ja huonot puolensa. Riippuukin käyttötarkoituksesta, kannattaako sovelluksen kehitykseen valita SVG vai Flash.

Asiasanat (avainsanat)
SVG, vektorigrafiikka, vektoriformaatti, Flash, Flex

Sivumäärä	Kieli	URN
34 s. + liitt. 4 s.	Suomi	

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi
Arto Väätäinen

Opinnäytetyön toimeksiantaja
Enkora Oy Ltd

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

Date of the bachelor's thesis

5 February 2010

Author(s)
Johanna Pelkonen

Degree programme and option
Business Information Technology

Name of the bachelor's thesis

SVG – The new old vector format

Abstract

Flash is a popular format for vector graphics whereas SVG is still quite unknown although it was standardized nearly ten years ago by W3C. This study was done for a company named Enkora and its goal was to present SVG so well that readers could program their own SVG applications. Another goal was to compare SVG and Flash since both could be used to create interactive applications for the Internet.

This study presented the structure of SVG documents and all important elements. The study included many code examples for creating the elements and a short tutorial about programming SVG with JavaScript and animating it with SMIL. In addition, a sample application was made in SVG. With the sample application SVG was more appropriate format than Flash because the application had to communicate with other applications. This communication was not a problem since SVG was basicly XML. The application was an internet-based tool that could be used to create floorplans for buildings allowing users to create rooms and place doors and cameras where they want to. The application would be a part of Enkora's other access control software and it would fasten their processes.

As a result, SVG and Flash were quite similar because they could both be created by programming or drawing. The main difference between SVG and Flash was how they are saved: SVG was plain text that a human could read but Flash was saved in a binary format that was readable only for computers. Although the goal for this study was to compare SVG and Flash, it appeared to be impossible to say which one would be better. Both SVG and Flash had their good and bad features. It depends on the purpose of the software, which one would be better.

Subject headings, (keywords)
SVG, Flash, Flex, vector graphics, vector format

Pages	**Language**	**URN**
34 p. + app. 4 s.	Finnish	

Remarks, notes on appendices

Tutor	**Bachelor´s thesis assigned by**
Arto Väätäinen	Enkora Oy Ltd

SISÄLTÖ

1 JOHDANTO

Tämä opinnäytetyö esittelee SVG-vektorigrafiikkaformaatin. W3C standardisoi SVG-tiedostomuodon jo vuonna 2001, mutta silti tämä formaatti on edelleen melko tuntematon. SVG on XML-pohjainen kieli, jolla voi luoda vektorikuvia joko piirtämällä tai kirjoittamalla. Esittelen tässä opinnäytetyössä SVG:n koodaamisen perusteet. Lisäksi esittelen hieman Flashia ja vertailen sitä SVG:n kanssa.

Tutkimusongelmani on: kumpi soveltuu paremmin esimerkkitapauksen toteutukseen, SVG vai Flash? Tavoitteeni on esitellä SVG riittävän laajasti, jotta lukijat voivat tehdä omia SVG-sovelluksiaan. Tavoitteenani on myös tuoda esille SVG:n ja Flashin välisiä eroja.

Esimerkkitapauksen toimeksiantaja oli Enkora Oy. Esimerkkisovellus on internetpohjainen työkalu rakennusten pohjapiirrosten tekemiseen. Tein sovelluksen SVG:llä, sillä XML-kieleen pohjautuva SVG oli tarkoituksenmukaisempi kuin Flash. Sovelluksessa käyttäjät voivat ladata pohjapiirroksen kuvan malliksi ja piirtää sen päälle huoneet sekä sijoittaa ovia ja kameroita haluamiinsa paikkoihin. Sovellus liittyy Enkoran kulunvalvonta sovelluksiin ja nopeuttaa heidän toimintaprosessejaan.

Opinnäytetyön ensimmäisessä osassa esittelen vektorigrafiikan eroja rasterigrafiikkaan. Tämän jälkeen siirryn esittelemään SVG-formaattia: sen kirjoitussääntöjä, perusmuotojen koodausta sekä animointia ja ohjelmointia. Kun SVG on esitelty, esittelen lyhyesti Flashin ja Flexin sekä vertailen SVG:tä ja Flashia toisiinsa. Opinnäytetyön lopuksi esittelen esimerkkisovelluksen toiminnan.

2 GRAFIIKKA INTERNETISSÄ

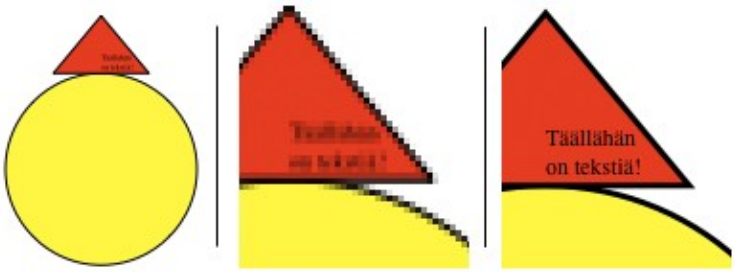
Internet koostuu useista eri elementeistä, kuten tekstistä, kuvista ja videoista. Kuvia voidaan esittää kahdessa eri muodossa: vektoreina tai rastereina. Rasterikuvat koostuvat yhden pikselin kokoisista neliöistä, joista kukin on täytetty jollakin RGB-värillä. Tämän vuoksi rasterikuvia kutsutaan myös bittikarttakuviksi. Ihanteellisin kohde käyt-

tää rasterigrafiikkaa ovat valokuvat, joiden halutaan säilyttävän muotonsa – vaikka kyseessä olisikin vain viivoista koostuva piirros, ei jokaista viivaa tarvitse erikseen päästä muokkaamaan, vaan piirrosta käsitellään kokonaisuutena. Rasterikuvia välitetään pakatuissa muodoissa, kuten JPEG, GIF tai PNG. (Eisenberg 2002, 13 – 14.)

Jos rasterikuvat ovat joukko neliöitä, joilla kullakin on oma paikkansa ja värinsä, vektorikuvat ovat joukko ohjeita koordinaateista, joiden mukaan kuva piirretään. Vektorikuvat koostuvat geometrisista muodoista eli objekteista, kuten viivat, kaaret ja monikulmiot, jotka vektorinkatselуohjelma piirtää saamiensa koordinaatistojen ja koordinaattien mukaan. (Eisenberg 2002, 13 – 14.) Vektorigrafiikkaa välitetään esimerkiksi muodoissa AI (Adobe Illustrator) tai SVG.

Koska vektorigrafiikka muodostuu juuri objekteista eikä pikseleistä, kuten rasterikuvat, vektorikuvaa voi muokata esimerkiksi vaihtamalla yksittäisten objektien kokoa tai väriä. Vektorigrafiikalla piirrettyyn tekstiin voi myös suorittaa hakuja XML-kieltä taitavalla hakukoneella (Watt ym. 2003, 23). Tekstihän on loppujen lopuksi vain tekstiä, riippumatta siitä kuinka sitä on väritetty tai taivuteltu (Eisenberg 2002, 13 – 14). Vektorigrafiikassa on myös mahdollista lisätä eri kieliversioita tekstille, joista valitaan sopiva esimerkiksi selaimen kielen mukaan. Näin ollen käyttäjälle voidaan näyttää teksti hänen omalla kielellään ilman, että kullekin kielelle tarvitsee tehdä kokonaan omaa kuvaa. (Watt ym. 2003, 19 – 20.)

Tekstin käsiteltävyyden lisäksi vektorigrafiikasta tekee rasterigrafiikkaa monikäyttöisemmän sen skaalautuvuus, eli kuvan laatu ei kärsi, vaikka sitä suurennettaisiin tai pienennettäisiin mielivaltaisesti. Siinä missä rasterikuvaa suurennettaessa jokaista pikseliä suurennetaan kertoimen verran, vektorikuvan objektien koordinaatit kerrotaan kertoimella, jonka jälkeen ohjelma piirtää kuvan uusien koordinaattien mukaan. Tällöin kuvan laatu ei kärsi, kuten rasterigrafiikassa, jossa kuva pikselöityy eli tulee karkeammaksi. (Eisenberg 2002, 15 – 16.) Kuvassa 1 vasemmalla on alkuperäinen kuva, keskellä rasterigrafiikan suurennos ja oikealla vektorigrafiikan suurennos.



KUVA 1. Rasteri- ja vektorigrafiikan suurennoksen erot

Vektorigrafiikan ominaisuus rajattomaan suurentamiseen antaa mahdollisuuden sijoittaa yksityiskohtaista tietoa kuviin. Esimerkiksi karttakuvissa voi hyödyntää tätä ominaisuutta sijoittamalla jopa rakennuskohtaista tietoa hyvinkin laajaa aluetta esittävään kuvaan. Rasterigrafiikalla pitäisi piirtää kaksi kuvaa, joista toinen näyttää laajemman alueen ja toinen yksityiskohtaisen tiedon rajatummasta alueesta. (Watt ym. 2003, 12.)

Vektorigrafiikan levittämiseen internetissä on nykyään kaksi eri formaattia: Flash ja SVG. Flash on varmasti kaikille tuttu formaatti, jolla toteutetaan sekä näyttäviä nettisivuja että pikkuisia selainpohjaisia pelejä. SVG sitä vastoin on edelleen monille vieras formaatti, vaikka W3C standardisoikin sen jo vuonna 2001. SVG ja Flash ovat joillakin osa-alueilla kilpailevia formaatteja, mutta molemmilla on ominaisuutensa, joissa toinen formaatti on selkeästi tarkoituksenmukaisempi.

3 SVG

SVG on lyhenne sanoista Scalable Vector Graphics. Se on XML:än perustuva kuvauskieli, jolla voidaan luoda vektorigrafiikkaa. SVG on täysin vapaasti käytettävissä, sillä sitä eivät sido mitkään patentit tai kopiosuojaukset, eikä sen luomiseen tarvita mitään erityisiä ohjelmia – millä tahansa tekstieditorilla voi luoda SVG-tiedoston, sillä sehän on tekstipohjaista XML:ää. (Mertz 2005.)

SVG-formaatin asemaa vahvistaa se, että W3C on ottanut sen suositeltavien formaattiensa joukkoon. SVG:stä on saatavilla täydellinen dokumentaatio, jota tukevat monet suuret yhtiöt, kuten Adobe, Apple, Microsoft ja Sun (Jackson 2002, 1). Niinpä SVG-

kuvien katseluun tarvittavat ominaisuudet on liitetty kiinteäksi osaksi nykyisiä selaimia ja vanhemmilla selaimilla grafiikan saa näkyviin esimerkiksi Adoben SVG-katselulisäosalla (Mertz 2005).

SVG-tiedostojen pääte on .svg tai .svgz. Jälkimmäinen tarkoittaa gzip-menetelmällä pakattua SVG-tiedostoa. Gzip on ainoa standardiksi valittu tapa pakata SVG-tiedostoja, joten kaikkien SVG-tiedostoja näyttävien ohjelmien on tuettava myös pakattua tiedostomuotoa. (Jackson 2002, 2.)

