Timo Kataja

Internet-multimediasovelluksen tehosteiden ääniraitasynkronointi

Metropolia Ammattikorkeakoulu
Insinööri (AMK)
Mediatekniikan koulutusohjelma
Insinöörityö
18.4.2011

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Timo Kataja Internet-multimediasovelluksen tehosteiden ääniraitasynkronointi 33 sivua + 4 liitettä 18.4.2011
Tutkinto	insinööri (AMK)
Koulutusohjelma	mediatekniikka
Suuntautumisvaihtoehto	digitaalinen media
Ohjaaja	yliopettaja Erkki Rämö

Insinöörityössä toteutettiin helposti laajennettava komponentti Internet-multimediasovelluksen tehosteiden synkronointiin ääniraidassa tapahtuvien muutoksien perusteella. Luotiin siis ohjelmallinen komponentti, jonka tarkoituksena on käyttäjälle näkyvien visuaalisten elementtien muutosten ajoittaminen äänen mukaan. Tätä voidaan käyttää esimerkiksi näyttävien musiikkisoitinten ja dynaamisten musiikkivideoiden tekoon sekä animaatiohahmojen huulien liikkeiden synkronoimiseksi puheeseen.

Käytetyiksi vaihtoehtoisiksi tekniikoiksi valittiin Adobe Flash ja Microsoft Silverlight, jotka ovat RIA-toteutuksissa (Rich Internet Application) suosituimmat toteutustavat. Työssä toteutettiin metodit ajonaikaiselle, ajoa edeltävälle ja manuaaliselle synkronoinnille sekä lisäksi ääniraidasta riippumaton keinotekoinen visualisointi.

Kaikilla työtavoilla oli hyvät puolensa, riippuen siitä, millaisilla määrityksillä ääneen sitoutettuja efektejä sisältävä sovellus tehdään. Kun halutaan käyttää ajoa edeltävää synkronointia, voidaan käyttää Silverlight-kehitysympäristöä tai Flashin vanhempaa versiota. Ajoa edeltävässä metodissa käytetään ulkoista ohjelmaa tietyn ajanhetken äänenvoimakkuuden selvittämiseen. Manuaalinen synkronointi sopii tehtäviin, joissa vaaditaan tarkkaa äänen ja animaation yhteistoimintaa. Ajonaikainen synkronointi hyvin pitkälle korvaa ääniraidasta riippumattoman visualisoinnin, jota voi silti edelleen käyttää nopeana korvikkeena esimerkiksi loppukäyttäjien selainten lisäosarajoitusten takia.

Nykyiseen tekniikoiden yleisyyteen selaimissa ja työmäärään nähden tehokkaimmaksi metodiksi osoittautui Flashilla toteutettu ajonaikainen ääniraitasynkronointi. Se on vaihtoehdoista parhaiten jatkokehitettävissä tarvittavaan suuntaan ja toimii äänileikkeillä, jotka ladataan vasta ajon aikana. Ladatun äänileikkeen spektriä tulkitaan sitä toistettaessa, ilman että tulkintaan tarvitaan ulkopuolista työkalua.

Avainsanat	ääni, animaatio, Flash, Silverlight

Metropolia

Abstract

Author Title Number of Pages Date	Timo Kataja Audio synchronization of effects in an internet application 33 pages + 4 appendices 18 April 2011
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructor	Erkki Rämö, Principal Lecturer

The purpose of this study was to find out and produce a programmatical component for easily expandable audio synchronization of effects in a web-based multimedia application. In short, making the changes in attributes of visual elements match with changes in audio. This can be used for creating visually impressive audio players, dynamic music videos and for synchronizing the lip movements of animation characters to speech.

The chosen technologies were Adobe Flash and Microsoft Silverlight, which are the most popular tools for creating RIAs (Rich Internet Application). The produced methods included runtime, before-runtime and manual synchronization. Additionally, sound-independent visualization was created.