3.1 SVG-dokumentti

SVG-grafiikka sopii erinomaisesti sellaisten kuvien piirtämiseen, jotka pohjautuvat XML-muotoiseen tietoon (Watt ym. 2003, 7). Esimerkiksi lämpötiladiagrammin voisi helposti piirtää SVG:llä, jos vaikkapa päiväkohtaiset keskilämpötilat tarjottaisiin XML-tiedostona. SVG:llä voi näyttää kolmea erilaista visuaalista kokonaisuutta: graafisia muotoja, tekstiä ja bittikarttakuvia (Watt ym. 2003, 8). Vektorikuvaan voi siis sisällyttää myös rasterikuvia. Ne eivät silti saa samoja ominaisuuksia kuin vektorikuvat, kuten ääretöntä skaalautuvuutta.

XML:än perustuminen sitoo SVG:n tiukasti osaksi internetin toiminnallisuutta (Jackson 2002, 2). SVG-formaattia voi kuitenkin käyttää internetin ulkopuolellakin: se voi olla formaatti, jolla kuvia välitetään eri vektorigrafiikkaohjelmalta toiselle tai jopa aivan erityyppisten ohjelmien kesken (Moock 1999). Kuten Moock (1999) sanoo: ”Voisit luoda kuvan Wordissa ja lähettää sen sähköpostilla ystävällesi, joka lukee tekstimuotoisen kuvauksen matkapuhelimestaan ja lataa sen sitten nettisivuilleen kollegansa nähtäville, joka voi ladata sen omaan vektorigrafiikkaohjelmaansa ja jatkaa kuvan muokkausta”.

Vaikka SVG:tä voikin käyttää monessa eri ympäristössä, se sopii täydellisesti internetiin. Niin se on suunniteltukin olemaan, sillä kaikki sen ominaisuudet ovat standardisoituja ja avoimia lähdekoodiltaan. Perustuminen XML:än antaa mahdollisuuden käyttää muotoiluja kuten XSL ja pääsyn dokumenttirakenteeseen DOM:in avulla.

Tyylitiedoston voi tehdä myös CSS:llä ja SVG-dokumentin ohjelmointiin voi käyttää mitä tahansa ohjelmointikieltä. (Jackson 2002, 2.)

3.1.1 Kirjoitussäännöt

Koska SVG-dokumentit ovat XML:ää, niillä on tarkka rakenne ja kirjoitussäännöt. Nämä säännöt ovat tismalleen samoja kuin XML:ssä. (Watt ym. 2003, 51.) Tärkeimmät kirjoitussäännöt ovat:

1) kukin tagi on lopetettava asianmukaisesti
2) isot ja pienet kirjaimet ovat eri merkkejä
3) erikoismerkit on kirjoitettava entiteeteillä.

SVG:ssä kaikki sisältö kirjoitetaan asiaankuuluvien tagien sisään. Esimerkiksi koko SVG-dokumentti kirjoitetaan tagiparin `<svg></svg>` sisään. Aloittavalla tagilla on oltava myös päättävä tagi, joka merkitään samalla tavalla kuin aloittava tagi, mutta se alkaa kauttaviivalla. On tilanteita, joissa tagiparin sisään ei tule tekstiä, jolloin tagin voi päättää ensimmäisen tagin sisällä kauttaviivaan: `<circle ... />`. (Watt ym. 2003, 53 – 54.)

Suurin osa SVG:n valmiiksi määritellyistä tageista sisältää ainoastaan pieniä kirjamia. Iso ja pieni kirjain merkitsevät eri asioita, joten vaikka SVG-dokumentti tunnistaa tagin `<circle>`, se ei tunnista tagia `<CIRCLE>` tai `<Circle>`. SVG:ssä on määritelty yksi tagi, jossa on myös iso kirjain: `<clipPath>`. On tärkeää kirjoittaa tagi juuri niinkuin se on määritelty, sillä muuten SVG ei tunnista sitä. (Watt ym. 2003, 53.) Tagin `clipPath` nimeämisessä on on käytetty camelCase-sääntöä, eli ensimmäinen sana alkaa pienellä kirjaimella ja seuraavat sanat isolla.

Koska SVG-dokumentin rakenne koostuu tageista, jotka erotellaan tavallisesta tekstistä merkein < ja >, ei näitä merkkejä voi käyttää sellaisenaan normaalin tekstin joukossa. Esimerkiksi seuraava lause aiheuttaisi virheen, sillä SVG luulee löytäneensä circle-elementin aloitustagin, mutta ei löydä sille lopetustagia (Watt ym. 2003, 56):

```
<text x="10" y="10" fill="black">Ympyrä luodaan <circle>-
tagilla.</text>
```

Yllä olevan sijaan <- ja >-merkit tulee korvata entiteeteillä `<` ja `>`. Muita korvattavia merkkejä ovat lainausmerkki ” (`&quot;`), hipsu ' (`&apos;`) sekä et-merkki & (`&`). (Watt ym. 2003, 55.)

3.1.2 Dokumentin rakenne

Kuten mainittua, SVG-dokumentin sisältö kirjoitetaan `<svg></svg>` -tagiparin sisään. Dokumentin alkuun, svg-tagien ulkopuolelle, voi kuitenkin sijoittaa tietoa käytettävästä XML-versiosta ja dokumentin tyypin määrityksen (Document Type Declaration, DTD). SVG-dokumentin rakenne voisi olla seuraava:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg>
<!-- Kommentti: Tähän dokumentin sisältö -->
</svg>
```

Jos XML-merkintä lisätään dokumenttiin, se tulee sijoittaa aivan dokumentin alkuun. Edes riviväliä tai välilyöntiä ei saa olla ennen XML-merkintää. Version-attribuutti on pakollinen, mutta encoding-attribuuttia ei ole pakko laittaa. Toisella ja kolmannella rivillä oleva doctype-määritys kertoo dokumentin olevan SVG:tä, jonka dokumentaatio löytyy toisella rivillä olevasta osoitteesta. (Watt ym. 2003, 52.)

Kuten yllä olevasta esimerkistä käy ilmi, SVG-dokumenttiin voi lisätä kommentteja `<!-- -->` merkien väliin. SVG-dokumenttiin voi myös sijoittaa JavaScript tai CSS-koodia. Koska kumpikaan niistä ei ole XML-kieltä, ne on erotettava SVG-dokumentista seuraavalla tavalla:

```
<!-- Seuraavaksi JavaScript-koodia -->
<script type="text/javascript">
<![CDATA[
//JavaScript-koodi tähän
]]>
</script>
```

```
<!-- Seuraavaksi CSS-koodia -->
<style type="text/css">
<![CDATA[
//CSS-koodi tähän
]]>
</style>
```

(Eisenberg 2002, 50; Watt ym. 2003, 59.)

SVG-tiedostoon voi myös liittää ulkoisia tiedostoja. CSS-tiedosto liitetään <svg>-tagien ulkopuolelle seuraavalla koodilla:

```
<?xml-stylesheet href="tyylitiedosto.css" type="text/css"?>
```

(Eisenberg 2002, 51.)

Ulkoinen JavaScript-tiedosto saadaan linkitettyä <svg>-tagien sisällä seuraavalla koodilla:

```
<script xlink:href="javascript.js" />
```

JavaScript-tiedoston linkittämiseen on käytetty xlink-nimiavaruutta, joka on määriteltävä `<svg>`-tagin sisässä seuraavasti:

```
<svg width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
```

SVG-dokumentin rakenteeseen kuuluvat svg-elementin lisäksi defs-, use- ja g-elementit. Svg-elementti muodostaa katseluikkunan (engl. *viewport*), jossa kaikki muut elementit näytetään. Katseluikkuna voi sisältää useita svg-elementtejä. Svg-elementille voi määritellä leveyden ja korkeuden attribuuteilla `width` ja `height`. Oletusarvot ovat molemmille 100 %. (Watt ym. 2003, 60 – 63.) Katseluikkuna sisältää koordinaatiston, jonka nollapiste on vasen yläkulma.

Defs-elementin sisällä olevia elementtejä ei tulosteta heti, vaan niitä voi use-elementin avulla käyttää myöhemmin dokumentissa (Eisenberg 2002, 55). Alla on esimerkki, joka tulostaa neljä peruselementtiä. Use-elementin käyttäminen vaatii xlink-laajennoksen, jonka lisääminen on esitetty edellisessä esimerkissä.

```
<defs>
    <circle id="ympyra" stroke="#cc0099" fill="#ff99cc" r="25"
cx="30" cy="25"/>
    <rect id="suorakulmio" x="75" y="5" width="30" height="40" stro-
ke="black" fill="#33ff33" stroke-dasharray="4,2" />
    <path id="polku" d="M 5,100 A 25,100 0,0,0 30,90 A 20,80 0,0,1
50,70" fill="none" stroke="#ff9933" stroke-width="3" />
    <polygon id="tahti" stroke="black" fill="#3366ff" points="60,80
80,80 90,60 100,80 120,80 105,95 110,115 90,105 70,115 75,95" />
</defs>

<use xlink:href="#ympyra" />
<use xlink:href="#suorakulmio" />
<use xlink:href="#polku" />
<use xlink:href="#tahti" />
```

Yllä oleva koodi tulostaa kuvan 2:

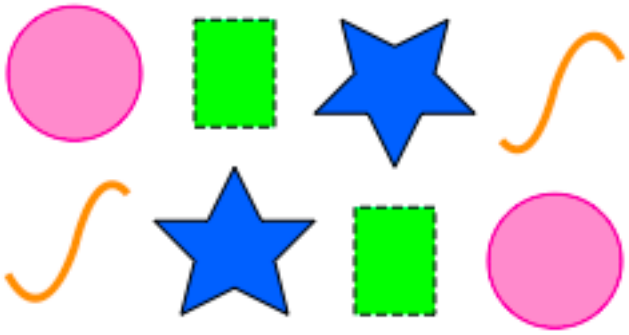
KUVA 2. Defs- ja use-elementin käyttöä

G-elementillä voi ryhmittää useita elementtejä. Näin kaikille ryhmän elementeille voi antaa esimerkiksi samat tyyli- tai paikkamäärittelyt g-elementin attribuutteina. Tämä lyhentää koodia, kun jokaiselle elementille ei tarvitse erikseen kirjoittaa samoja tietoja. (Eisenberg 2002, 53; Watt ym. 2003, 69.) Alla on esimerkki edellisen esimerkin koodista, joka on asetettu g-elementin sisään. Perusmuodot on tulostettu kahdesti, joista oikeanpuoleiseen on use-elementissä käytetty transform-attribuuttia kääntämään kuvaa 180 astetta.