Each method had its advantages, depending on the specification of the project that audio synchronized effects are created for. Before-runtime synchronization can be used if the chosen technology is Silverlight or a legacy version of Flash. With before-runtime synchronization an external tool is used to analyze audio volume compared to time. Manual synchronization is useful for tasks that require very specific sound and animation synchronization. Run-time synchronization very much replaces the need for sound-independent visualization, but the latter can still be used as a quick method because of for example plugin requirements.

Based on current plugin penetration and workloads, the most efficient method turned out to be runtime synchronization. It is the easiest one for further development for project-specific needs and it works with audio clips that are loaded into the application during runtime. The audio spectrum of the audio clip is analyzed as it plays, without the need for an external application to do the analyzing.

Keywords	Sound, animation, Flash, Silverlight

Sisällys

1 Johdanto

Insinöörityön tarkoituksena on luoda helposti muokattava komponentti tai työtapa visuaalisten tehosteiden, käytetyn tekniikan rajoissa mahdollisten, ohjelmallisten kutsujen tai sovelluksen ulkoisten kutsujen synkronoimiseksi internet-multimediasovelluksen yhteen tai useampaan ääniraitaan. Tehosteiden ääniraitasynkronoinnilla tarkoitetaan jotakin visuaalista muutosta, joka tapahtuu tahdissa äänen, yleensä musiikin, kanssa. Tällainen olisi esimerkiksi musiikin tahdissa näytöllä pomppiva ryhmä palloja tai musiikin äänenvoimakkuuden mukaan hidastuva ja nopeutuva videon toistaminen. Tämäntyyppinen ratkaisu toimii mielenkiinnon ylläpitäjänä hyvin esimerkiksi sivustolla, jonka päätarkoituksena on esitellä artistien musiikkia. Samaa ratkaisua voisi myös soveltaa yökerhoihin, heijastamalla synkronoitua animaatiota videotykillä seinälle. Tämä toimii myös ratkaisuna piirrostyylisten animaatiohahmojen huulien synkronointiin puheeseen, eli ratkaisuna, joka vähentää animaattorien ajankäyttöä rutiinitöihin.

Projektin taustana on työelämässä usein huomattu tarve äänen synkronoinnille Flash 8:lla toteutetuissa verkko-oppimismateriaaleissa. Työn tarkoitus ei kuitenkaan ole keskittyä vain yhteen tekniikkaan, vaan tarkastella tasapuolisesti yleisiä web-teknologioita, jotka voi alustavasti todeta sopiviksi ja riittävän yleisiksi haluttuun tarkoitukseen: Flash 8, joka on vielä nykyäänkin hyvin pitkälti web-multimediasovellusten de facto -standardi vaikkakin ikääntyvä sellainen, Flash 10, joka on tekniikaltaan huomattavasti kehittyneempi muttei käyttäjien selaimissa vielä niin yleinen, sekä Microsoftin Silverlight-tekniikka, jonka avaamien mahdollisuuksien tutkiminen on tekniikan tuoreuden vuoksi vielä suhteellisen alkuvaiheessa (1).

Huomionarvoista on, että Flashin ActionScriptin eri versioiden, Javan ja JavaScriptin kehitykseen on vaikuttanut ECMAscript, joka antaa niille alkeellisen yhteisen pohjan tarkastelua varten (2). Sitä vastoin Silverlight on XAML-pohjainen, ja sen ohjelmointiin käytetään C#:tä tai VisualBasicia. XAML on XML:ään pohjautuva kuvauskieli, jonka opettelu ei ole erityisen haastavaa kenellekään, joka on tutustunut verkkosivuilla käytettävään tavalliseen HTML-kuvauskieleen. HTML5 on myös tulossa ja kovasti ammattipiireissä kohuttu, mutta vielä sen tarkempi tutkiskelu tähän tarkoitukseen ei ole mahdollista (3).

Ääniraitasynkronointia on aikaisemmin toteutettu hajanaisilla metodeilla, joiden käyttö, projektikohtainen valinnan tekeminen ja sujuva toistaminen ja laajentaminen ovat olleet selkeää lähinnä vain niiden alkuperäisille luojille, jotka eivät välttämättä ole saman yrityksen palkkalistoilla pitkiä aikoja kerrallaan. Käytäntöjen dokumentaatio on ollut puutteellista, jopa täysin olematonta, ja suurin työmäärä on usein syntynyt synkronoinnin toteutuksen ymmärtämisestä, ennen kuin sitä voi käytännössä ruveta soveltamaan uuteen projektiin. (4; 5; 6.)

Tarkoitus on siis luoda standardisoitu ohjelmakomponentti, jonka oppimiskynnys on matala, käytön toistamiseen tarvittava aika riittävän pieni ja loppukäyttäjien yhteneväinen käyttökokemus suhteellisen varma. Täydellisen varmasti toimivan ohjelmallisen komponentin luominen on käytännössä mahdotonta, eikä loppukäyttäjien laitteistojen eroavuuksien vuoksi siihen kannata ajankäytön takia pyrkiä yhdessäkään monimutkaisemmassa web-sovelluksessa.

Kuvan ja äänen synkronoinnilla tarkoitetaan kuvassa tapahtuvien muutosten samanaikaistamista ääniraidan tapahtumiin, useimmiten äänenvoimakkuuden muutoksiin. Kuva-ääniraitasynkronointi on hyvin karkeasti yksinkertaistettu termi, koska kuvan paikkaa, väriskaalaa, tummuutta ja lukemattomia muita attribuutteja muuttamalla tietyllä aikajanalla kyseessä on periaatteessa jo alkeellinen animaatio eikä enää pelkkä kuva. Ei kuitenkaan ole selkeyden kannalta erityisen järkevää puhua animaation synkronoinnista ääniraitaan, koska toisinaan animaation ymmärretään jo itsessään sisältävän sekä kuvaa että ääntä.

Ääniraitasynkronointiin käytettävä metodi on perinteisesti valittu projektikohtaisesti. Esimerkiksi alle kymmenen sekunnin mittaisen äänileikkeen äänenvoimakkuuden ohjelmallisen seurannan toteutusta ei voi pitää kovin järkevänä ajan lyhyyden vuoksi. Mutta jos on mahdollista luoda ohjelmallinen komponentti, joka toimii erittäin pienellä vaivannäöllä riippumatta äänileikkeen pituudesta, voidaan ottaa askel totutuista, turhaa toistoa vaativista tavoista eteenpäin kohti standardisoitua elementtiä, jota soveltaen voidaan rakentaa vaadittavat toiminnallisuudet. Poistetaan siis turha työvaihe, jotta saataisiin enemmän aikaa tärkeimmän osan toteutukseen.

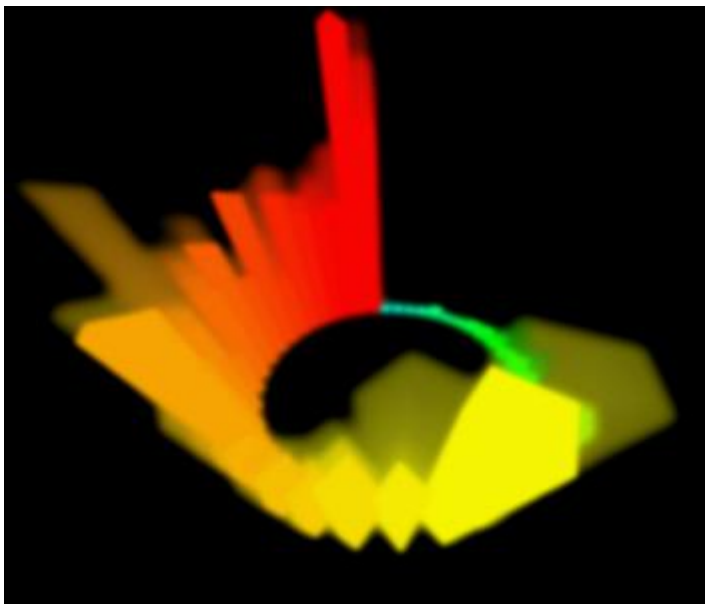
Jos on mahdollista luotettavalla tavalla poistaa manuaalinen työ sillä edellytyksellä, että mahdollisuus on luotu jo aiemmin muitten projektien puitteissa, toimintatapa alkaa kuulostaa järkevältä. Ajan käyttäminen helposti uudelleenkäytettävien elementtien kehittämiseen ei ole yksinomaan uusmedia-alan tehokkuutta lisäävää, vaan se tunnistetaan yleisesti hyödyllisenä lähes ammatista riippumatta.