```
<defs>
    <g id="perusmuodot">
        <circle id="ympyra" stroke="#cc0099" fill="#ff99cc" r="25"
cx="30" cy="25"/>
        <rect id="suorakulmio" x="75" y="5" width="30" height="40"
stroke="black" fill="#33ff33" stroke-dasharray="4,2" />
        <path id="polku" d="M 5,100 A 25,100 0,0,0 30,90 A 20,80
0,0,1 50,70" fill="none" stroke="#ff9933" stroke-width="3" />
        <polygon id="tahti" stroke="black" fill="#3366ff"
points="60,80 80,80 90,60 100,80 120,80 105,95 110,115 90,105 70,115
75,95" />
    </g>
</defs>
```

```
<use xlink:href="#perusmuodot" x="10" y="10" />
<use xlink:href="#perusmuodot" x="130" y="10" transform="rotate(180
190 70)" />
```

Yllä oleva koodi tulostaa kuvan 3:



KUVA 3. Defs-, g- ja use-elementin käyttöä

3.1.3 Perusmuodot

SVG-tiedostoja voi katsella ohjelmilla, jotka osaavat sitä lukea. Tällaisia ohjelmia ovat esimerkiksi Adobe Illustrator tai useimmat selaimet, kuten Mozilla Firefox. SVG-tiedoston voi myös upottaa osaksi HTML-sivustoa `object`- tai `embed`-tagilla. Selaimesta riippuu, kumpi tageista toimii paremmin. (Mertz 2005.) Alla on esimerkki `embed`-tagin käytöstä:

```
<embed id="svg_pic" src="svg.svg" width="800" height="500"
type="image/svg+xml"
pluginspage="http://www.adobe.com/svg/viewer/install/"
style="border: 1px solid black;" />
```

Kuten muissakin ohjelmointikielissä, myös SVG:ssä ensiksi kirjoitettu elementti piirretään alimmaiseksi ja viimeisenä koodissa oleva elementti päällimmäiseksi.

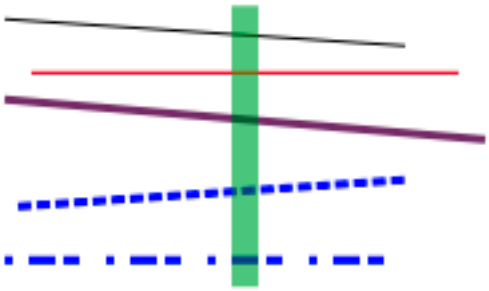
Viiva

Viivoja piirretään `line`-elementillä. Pakolliset attribuutit ovat `x1` ja `y1`, jotka määrittävät viivan alkupisteen koordinaatistossa, `x2` ja `y2` jotka määrittävät viivan loppupis-

teen sekä `stroke`, joka määrittää viivan värin. (Watt ym. 2003, 110.) Koordinaateille ei tarvitse merkitä yksikköä, jolloin oletuksena käytetään pikseleitä. Yksikkönä voi kuitenkin käyttää myös esimerkiksi senttimetrejä, jolloin koordinaatin perään tulee lisätä yksikkö cm. (Eisenberg 2002, 34.) Alla on esimerkki yhden piksen paksuisesta mustasta viivasta:

```
<line x1="10" y1="10" x2="20" y2="150" stroke="black" />
```

`line`-elementille voi antaa useita muitakin tyylimäärityksiä kuin `stroke`. Esimerkiksi `stroke-width` määrittää viivan leveyden, `stroke-dasharray` katkoviivan ulkonäön ja `opacity` läpinäkyvyyden. `stroke`-attribuutin arvoksi voi antaa muitakin värejä kuin mustan. SVG tukee myös värien heksadesimaalista esitysmuotoa.



KUVA 4. Viivoja, joihin on käytetty eri tyylimuuttujia

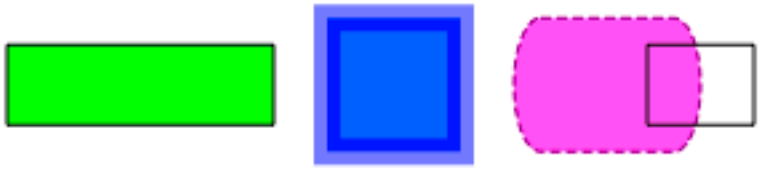
Kuva 4 on saatu aikaan alla olevalla koodilla. Koodin ensimmäinen `line`-elementti kuvaa ylintä viivaa, toinen toiseksi ylintä, jne. Alin `line`-elementti kuvaa pystysuoraa vihreää viivaa, joka on puoliksi läpinäkyvä. Siksi muut viivat näkyvät vihreän viivan läpi, vaikka ne onkin piirretty vihreän pystyviivan alle.

```
<line x1="10" y1="10" x2="160" y2="20" stroke="black" />
<line x1="20" y1="30" x2="180" y2="30" stroke="red" />
<line x1="10" y1="40" x2="190" y2="55" stroke="#663366" stroke-
width="3" />
<line x1="15" y1="80" x2="160" y2="70" stroke="blue" stroke-width="3"
stroke-dasharray="5, 2" />
<line x1="10" y1="100" x2="160" y2="100" stroke="blue" stroke-
width="3" stroke-dasharray="3, 6, 10" />
<line x1="100" y1="5" x2="100" y2="110" stroke="green" stroke-
width="10" opacity="0.5" />
```

Suorakulmio

Suorakulmio luodaan `rect`-elementillä. Pakollisia attribuutteja ovat `x`, `y`, `width` ja `height`. Näistä `x` ja `y` kuvaavat suorakulmion vasemman ylänurkan paikan, `width` suorakulmion leveyden ja `height` korkeuden. Oletuksena suorakulmio sijoitetaan koordinaatistossa kohtaan (0,0) ja sen leveydeksi ja korkeudeksi määritellään 0. Oletuksena suorakulmio täytetään mustalla, eikä sille aseteta reunaviivaa, mutta täytön värin voi itsekin määrätä `fill`-attribuutilla. (Watt ym. 2003, 119.)

Suorakulmion reunaviivaa voi muokata samoilla attribuuteilla kuin yllä esitettyä viivaakin. Reunaviiva piirretään puoliksi kuvion sisään ja puoliksi sen ulkopuolelle (Eisenberg 2002, 40). Tästä on esimerkki kuvassa 5, jossa sinisen neliön reunaviiva on puoliksi läpinäkyvä. Tällöin suorakulmion ulkopuolelle jäävä reunus näkyy vaaleammalla kuin suorakulmion sisällä oleva reunus, jonka läpi näkyy täytön väriä. Violetissa suorakulmiossa on käytetty `rx`- ja `ry`-attribuutteja, joiden avulla voidaan pyöristää kulmia x- ja y-akselien suuntaisesti. Kaikkein oikeammanpuoleisimman suorakulmion täytöksi on asetettu `none`, jolloin suorakulmiosta tulee läpinäkyvä.



KUVA 5. Esimerkkejä suorakulmioista

Yllä olevan kuvan 5 saa aikaiseksi seuraavalla koodilla:

```
<rect x="10" y="20" width="100" height="30" fill="#33ff33"
stroke="black" />

<rect x="130" y="10" width="50" height="50" fill="#3366ff"
stroke="blue" stroke-width="10" stroke-opacity="0.5" />

<rect x="200" y="10" rx="10" ry="25" width="70" height="50"
fill="#ee33ee" fill-opacity="0.7" stroke="purple" stroke-
dasharray="4,2" />

<rect x="250" y="20" width="40" height="30" fill="none"
stroke="black" />
```

Ympyrä ja ellipsi

Ympyrä on periaatteessa tarpeeton elementti, sillä sen voi määritellä ellipsin avulla. Ympyrälle on kuitenkin oma `circle`-elementti, jonka keskipiste asetetaan `cx`- ja `cy`-attribuuteilla ja säde `r`-attribuutilla. Ellipsi luodaan `ellipse`-elementillä, jonka keskipiste määritellään kuten ympyrällekin, mutta säde määritellään `rx`- ja `ry`-attribuuteilla. (Watt ym. 2003, 124 – 125.) Sekä ympyrälle että ellipsille voi määritellä täytön ja reunaviivan muotoiluja kuten edellä kuvatuille elementeillekin. Alla oleva koodi tulostaa kuvan 6 mukaisen kuvion.

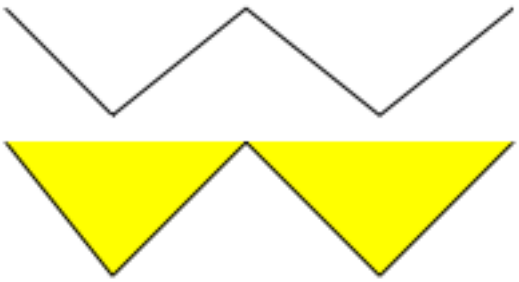
```
<circle cx="40" cy="40" r="30" fill="red" stroke="black" />
<ellipse cx="40" cy="25" rx="20" ry="10" fill="yellow"
stroke="black" />
```

KUVA 6. Ympyrä ja ellipsi

Monikulmaviiva

Monikulmaviivalla (engl. *polyline*) voi luoda viivan, jossa on useita kulmia. Kulmien koordinaatit määritellään `points`-attribuutilla. (Watt ym. 2003, 126.) Monikulmaviivalle voi käyttää kaikkia tyyliattribuutteja, mitä viiva-elementillekin ja se voi lisäksi saada täytön `fill`-attribuutilla. Alla oleva koodi tulostaa kuvan 7 mukaisen kuvion.

```
<polyline points="10,10 50,50 100,10 150,50 200,10" stroke="black"
fill="none" />
<polyline points="10,60 50,110 100,60 150,110 200,60" stroke="black"
fill="yellow" />
```



KUVA 7. Monikulmaviiva

Monikulmio

Monikulmio luodaan samalla tavoin kuin monikulmaviiva, mutta elementin nimi on `polygon`. Lisäksi monikulmio suljetaan, vaikkei ensimmäistä koordinaattia merkitsisikään `points`-attribuutin viimeiseksi koordinaatiksi. (Watt ym. 2003, 128.) Kuvassa 8 oleva tähti syntyy seuraavalla koodilla:

```
<polygon id="tahti" stroke="black" fill="#3366ff" points="60,80 80,80
90,60 100,80 120,80 105,95 110,115 90,105 70,115 75,95" />
```

KUVA 8. Monikulmio

Polku

Edellä kuvatut elementit ovat hyvin rajattuja muodoltaan. Polkuelementin avulla voidaan piirtää lähes mielivaltaisia viivoja ja kaaria. Halutut koordinaatit annetaan `d`-attribuutin arvoksi erinäisten käskyjen kera. Mahdollisia käskyjä ovat:

- *moveto M*, joka siirtää kynän haluttuun koordinaattiin
- *lineto L*, joka piirtää suoran viivan
- *curveto C / Q*, joka piirtää Bezier-kaaren

- *arc A*, joka piirtää elliptisen kaaren
- *closepath z*, joka sulkee polun

(Watt ym. 2003, 231-233.)

Alla oleva koodi siirtää kynän kohtaan (30,30), josta se piirtää viivan koordinaattiin (120,60), sieltä uuden viivan kohtaan (120,40) ja lopulta sulkee polun.

```
<path d="M 30,30 L 120,60 120,40 z" stroke="black" fill="none" />
```

Tämä koodi tulostaa kuvan 9:

KUVA 9. Polku

Bezier-kaaren voi piirtää yhdellä tai kahdella kontrollipisteellä. Jos kaaren piirtää kahdella pisteellä, komentona käytetään kirjainta C. Syntaksi kaarelle on seuraava:

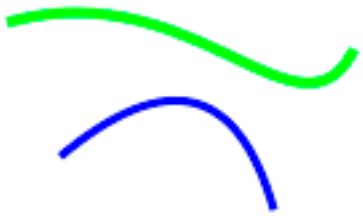
```
d="M x0,y0 C x1,y1 x2,y2 x,y"
```

Yllä olevassa koodissa `x0` ja `y0` tarkoittavat kaaren lähtöpistettä, `x1` ja `y1` lähtöpisteen kontrollipistettä, `x2` ja `y2` loppupisteen kontrollipistettä ja `x` ja `y` loppupistettä. Yhden kontrollipisteen kaari piirretään käskyllä Q. Syntaksi on seuraava:

```
d="M x0,y0 Q x1,y1 x,y"
```

Yllä olevassa koodissa `x0` ja `y0` tarkoittavat kaaren lähtöpistettä, `x1` ja `y1` kaaren kontrollipistettä ja `x` ja `y` loppupistettä. (Watt ym. 2003, 240-243.) Kuvassa 10 on vihreällä Bezier-kaari kahdella kontrollipisteellä ja sinisellä yhdellä kontrollipisteellä. Kaaret saa aikaiseksi seuraavalla koodilla:

```
<path d="M 20,100 C 90,80 130,150 150,110" stroke="#33ff33" stroke-
width="4" fill="none" />
<path d="M 40,150 Q 100,100 120,170" stroke="blue" stroke-width="3"
fill="none" />
```



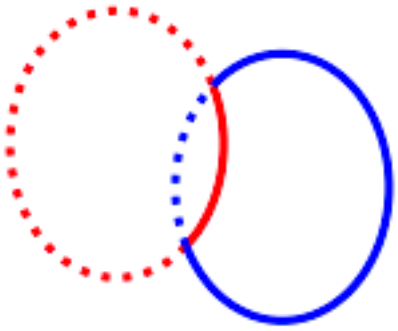
KUVA 10. Bezier-kaaria

Komennolla A voi piirtää ellipsin kaaren mukaan kulkevia kaaria seuraavalla syntaksilla:

```
d="M x0,y0 A rx,ry x-axis-rotation large-arc-flag sweep-flag x,y"
```

Yllä olevassa koodissa `x0` ja `y0` tarkoittavat kaaren lähtöpistettä, `rx` ja `ry` ellipsin x- ja y-sädettä, `x-axis-rotation` tarkoittaa x-akselin kiertoastetta, `large-arc-flag` käytetäänkö pitkää vai lyhyempää kaarta ja `sweep-flag` kierretäänkö myötä- vai vastapäivään. `x` ja `y` tarkoittavat kaaren loppupistettä. (Watt ym. 2003, 244.) Alla oleva koodi tulostaa neljä kaarta, jotka muodostavat ne kaksi ellipsiä, joiden mukaan kaari voi kulkea. Huomaa, että kaaren määrittelyssä muutetaan ainoastaan pitkän tai lyhyen kierron sekä vasta- tai myötäpäivän määrittävää arvoa. Koodi tulostaa kuvan 11 mukaisen kuvion.

```
<path d="M 150,300 A 40,50 0 0 0 160,240" stroke="red" stroke-
width="3" fill="none" />
<path d="M 150,300 A 40,50 0 1 1 160,240" stroke="red" stroke-
width="3" stroke-dasharray="3,5" fill="none" />
<path d="M 150,300 A 40,50 0 1 0 160,240" stroke="blue" stroke-
width="3" fill="none" />
<path d="M 150,300 A 40,50 0 0 1 160,240" stroke="blue" stroke-
width="3" stroke-dasharray="3,5" fill="none" />
```



KUVA 11. Elliptiset kaaret

SVG-kuva voi sisältää tekstiä `text`-elementin avulla. Elementille määritetään attribuutit `x` ja `y`, joista `x` tarkoittaa etäisyyttä vasemmasta reunasta ja `y` kirjainten alareunan etäisyyttä ylhäältä – tällä kertaa ei siis määritetä vasenta yläkulmaa, vaan vasen alakulma. (Watt ym. 2003, 310.) Tekstisisältö kirjoitetaan `text`-tagien väliin, kuten alla olevassa koodissa:

```
<text x="10" y="20">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit.
</text>
<text x="10" y="40">
    Etiam sit amet urna augue.
</text>
```

Molemmat lauseet tulostetaan omille riveilleen. Jos lauseet laitetaan eri `text`-tagien sisään, niitä ei voi maalata yhtä aikaa. Siksi onkin suotavaa käyttää `tspan`-elementtiä erottamaan rivit toisistaan saman `text`-tagin sisällä (Watt ym. 2003, 312):

```
<text x="10" y="20">
    <tspan>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    </tspan>
    <tspan x="10" dy="20">
        Etiam sit amet urna augue.
    </tspan>
</text>
```

Attribuuttia `dy` voi käyttää määrittämään `tspan`-elementin sijainti y-akselilla relatiivisesti edelliseen `tspan`-elementtiin nähden. Vastaavasti `dx` määrittää relatiivisen sijainnin x-akselin suhteen. (Watt ym. 2003, 312.)

Tekstistä – tai mistä tahansa muusta elementistä – voi myös tehdä linkin. Siihen tarvitaan xlink-laajennosta. (Watt ym. 2003, 76.) Alla oleva koodi tuottaa sanan Google, jota klikkaamalla Googlen etusivu aukeaa uuteen ikkunaan.

```
<a xlink:href="http://google.com" target="_blank">
    <text x="10" y="100">
        Google
    </text>
</a>
```

3.2 SVG:n ohjelmointi

SVG:n vahvuus on sen ohjelmoitavuus. Koska se on XML-kieltä, sitä voi ohjelmoida millä kielellä tahansa, joka vain tukee DOMia. Suosituin kieli SVG:n ohjelmointiin on kuitenkin ECMAScript tai JavaScript. ECMAScript on standardisoitu kieli, joka on kehitetty JavaScriptin pohjalta. Näillä kahdella kielellä on kuitenkin hyvin vähän eroavaisuuksia. (Watt ym. 2003, 578.) Tämän luvun esimerkeissä käytetään JavaScriptiä.

3.2.1 Tyylitiedosto CSS-kielellä

Kuten aikaisemmin on mainittu, SVG-dokumentin tyylin voi määritellä CSS-kielellä. Tämä on suotavaa silloin, jos useisiin elementteihin halutaan samanlainen muotoilu. Jos esimerkiksi kaikista ympyröistä halutaan keltaisia, joilla on punainen reunaviiva, CSS:llä voitaisiin määritellä seuraavasti:

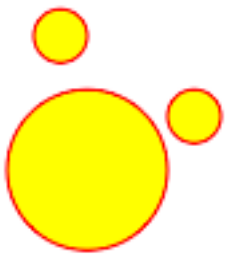
```
circle {
    fill: yellow;
    stroke: red;
}
```

Tyylitiedosto on lisättävä SVG-tiedostoon ennen `svg`-tagia:

```
<?xml-stylesheet href="tyylitiedosto.css" type="text/css"?>
```

Tämän jälkeen kaikki `circle`-elementit saavat tyylitiedoston mukaisen muotoilun, kuten kuva 12 näyttää.

```
<circle cx="50" cy="30" r="10" />
<circle cx="100" cy="60" r="10" />
<circle cx="60" cy="80" r="30" />
```



KUVA 12. Tyylitiedostolla muotoillut ympyrät

CSS tukee class-muotoiluja, joilla voidaan määritellä tarkemmin, mitä elementtiä tyyli koskee. Jos kaksi pientä ympyrää halutaan keltaisen sijaan punaiseksi, tyylitiedostoon tulee lisätä seuraava määritys:

```
circle.punainen {
    fill: red;
    stroke: red;
}
```

SVG-tiedostossa täytyy lisätä class-määreet pienille ympyröille:

```
<circle class="punainen" cx="50" cy="30" r="10" />
<circle class="punainen" cx="100" cy="60" r="10" />
```

Lopputulos on kuvan 13 mukainen:

KUVA 13. Class-määrittelyn käytön tulos

Tyylitiedostoa käytettäessä täytyy muistaa, että tyylitiedosto ylikirjoittaa SVG-tiedostossa määritellyn tyylin. Esimerkiksi yllä kuvatun CSS-tiedoston kanssa alla oleva koodi tulostaa keltaisen ympyrän, vaikka ympyrä onkin määritelty siniseksi SVG-tiedostossa:

```
<circle cx="50" cy="30" r="10" fill="blue" />
```

Elementille voi kuitenkin määrittää tyylin style-attribuuttiin, jolloin SVG-tiedostossa määritelty tyyli ylikirjoittaa CSS-tiedostossa määritellyn. Alla oleva koodi tulostaa vihreän ympyrän mustalla reunaviivalla, vaikka yllä kuvattu CSS-tiedosto olisikin linkitettynä:

```
<circle class="punainen" cx="100" cy="60" r="10" style="fill: green;
stroke: black;" />
```

`style`-attribuutin käyttö on mielestäni kuitenkin vähemmän suositeltavaa kuin tyylimääreiden kirjoittaminen CSS-tiedostoon tai tyyliattribuuttien käyttö suoraan elementissä. Näin siksi, että attribuuttien arvoja pääsee helpommin muuttamaan ohjelmallisesti, kun ne ovat omina attribuutteinaan sen sijaan, että ne olisivat yhtenä merkkijonona `style`-attribuutin arvona.

3.2.2 Ohjelmointi JavaScript-kielellä

JavaScript on oiva kieli SVG:n ohjelmointiin, koska sillä voi tehdä SVG-kuvaan muutoksia asiakaspäässä. Esimerkiksi hiiren tai näppäimistön painalluksiin reagoivat JavaScriptin tapahtumankäsittelijät sopivat mainiosti vuorovaikutteisen SVG-dokumentin tekemiseen. Kuten on sanottu, JavaScriptiä voi kirjoittaa suoraan SVG-tiedostoon `<![CDATA[ ]]>` -lohkon sisään, mutta ulkoisen JavaScript-tiedoston voi linkittää tiedostoon seuraavalla tavalla:

```
<script xlink:href="javascript.js" />
```

Koska SVG-dokumentti on XML:ää, sen rakenne voidaan käsittää puumaisena lapsi-, sisarus- ja vanhempielementeistä koostuvana kokonaisuutena. Tämän rakenteen kaikkiin elementteihin voi päästä käsiksi, mutta ensiksi on tiedettävä juurielementti, eli itse SVG-dokumentti. Juurielementtiin päästään käsiksi seuraavalla JavaScript-koodilla:

```
var svgdoc = evt.target.ownerDocument;
```

Muuttuja, johon juurielementti tallennetaan, kannattaa määritellä globaaliksi muuttujaksi. Yllä oleva koodi kannattaa suorittaa heti, kun SVG-dokumentti on ladannut. Tällöin SVG-dokumentin `svg`-tagi voisi näyttää seuraavalta (huomaa paksunnettu osa):

```
<svg id="root" width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg"
```

```
xmlns:xlink="http://www.w3.org/1999/xlink"
onload="init(evt)">
```

Ja JavaScript-tiedosto voisi alkaa seuraavalla koodilla:

```
var svgdoc = false;
var svgns = "http://www.w3.org/2000/svg";
var svgXlink = "http://www.w3.org/1999/xlink";

function init(evt) {
    svgdoc = evt.target.ownerDocument;
}
```

JavaScript-tiedoston alussa määritellään globaalit muuttujat `svgdoc`, joka saa arvokseen juurielementin, `svgns`, joka määrittää SVG:n nimiavaruuden ja `svgXlink`, jonka avulla voidaan tehdä linkkejä. `evt`-muuttuja, joka välitetään init-funktiolle, on SVG:n varattu sana. Se pitää sisällään kaikki ominaisuudet ja toiminnot, jotka kuvaavat tapahtunutta toimintoa. `evt`-muuttujan avulla saadaan esimerkiksi selville hiiren sijainti kullakin hetkellä. (Eisenberg 2002, 183.)

Kun SVG-dokumentin juurielementti on määritelty, sen avulla pääsee käsiksi mihin tahansa muuhun elementtiin. Elementtejä voi hakea id-arvon perusteella, tagin nimen perusteella tai niiden suhteiden perusteella muihin elementteihin. (Watt ym. 2003, 584, 588.) Alla olevista JavaScript-koodeista ensimmäisellä haetaan elementtejä id:n perusteella ja toisella tagin perusteella.