2 Äänisynkronoinnin käyttö multimediasovelluksen tehosteena

2.1 Synkronoinnin määritelmä

Yksinkertaisimmillaan multimediasovelluksen tapahtumien äänisynkroinnilla tarkoitetaan sitä, että jokin yleensä visuaalinen elementti muuttaa yhtä tai useampaa piirrettä toistetun äänen tahdissa. Esimerkiksi kun tarkasteltavassa sovelluksessa toistettavassa äänitiedostossa äänenvoimakkuus muuttuu suhteellisella tasolla sadasta kahdeksaankymmeneen ja samanaikaisesti vektorigrafiikalla tuotetun suorakulmion korkeus muuttuu 20 prosenttia pienemmäksi aiemmasta, kyseessä on määritelmällisesti efektin äänisynkronointi, vaikkei efekti kovin visuaalisesti vaikuttava vielä olekaan.

Kun lisätään enemmän attribuutteja sovellukseen, saadaan kuitenkin verrattain helposti aikaiseksi näyttävää tulosta. Attribuutit voivat käytännössä olla mitä tahansa visuaalisia ominaisuuksia, joihin ääniraidan eri muutokset vaikuttavat. Muun muassa yhdistämällä äänispektrin analyysi avoimen lähdekoodin lisenssillä levitettävään 3D-tekniikkaan, saadaan aikaiseksi hyvinkin vaikuttavan näköistä animaatiota pienin kuluin (kuva 1).



Kuva 1. Avoimen Papervision 3D -kirjaston toiminnallisuutta hyödyntävä äänispektrin visualisointi (7).

Käytetyn ääniraidan toiston edetessä muutettavan attribuutin ei tosin välttämättä tarvitse olla suoraan visuaalinen määre, vaikka se ilmenisikin visuaalisesti animaation lopputuloksen muutoksena. Samalla periaatteella esimerkiksi ajettavan aikaisemmin tuotetun perinteisen animaation kehysnopeutta eli hetkellistä toiston nopeutta voisi kontrolloida ääniraidan muutoksia seuraamalla ja saada näin jo aiemmin tuotettu animaatiopätkä toistumaan ääniraidan tahdissa vaihtelevasti nopeutuen ja hidastuen.

2.2 Äänisynkronoinnin käytön tarpeellisuus

Ennen kuin mitään äänielementin visualisointityötä aloitetaan, tai oikeastaan mitään työtä yleisestikään, on aiheellista kysyä, kannattaako tai pitääkö jotain tehdä vain siitä syystä, että se voidaan tehdä. Kun tarkastellaan huonoimpia esimerkkejä efektien satunnaisesta lisäämisestä erilaisiin verkkosovelluksiin, huomataan, että jonkin asian toteutuksen mahdollisuus tai helppous ei riitä syyksi toteutukseen.