```
svgdoc.getElementById();
svgdoc.getElementsByTagName();
```

Kun elementtiin on päästy käsiksi, voidaan sille lisätä attribuutteja ja lukea sillä jo olevia attribuutteja. `getAttribute()`-metodi palauttaa parametrina määritellyn attribuutin arvon ja `setAttribute()`-metodi asettaa ensimmäisenä parametrina määritellylle attribuutille arvon, joka määritellään metodin toisena parametrina. (Watt ym.

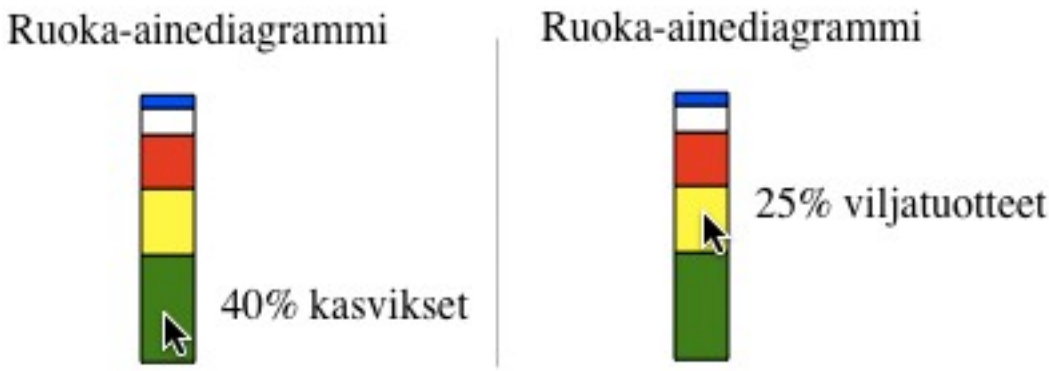
2003, 589.) Attribuuttien lukeminen ja kirjoittaminen onnistuu seuraavalla JavaScript-koodilla:

```
var elementti = svgdoc.getElementById("ympyra");
var attribuutti = elementti.getAttributeNS(null, "r");
elementti.setAttributeNS(null, "r", attribuutti + 10);
```

Metodien perässä olevat kirjaimet NS tarkoittavat nimiavaruutta. Tällöin metodille tulee antaa ensimmäiseksi parametriksi nimiavaruus, jonka voi määrittää myös null-arvoksi. Elementtien luonnissa tarvitaan myös nimiavaruutta, jolloin sille on annettava arvo. Tässä tapauksessa arvo on määritelty globaaliin `svgns`-muuttujaan.

```
svgdoc.createElementNS(svgns, "circle");
```

Kuten aikaisemmin on mainittu, SVG-tiedoston saa reagoimaan hiiren ja näppäimistön lähettämiin syötteisiin. Tähän käytetään JavaScriptin tuttuja tapahtumanäksitteli-jöitä, kuten onclick, onmousover ja onkeypress. Näiden ominaisuuksien käytöstä on koodiesimerkki liitteessä 1. Koodi tuottaa kuvan 14 mukaisen kuvion, jossa teksti seuraa hiirtä.



KUVA 14. Tapahtumankäsittelijän avulla teksti seuraa hiirtä

3.2.3 Animointi SMIL-kielellä

SVG-tiedostoa voi muokata mielivaltaisesti esimerkiksi JavaScriptillä, mutta SVG-kuvien animointiin SMIL-kieli (lausutaan kuten *smile* (engl.)) on kaikkein sopivin.

SMIL on W3C:n suosittelema kieli, joka pohjautuu SVG:n tapaan XML-kieleen. Animoinnille on määritelty viisi eri elementtiä:

- `animate`: yleiseen elementtien animointiin
- `animateColor`: värien animointiin
- `animateMotion`: polkujen mukaan kulkevan liikkeen animointiin
- `animateTransform`: `transform`-attribuutin animointiin
- `set`: arvojen muutokseen ilman välivaiheita (Watt ym. 2003, 426 – 427.)

Animointi liitetään animoitavaksi halutun elementin lapsielementiksi:

```
<circle ...>
    <animate ... />
</circle>
```

Tai käyttäen linkitystä:

```
<animate xlink:href="#elementID" ... />
```

(Watt ym. 2003, 428.)

Kukin animointi-elementti voi koskea vain yhtä attribuuttia animoitavasta elementistä. Tämä attribuutti määritellään animointi-elementtiin `attributeName`-attribuutilla, ja sen arvoksi annetaan mikä tahansa animoitavan elementin attribuutti. `attributeType`-attribuutilla voidaan määritellä onko animoitava attribuutti CSS:n (arvoksi `css`) vai XML:n (arvoksi `xml`) attribuutti. Oletuksena tämän attribuutin arvona on `auto`, jolloin animoitavaa attribuuttia etsitään ensin CSS:n attribuuttien joukosta ja sitten XML:stä. (Watt ym. 2003, 428 – 429.)

Aika on animaatioissa tärkeä tekijä. SVG:ssä on sisäänrakennettu kello, joka käynnistyy SVG-dokumentin latauduttua ja lakkaa käymästä vasta, kun käyttäjä poistuu SVG-dokumentista. Ajan voi ilmaista usealla eri tavalla:

- tunneissa, minuuteissa ja sekunteissa `1:30:40`, tai minuuteissa ja sekunteissa `2:25`

- yksittäisenä arvona ajanmääreen kanssa, joista `h` tarkoittaa tuntia, `min` minuutteja, `s` sekunteja ja `ms` millisekunteja (ajanmääre tulee sijoittaa heti numeerisen arvon perään ilman välilyöntiä, esim. `6s`)
- relatiivisena arvona edellisen animaation alkamiseen tai loppumiseen, kuten `ensimmaisenId.end` tai `ensimmaisenId.begin + 5s`. Vaikka relatiivisiin arvoihin voi lisätä aikaa, sitä ei voi vähentää (esim. `ensimmaisenId.end - 2s`), sillä SVG ei voi tietää milloin toinen animaatio loppuu tai alkaa ennen kuin se on loppunut tai alkanut (Eisenberg 2002, 175.)

Animaation käynnistysajankohta määritellään `begin`-attribuutilla, joka voi saada arvoikseen mitä tahansa edellä kuvatuista ajanmääreistä. Lisäksi sille voi antaa useita ajankohtia puolipisteellä eroteltuina, jolloin animaatio aloitetaan uudestaan jokaisessa ajankohdassa. On huomioitava, että vaikka arvot erotellaan puolipistein, viimeisen arvon perään ei tule laittaa puolipistettä. Arvon ei kuitenkaan tarvitse olla mikään ajanmääre, vaan attribuutti voi saada arvokseen jonkin tapahtumankäsittelijän, kuten `begin="click"` tai `begin="toisenElementinId.click"`. Tällöin animaatio aloitetaan, kun elementtiä tai toista id:llä määriteltyä elementtiä klikataan. (Watt ym. 2003, 429 – 430.)

Animaation kesto määritellään `dur`-attribuutilla, joka niinikään voi saada arvokseen minkä tahansa ajanmääreen. Animaation voi lopettaa `end`-attribuutilla, jolle voi antaa samoja arvoja kuin `begin`-attribuutillekin. Animaation toistumiskertojen lukumäärän voi määritellä `repeatCount`-attribuutilla. Sen arvoksi voi antaa numeeristen arvojen lisäksi `indefinite`, jolloin animaatiota toistetaan kunnes käyttäjä poistuu SVG-dokumentista. Sen, saako animaation aloittaa uudestaan, voi määritellä `restart`-attribuutilla. Se voi saada arvokseen `whenNotActive`, `always` tai `never`. Tällöin animaation saa käynnistää, jollei se ole jo käynnissä, käynnistää milloin vain uudestaan tai sitä ei koskaan saa käynnistää uudestaan. (Watt ym. 2003, 430 – 431.)

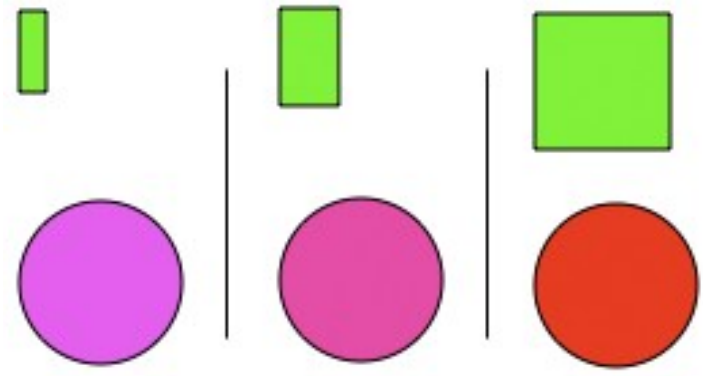
Animaation loputtua kuva palaa alkuperäiseen tilaansa, ellei sille määrittele `fill`-attribuutin arvoksi `freeze`. Animointi-elementeissä `fill`-attribuutilla ei tarkoiteta täyttöä, vaan miten loppuaika animaation jälkeen täytetään – palautetaanko elementti al-

kuperäiseen muotoonsa vai pidetäänkö muutokset. remove-arvolla elementti palautetaan alkuperäiseksi. (Watt ym. 2003, 431.)

Animoinnin tarkoituksena on tietysti muuttaa elementin tilaa jostakin johonkin toiseen. Tämän voi määritellä from- ja to-attribuuteilla. from-attribuutti määrittelee alkutilanteen arvon ja to-attribuutti lopputilanteen arvon. (Watt ym. 2003, 432.) Suorakulmion kokoa ja ympyrän väriä voisi muokata seuraavalla koodilla:

```
<rect x="10" y="10" width="10" height="30" fill="#33ff33"
stroke="black">
    <animate attributeName="width" begin="0s" dur="3s" to="50"
fill="freeze" />
    <animate attributeName="height" begin="0s" dur="3s" to="50"
fill="freeze" />
</rect>
<circle cx="40" cy="110" r="30" fill="#ee33ee" stroke="black">
    <animateColor attributeName="fill" begin="0s" dur="3s"
from="#ee33ee" to="red" fill="freeze" />
</circle>
```

Kuva 15 näyttää animaation tilanteen eri vaiheissa: vasemmanpuoleisin kuva on animaatiosta nollan sekunnin kohdalla, keskimmäinen puolessa välin animaatiota ja oikeanpuoleisin kuva kolmen sekunnin kohdalla, eli kun animaatio on päättynyt.



KUVA 15. Animoidun SVG-kuvan vaiheet

set-elementillä voi muuttaa elementtien arvoja ilman välivaiheita. Se ei tarvitse fill-attribuuttia, sillä set-elementti säilyttää tilansa, ellei sille määrittele end-attribuuttia. (Watt ym. 2003, 433 – 434.)

4 VEKTORIGRAFIIKKAFORMAATTIEN VERTAILU

SVG:n selkeänä kilpailijana netissä ovat Flash ja Flex. Nämä kaikki kuvamuodot tuottavat vektorigrafiikkaa, jota voi animoida ja josta voi saada käyttäjän kanssa vuorovaikutteisen elementin. Kullakin formaatilla on omat etunsa ja haittansa, mutta jokainen on varsin pätevä omalla alueellaan. Myöhemmin tässä luvussa vertaillaan SVG:n ja Flashin ominaisuuksia.

4.1 Flash ja Flex

Flashilla voidaan luoda nettisivuille vuorovaikutteisia elementtejä, kuten videoita, animaatioita tai koko nettisivun. Flashia tehdäkseen ei tarvitse osata koodata, sillä Flash-animaatiot voidaan tehdä Adobe Flash -ohjelmalla, vaikkakin Flashia voidaan myös koodata. Ohjelma tuottaa .swf-tiedostoja, jotka voi sisällyttää nettisivuille. Flashia katsoakseen käyttäjät tarvitsevat Adoben sivuilta ilmaiseksi ladattavan Flash Playerin. (W3Schools.)

Myös Adobe Flexillä voi luoda .swf-tiedostoja. Se on kuitenkin suunnattu enemmän ohjelmoijille kuin Adobe Flash, sillä Flexillä tehdäkseen täytyy osata ohjelmoida – sitä ei voi luoda graafisesti (Adobe Creative Team 2008, 12). Käyttöliittymä ohjelmoidaan XML-kieleen pohjautuvalla MXML-kielellä ja vuorovaikutteisuus JavaScriptiin pohjautuvalla ActionScript-kielellä. Tätä kieltä voi käyttää myös Flashin vuorovaikutteisuuden ohjelmointiin. (Adobe.) Flex on lisäksi yhteensopiva minkä tahansa palvelinpuolen ohjelmointikielen kanssa. Näin ollen Flexillä voi ohjelmoida vain käyttöliittymän ja keskittyä muun toiminallisuuden tekemiseen sillä kielellä, minkä itse parhaiten taitaa. (Patrick, 2008.)

Flashin käyttämistä nettisivuilla on kritisoitu paljon. Sen on sanottu heikentävän käytettävyyttä ja häiritsevän sivuston ydinsisällön hahmottamista. Joillakin sivustoilla käytetyt Flash-introt rikkovat internetin periaatetta siitä, että käyttäjä päättää itse min-

ne ja milloin hän seuraavaksi menee. Introt pakottavat käyttäjän vain katsomaan ja odottamaan, vaikka onneksi suurin osa introista on mahdollista sivuuttaa jonkin linkin painalluksella. (Nielsen 2000.) Nettisivuista saa hyvin sekavat Flashilla, jos sitä ei osaa käyttää. Koska Flashilla on melko helppo tehdä pyöriviä, vilkkuvia ja soivia elementtejä, niitä myös tehdään. (Nielsen 2000; Truese 2001.)

Flash on kuitenkin hyvin voimakas työkalu, kun sitä osaa käyttää. Sillä voidaan tehdä viihdyttäviä ja tehokkaita elementtejä, joilla esimerkiksi yritykset pitävät yllä imagoaan. Lisäksi Flash on laajalle levinnyt formaatti ja sitä voidaan katsoa lähes jokaisessa tietokoneessa. (Truese 2001.)

4.2 Vertailu

SVG ja Flash ovat olleet kilpailevia formaatteja jo kauan, vaikka SVG onkin jäänyt melko tuntemattomaksi. Silti lähes kaikissa selaimissa on nykyään natiivi tuki SVG:lle, jolla on valmiudet haastaa Flash. Seuraava vertailu on koottu Artymiakin (2002) ja Jacksonin (2002) artikkeleista, ellei toisin ole mainittu.

Flashia tehdäkseen täytyy omistaa Adobe Flash -kehitysympäristö. SVG-tiedoston tekemiseen ei tarvita mitään tiettyä ohjelmaa, eikä sen tarvitse edes olla erityinen kehitysympäristö, sillä tavallinen tekstieditori riittää SVG-tiedoston kirjoittamiseen. SVG:n yksi vahvuus Flashiin verrattuna onkin sen avoimuus ja XML-kieleen pohjautuminen. Näin se on laajennettavissa mielivaltaisesti. Toki Flashiinkin voi lisätä omia laajennuksia, mutta Adoben Flash Player ei välttämättä kykene toistamaan tällaisia tiedostoja. SVG-tiedostoja on myös helppo liikuttaa eri ohjelmien välillä, jotka tukevat XML:ää (Eisenberg 2002, 17).

Koska SVG perustuu XML:n, se tallennetaan tekstimuodossa, joka on myös ihmiselle luettavaa. Flash-tiedostot sen sijaan eivät ole lähellekään ihmisen luettavissa. Tarpeellisuus kuvan havainnointiin tekstin kautta voidaan tietysti asettaa kyseenalaiseksi, mutta tällekin on puolustus: mikä olisi sen parempi tapa oppia itse, kuin lukea toisten

tekemiä sovelluksia? Ja kun lähdekoodi on esillä, siihen voi tehdä muutoksia. Jos käyttäjä on esimerkiksi värisokea, hän voi muokata lähdekoodin värimaailmaa itselleen sopivammaksi.

Sekä Flash- että SVG-tiedostoihin voi hakea tietoa tietokannasta tai luoda dynaamisesti koko kuvan. Molemmilla tiedostomuodoilla on myös yksi alusta, jota ne käyttävät. Flashilla se on sidotumpi, koska tiedostojen on sopeuduttava Adobe Playerin vaatimuksiin, jotta ne näkyvät kaikille käyttäjille. SVG:lle on kirjoitettu tarkka spesifikaatio, josta on kuitenkin mahdollista poiketa. Käyttäjät luultavasti suosivat kuitenkin spesifikaation mukaisia sovelluksia, sillä ne ovat standardin mukaisia ja siten helpompia muokata ja integroida olemassa oleviin järjestelmiin.

Sekä Flashilla että SVG:llä on yksi ominaisuus yli toisen: Flash kykenee toistamaan ääntä ja videota huomattavasti paremmin kuin SVG, mutta hakukoneet löytävät helpommin SVG- kuin Flash-tiedostoja. Vaikka Flash on tällä hetkellä suositumpi formaatti vektorigrafiikan luomiseen, SVG on tulossa hyvää vauhtia perässä. Kummallakin formaatilla on omat vahvuutensa, joten käyttötarkoitus määrää kumpi on kussakin tilanteessa sopivampi.

5 ESIMERKKITAPAUS

Esimerkkitapauksen toimeksiantaja on helsinkiläinen Enkora Oy Ltd. Enkora on järjestelmäratkaisujen tarjoaja, jonka päätuote on internetpohjainen kulunvalvontasovellus. Esimerkkitapauksen aihe on toteuttaa selaimella käytettävä sovellus, jolla voidaan piirtää pohjapiirroksia ja sijoittaa kameroita haluttuihin paikkoihin. Sovelluksen avulla nopeutetaan nykyistä prosessia pohjapiirrosten tekemisessä ja mahdollistetaan myös asiakkaiden osallistuminen pohjapiirrosten tekemiseen.

Esimerkkitapaus päätettiin tehdä SVG:llä, sillä pohjapiirroksen on kyettävä kommunikoimaan järjestelmän muiden osien kanssa. Kuten SVG:stä on jo kerrottu, se pohjautuu XML-kieleen ja on siten hyvin yhteensopiva minkä tahansa sovelluksen tai ohjel-

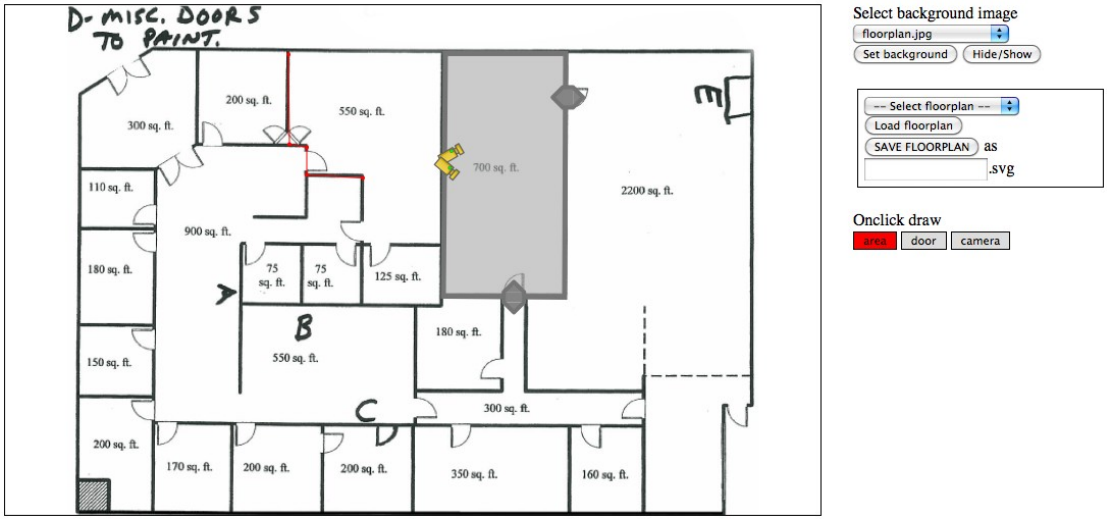
mointikielen kanssa, joka voi käsitellä XML:ää. Flash taas on binäärimuotoista dataa, jonka integrointi muihin järjestelmiin on hankalaa. Esimerkkitapauksen tuottama sovellus annetaan mahdollisesti myös asiakkaiden käyttöön, jolloin toimiminen useissa eri ympäristöissä on oltava mahdollista. Flash on hyvin laajalle levinnyt formaatti, mutta selainten natiivi tuki SVG:lle ei estänyt SVG:n käyttämistä.

5.1 Esimerkkitapauksen ominaisuudet

Esimerkkitapaus koostuu yhdestä SVG- ja JavaScript-tiedostosta sekä muutamasta PHP-tiedostosta. Tietokantaa ei käytetty, sillä ainoa tallennettava tieto ovat erilaiset pohjapiirrokset, jotka tallennetaan erillisiin .svg-tiedostoihin. Lisäksi on index-tiedosto, johon SVG-tiedosto on sisällytetty ja jossa sijaitsevat sen hallintaan tarvittavat toiminnot.

SVG-tiedosto ei sisällä paljoakaan koodia, ainoastaan nimiavaruuden määrittelyt ja muutaman JavaScript-tiedoston laukaisevan tapahtumankäsittelijän. Myös index-tiedostossa on painonappeja, jotka kutsuvat funktioita JavaScript-tiedostosta. Index-tiedostossa voi ladata kuvan palvelimelle, josta sen voi valita pohjapiirrokseen taustakuvaksi piirtämisen helpottamiseksi. Tiedoston lataaminen suoritetaan PHP:llä, mutta kaikki muu toiminnallisuus toteutetaan JavaScript-tiedostossa, joka kutsuu tarvittaessa PHP-tiedostoja suorittamaan esimerkiksi pohjapiirroksen tallentamisen tiedostoon.

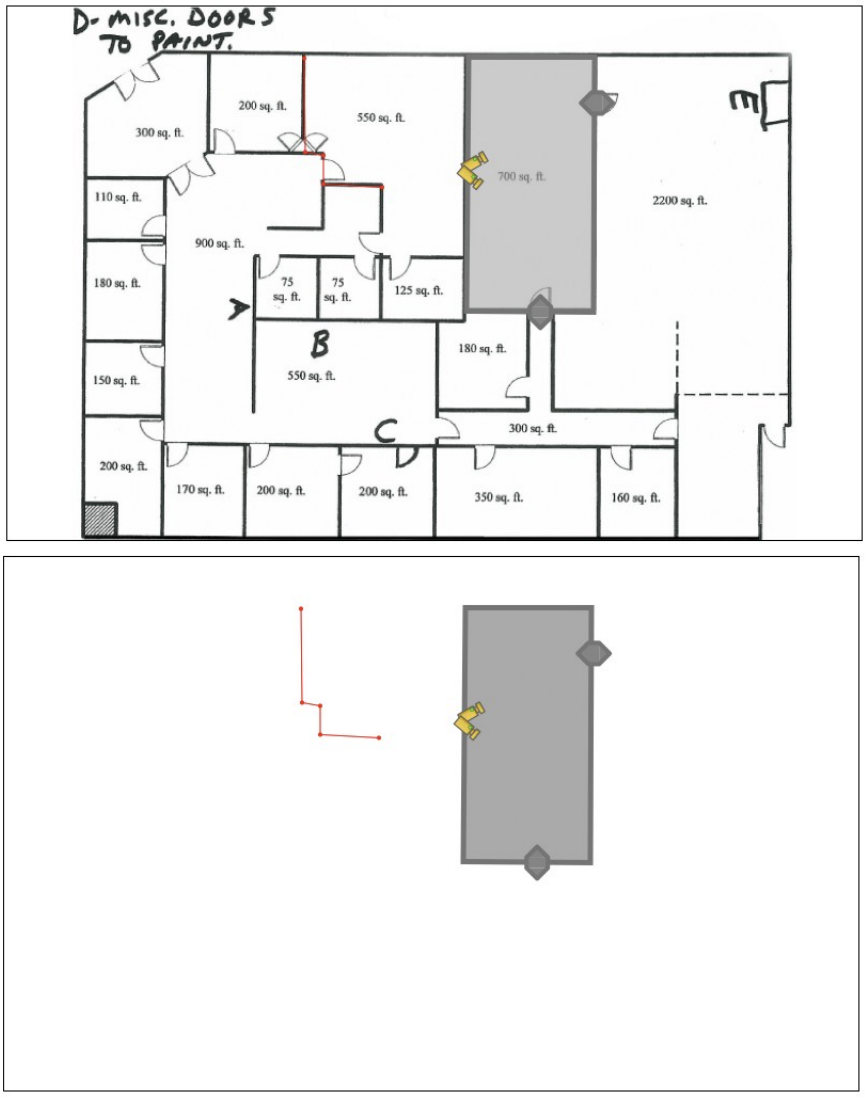
Kuvasta 16 voi nähdä sovelluksen ulkoasun. SVG-osio on vasemmalla ja hallintatyökalut oikealla. SVG:llä on piirretty yksi huone ja siihen kaksi kameraa ja ovea sekä aloitettu toisen huoneen piirtämistä punaisella viivalla valmiin huoneen vasemmalle puolelle. Hallintatyökalujen ylimmästä alasvetovalikosta voi valita taustakuvan ja sen alla olevista painonapeista piilottaa tai näyttää taustakuvan.Seuraava, laatikoitu alasvetovalikko antaa mahdollisuuden valita tallennettu pohjapiirros tai tallentaa pohjapiirros haluamallaan nimellä. Alimmista painonapeista voi valita piirtotyökalun. Aktiivinen työkalu on punaisella, tässä tapauksessa huonetyökalu. Käyttäjä voi myös valita ovi- tai kameratyökalun, jolloin klikkaus aiheuttaa pisteen sijaan oven tai kameran luonnin.



KUVA 16. Sovelluksen ulkoasu

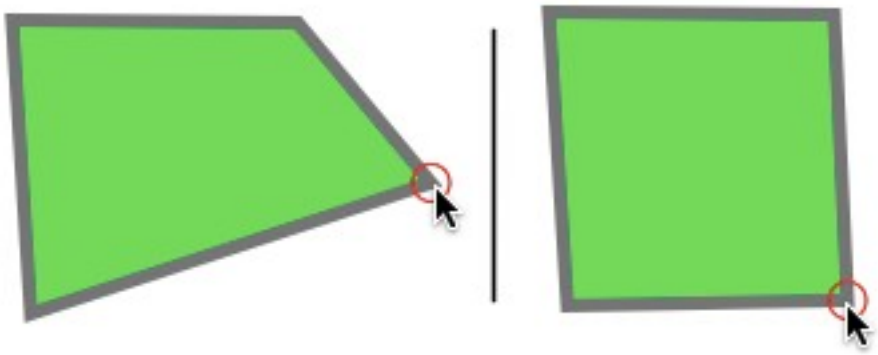
Pohjapiirroksen piirtäminen aloitetaan valitsemalla taustakuvaksi jpg-kuva valmiista pohjapiirroksesta. Vaihtoehtoisesti käyttäjä voi ladata jo aikaisemmin piirretyn ja tallennetun pohjapiirroksen ja jatkaa sen muokkaamista. Kun pohjapiirros on ladattu taustalle, sen voi tarvittaessa piilottaa. Tällöin käyttäjä voi nähdä tekemänsä pohjapiirroksen ilman taustakuvaa.

Piilotetun taustakuvan saa yhdellä painalluksella takaisin näkyville, jolloin piirretyt elementit tulevat hieman läpinäkyviksi, jotta esimerkiksi huoneiden rajat ja ovet näkyvät kuvasta elementtien alta. Taustakuvaksi voi myös valita täysin toisen taustakuvan missä tahansa vaiheessa. Kuvassa 17 ylempi osa kuvaa sovelluksen SVG-osaa pohjapiirroksen kanssa ja alempi ilman pohjapiirrosta. Harmaasta huoneesta voi huomata, että pohjapiirroksen kanssa elementti on hieman läpinäkyvä, mutta ilman pohjapiirrosta läpinäkymätön.



KUVA 17. Taustakuvan näyttäminen ja piilottaminen

Taustakuvan valinnan jälkeen pohjapiirrokseen voi piirtää huoneen valitsemalla ”area”-työkalun ja klikkaamalla pisteet huoneen kulmiin. Pisteiden välille piirretään automaattisesti viiva, kunnes klikataan ensimmäistä pistettä. Tällöin hahmotelma muutetaan monikulmioksi. Kulmapisteitä voi liikutella sen jälkeen, kun monikulmio on luotu. Kun hiiren vie jonkin kulman lähelle, kulmaan tulee punainen ympyrä korostamaan valittua kulmaa. Ympyrän sisältä klikkaamalla ja raahaamalla voi siirtää kulman haluttuun paikkaan. Tämän toiminnon mahdollistava koodi on liitteessä 2 ja kuva 18 esittää toiminnon.



KUVA 18. Huoneen kulman voi siirtää raahaamalla sitä punaisen ympyrän sisältä

Kameroita ja ovia voi luoda valitsemalla halutun työkalun ja klikkaamalla siihen kohtaan, mihin objektin haluaa. Koska sekä kameran että oven kuvien koodi on kopioitu kuvankäsittelyohjelmalla tehdyistä objekteista, näiden luominen ei vaadi yhtä monimutkaista prosessia kuin huoneen luominen. Klikattuun kohtaan yksinkertaisesti ilmestyy valmis kuva kamerasta tai ovesta.

Objektit – huoneet, ovet ja kamerat – muuttuvat aktiiviseksi, kun niitä klikataan. Aktiivinen objekti on korostettu vihreällä. Objekti, jonka päällä hiirtä pidetään klikkaamatta sitä, korostetaan oranssilla. Aktiivista objektia voi liikutella drag and drop -menetelmällä, eli pitämällä hiirtä pohjassa ja siirtämällä objekti haluttuun kohtaan. Aktiivisen objektin tiedot näkyvät pohjapiirroksen oikealla puolella. Oville ja kameroille voi siellä määritellä asteluvun, jonka verran objektia käännetään akselinsa ympäri. Näiden tietojen joukossa on myös painonappi, jolla objektin voi tuhota.

5.2 Esimerkkitapauksen arviointi

Tutkimusongelmani oli määrittää, soveltuuko esimerkkitapauksen toteutukseen paremmin SVG vai Flash. Toteutukseen päätettiin valita SVG, sillä se on helpompi integroida muiden järjestelmien kanssa sen XML-kieleen perustumisen vuoksi. Sovelluksen tekeminen SVG:llä onnistui erinomaisesti. Sovellus täyttää kaikki sille asetetut vaatimukset, joten SVG:n valinta oli hyvä. Verrokkisovellusta ei tehty Flashilla, mutta SVG:llä toteutettu oli tarpeeksi hyvä, ettei verrokkia tarvittu.

Esimerkkisovelluksessa käytettiin SVG:n ominaisuuksia monipuolisesti. Ainoastaan animointia ei käytetty, mutta lähes kaikkia SVG:n peruselementtejä käytettiin ja niitä ohjelmointiin JavaScriptillä. Kaikki SVG:n objektit koodattiin JavaScriptillä, mutta ovien ja kameroiden koodi on kopioitu kuvankäsittelyohjelmalla luoduista SVG-kuvista. Kuten aikaisemmin on mainittu, SVG:tä voi luoda joko kirjoittamalla tai piirtämällä, mutta piirtämisenkin pohjimmaisena tuloksena on XML-muotoista tekstiä. Niinpä valmiiden ovien ja kameroiden sisällyttäminen esimerkkisovellukseen onnistui vaivatta – vain kopioimalla XML-data.

Esimerkkitapaus on hyvin käytettävissä jo sellaisenaan. Kaikilla sovelluksilla on kuitenkin mahdollisuus kehittyä, eikä esimerkkitapauksena tehty sovellus ole poikkeus. Sovelluksen ominaisuuksia voisi laajentaa. Nykyisellään huoneisiin ei voi lisätä uusia kulmia, kun huone on tehty valmiiksi. Sovellukseen voisikin tehdä päivityksen, joka mahdollistaisi uuden kulman luomisen esimerkiksi tuplaklikkaamalla huoneen reunaa. Lisäksi kuvankäsittelyohjelmista tuttu ryhmitystoiminto voisi olla hyödyllinen. Näin huonetta ja sen sisältäviä kameroita ja ovia voisi siirtää yhdellä kerralla paikasta toiseen. Nykyisessä versiossa kaikkia objekteja täytyy siirtää yksitellen.

Ongelmasta ”kumpi soveltuu käyttötarkoitukseen paremmin, SVG vai Flash”, ei voi sanoa mitään yleistä ratkaisua. Jokaisen sovelluksen kohdalla on erikseen mietittävä, mitkä ovat sovelluksen tavoitteet ja käyttötarkoitus, joiden pohjalta valitaan SVG:n tai Flashin käyttäminen. Flash on tunnetumpi ja siten enemmän käytetty kuin SVG, mutta kuten esimerkkitapaus näyttää, myös SVG:llä voi tehdä vuorovaikutteisen sovelluksen, joka toimii internetissä ja on käytettävyydeltään selkeä ja yksinkertainen.

6 PÄÄTÄNTÖ

Mielestäni opinnäytetyötä oli mukavaa ja sopivan haastavaa tehdä. SVG oli minulle täysin uusi formaatti, johon tutustuin syvällisesti kirjoittamalla siitä teoriaosuuden ja soveltamalla teoriaa käytäntöön. Flash oli minulle entuudestaan tuttu, mutta Flexin löysin vasta opinnäytetyötä tehdessäni. Näin on asia suuremmankin käyttäjämäärän

osalta: Flash tunnetaan, mutta SVG:tä ei, vaikka se on ollut W3C:n standardoima jo lähes kymmenen vuotta. Tästä opinnäytetyön nimikin juontuu – SVG on vanha formaatti, mutta silti monille aivan uusi ja tuntematon.

SVG:n ja Flashin eroja oli mielenkiintoista selvittää. Näilläkin formaateilla oli omat innokkaat kannattajansa, kuten millä tahansa muulla lähes samaan tarkoitukseen tehdyillä ratkaisuilla – Windows ja Unix, Firefox ja Internet Explorer... Näitä vastakkainasetteluja löytyy it-maailmasta loputtomiin. Yhteistä niille on, että kaikilla niillä on omat hyvät ja huonot puolensa kilpailijaansa vastaan, aivan kuten SVG:llä ja Flashillakin.

Jos todella halutaan selvittää, kumpi käsitellyistä formaateista soveltuisi parhaiten esimerkkitapaukseen, voitaisiin tehdä sovellus myös Flashilla ja tehdä kysely sovellusten käyttäjien kokemuksista. Tällainen selvitys ei kuitenkaan kuulu tämän opinnäytetyön tavoitteisiin. SVG:stä saa varmasti tehtyä useita erilaisia selvityksiä ja sovelluksia, sillä se on vielä tavallisille käyttäjille tuntematon formaatti. Olisi hyvin mielenkiintoista vertailla SVG:llä ja Flashilla toteutettuja nettipelejä keskenään. Flashillahan on tehty pelejä mielin määrin, mutta SVG:llä ei tietääkseni ainoatakaan, vaikka se olisi siihen hyvin kykenevä SMIL-animointikielen ja JavaScriptin avulla.

Opin SVG:n hyvin tehdessäni tätä työtä ja mielestäni se soveltuu käyttöön aivan yhtälailla kuin Flashin. Toivonkin, että teoriaosuudessa tarkasti esittelemäni SVG saa uusia käyttäjiä tämän opinnäytetyön myötä!

LÄHTEET

Adobe 2009. Flex overview. WWW-dokumentti. http://www.adobe.com/products/flex/overview/. Luettu 8.12.2009.

Adobe Creative Team 2008. Adobe Flash CS4 Professional Classroom in a Book. Adobe Press.

Artymiak, Jacek 2002. SWF Is Not Flash (and Other Vectored Thoughts). WWW-dokumentti. http://www.oreillynet.com/pub/a/javascript/2002/05/24/swf_not_flash.html. Päivitetty 21.5.2002. Luettu 8.12.2009.

Eisenberg, J. David 2002. SVG Essential. O'Reilly Media.

Jackson, Dean 2002. SVG On the Rise. WWW-dokumentti. http://www.oreillynet.com/pub/a/javascript/2002/06/06/svg_future.html. Päivitetty 6.6.2002. Luettu 10.11.2009.

Mertz, David 2005. XML Matters: Program with SVG. WWW-dokumentti. http://www.ibm.com/developerworks/xml/library/x-matters40/. Päivitetty 15.4.2005. Luettu 10.11.2009.

Moock, Colin 1999. svg: is it flashier than flash? WWW-dokumentti. http://www.moock.org/webdesign/svg/articles/svg-vs-flash.html. Päivitetty 15.2.2000. Luettu 10.11.2009.

Nielsen, Jakob 2000. Flash: 99% Bad. WWW-dokumentti. http://www.useit.com/alertbox/20001029.html. Pävitetty 29.10.2000. Luettu 8.12.2009.

Patrick, Ted 2008. What is Flex? WWW-dokumentti. http://onflash.org/ted/2008/01/what-is-flex.php. Päivitetty 17.1.2008. Luettu 8.12.2009.

Truese, Michael 2001. Flash: 99% Good. WWW-dokumentti. http://articles.sitepoint.com/article/flash-99-good. Päivitetty 3.4.2001. Luettu 8.12.2009.

W3Schools. Flash Tutorial. WWW-dokumentti. http://www.w3schools.com/Flash/default.asp. Luettu 8.12.2009.

Watt, Andrew, Lilley, Chris, Ayers, Daniel J., George, Randy, Wenz, Christian, Hauser, Tobias, Lindsey, Kevin & Gustavsson, Niklas 2003. SVG Unleashed. Indianapolis: Sams Publishing.

LIITE 1(1).

Esimerkki tapahtumankäsittelijöiden käytöstä

Esimerkki tapahtumankäsittelijöiden käytöstä.

SVG-tiedosto:

```
<svg id="root" width="100%" height="100%" version="1.1"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
onload="init(evt)">
          <script xlink:href="js.js" />
</svg>
```

JavaScript-tiedosto:

```
var svgdoc = false;
var svgns = "http://www.w3.org/2000/svg";
var svgXlink = "http://www.w3.org/1999/xlink";

function init(evt) {
          svgdoc = evt.target.ownerDocument;
          luoDiagrammi(evt);
}

function luoDiagrammi(evt) {
          //Otsikko
          var otsikko = svgdoc.createElementNS(svgns, "text");
          otsikko.setAttributeNS(null, "x", 5);
          otsikko.setAttributeNS(null, "y", 16);
          var otsikkoteksti = svgdoc.createTextNode("Ruoka-ainediagrammi");
          svgdoc.documentElement.appendChild(otsikko);
          otsikko.appendChild(otsikkoteksti);

          //Diagrammi
          var herkut = svgdoc.createElementNS(svgns, "rect");
          herkut.setAttributeNS(null, "x", "50");
          herkut.setAttributeNS(null, "y", "30");
          herkut.setAttributeNS(null, "width", "20");
          herkut.setAttributeNS(null, "height", "5");
          herkut.setAttributeNS(null, "fill", "blue");
          herkut.setAttributeNS(null, "stroke", "black");
          herkut.setAttributeNS(null, "ruokaAine", "herkut");
          svgdoc.documentElement.appendChild(herkut);
          lisaaTapahtumat(herkut);

          var maito = svgdoc.createElementNS(svgns, "rect");
```

```
            maito.setAttributeNS(null, "x", "50");
            maito.setAttributeNS(null, "y", "35");
            maito.setAttributeNS(null, "width", "20");
            maito.setAttributeNS(null, "height", "10");
            maito.setAttributeNS(null, "fill", "white");
            maito.setAttributeNS(null, "stroke", "black");
            maito.setAttributeNS(null, "ruokaAine", "maito");
            svgdoc.documentElement.appendChild(maito);
            lisaaTapahtumat(maito);

            var liha = svgdoc.createElementNS(svgns, "rect");
            liha.setAttributeNS(null, "x", "50");
            liha.setAttributeNS(null, "y", "45");
            liha.setAttributeNS(null, "width", "20");
            liha.setAttributeNS(null, "height", "20");
            liha.setAttributeNS(null, "fill", "red");
            liha.setAttributeNS(null, "stroke", "black");
            liha.setAttributeNS(null, "ruokaAine", "liha");
            svgdoc.documentElement.appendChild(liha);
            lisaaTapahtumat(liha);

            var vilja = svgdoc.createElementNS(svgns, "rect");
            vilja.setAttributeNS(null, "x", "50");
            vilja.setAttributeNS(null, "y", "65");
            vilja.setAttributeNS(null, "width", "20");
            vilja.setAttributeNS(null, "height", "25");
            vilja.setAttributeNS(null, "fill", "yellow");
            vilja.setAttributeNS(null, "stroke", "black");
            vilja.setAttributeNS(null, "ruokaAine", "vilja");
            svgdoc.documentElement.appendChild(vilja);
            lisaaTapahtumat(vilja);

            var kasvikset = svgdoc.createElementNS(svgns, "rect");
            kasvikset.setAttributeNS(null, "x", "50");
            kasvikset.setAttributeNS(null, "y", "90");
            kasvikset.setAttributeNS(null, "width", "20");
            kasvikset.setAttributeNS(null, "height", "40");
            kasvikset.setAttributeNS(null, "fill", "green");
            kasvikset.setAttributeNS(null, "stroke", "black");
            kasvikset.setAttributeNS(null, "ruokaAine", "kasvikset");
            svgdoc.documentElement.appendChild(kasvikset);
            lisaaTapahtumat(kasvikset);
}

function lisaaTapahtumat(elementti) {
```

```
            elementti.setAttributeNS(null, "onmouseover", "naytaInfo(evt)");
            elementti.setAttributeNS(null, "onmousemove", "liikutaInfoa(evt)");
            elementti.setAttributeNS(null, "onmouseout", "poistaInfo(evt)");
}

function naytaInfo(evt) {
            var osio = evt.target;
            var prosentti = osio.getAttributeNS(null, "height");
            var ruokaAine = osio.getAttributeNS(null, "ruokaAine");

            var tekstiElementti = svgdoc.createElementNS(svgns, "text");
            tekstiElementti.setAttributeNS(null, "id", ruokaAine);
            tekstiElementti.setAttributeNS(null, "x", 80);
            tekstiElementti.setAttributeNS(null, "y", evt.clientY);
            svgdoc.documentElement.appendChild(tekstiElementti);

            var teksti = svgdoc.createTextNode(prosentti + "% " + ruokaAine);
            tekstiElementti.appendChild(teksti);
}

function liikutaInfoa(evt) {
            var osio = evt.target;
            var id = osio.getAttributeNS(null, "ruokaAine");
            var teksti = svgdoc.getElementById(id);
            teksti.setAttributeNS(null, "y", evt.clientY);
}

function poistaInfo(evt) {
            var osio = evt.target;
            var id = osio.getAttributeNS(null, "ruokaAine");
            var teksti = svgdoc.getElementById(id);
            if (teksti) teksti.parentNode.removeChild(teksti);
}
```

LIITE 2.

Esimerkkisovelluksen kulmien liikutus

Esimerkkisovelluksen koodi, jolla huoneen kulmia voi liikuttaa.

```
function mouseMove(evt) {
    if (mDown && myMovableObject) {
        var x = evt.clientX;
        var y = evt.clientY;
        var id = myMovableObject.getAttributeNS(null, "id");
        var objectId = id.substr(0, 1);

        if (id.substr(0,3) == "hcp") { //ROOM CORNER
            var polygon = svgdoc.getElementById(lastClickedId);
            var pointsString = polygon.getAttributeNS(null, "points");
            var points = pointsString.split(" ");

            var cornerX = myMovableObject.getAttributeNS(null, "cx");
            var cornerY = myMovableObject.getAttributeNS(null, "cy");

            for (var i = 0; i < points.length; i++) {
                if (points[i] == cornerX + "," + cornerY) {
                    points[i] = x + "," + y;
                    myMovableObject.setAttributeNS(null, "cx", x);
                    myMovableObject.setAttributeNS(null, "cy", y);
                    i = points.lenght;
                }
            }
            var newPoints = points.join(" ");
            polygon.setAttributeNS(null, "points", newPoints);
        }
    }
}
```