Ääniraitasynkronointi soveltuu hyvin esimerkiksi sivustoille, joilla esitellään artistin musiikkia. Pelkkä musiikkisoitin voisi olla pidemmän päälle tylsä. Ääniraitasynkronoitu, mahdollisuuksien mukaan automatisoitu visuaalinen lisä tuo enemmän vaikuttavuutta

sivustolle, jonka pääasiallinen tehtävä on tehdä vaikutus loppukäyttäjään. Musiikin tyylisuunnasta riippuen voidaan valita visuaaliselle lisälle sopiva värimaailma ja luoda näin sekä äänen että kuvan välityksellä tunnetila.

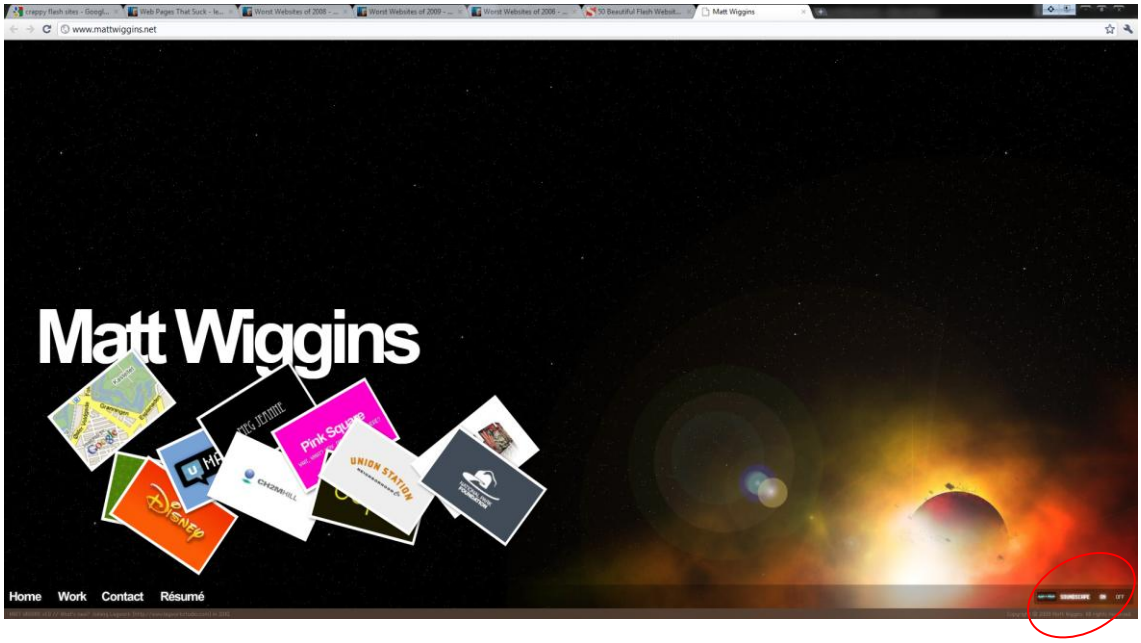
Samoja ääniraitasynkronoinnissa käytettyjä metodeja voi myös suoraan hyödyntää musiikkivideoiden tekoon. Esimerkiksi jos käytetty tekniikka on Adobe Flash, sen voisi suhteellisen pienellä työmäärällä kääntää Adobe Air -sovellukseksi ja näin käytännössä luoda esimerkiksi yökerhokäyttöön teknisesti edullinen työpöytäohjelma, joka luo musiikista visuaalista materiaalia seinälle heijastettavaksi. Musiikki siis olisi edelleen tekijänoikeusmaksujen alainen, mutta automaattisesti tuotettu videomateriaali olisi ohjelmoinnin alkupanostuksen jälkeen ilmaista. Ajonaikaisesti ääniraitasynkronoitu materiaali siis toimii myös hämärtimenä perinteisen musiikkivideon ja multimediasovelluksen välillä.

Kaupallisella musiikkikirjastosivustolla ExtremeMusic.com kaikki musiikkikappaleet näyttävät myös äänenvoimakkuuden tuotetun käyrän ajanhetkellä, jotta ostopäätöstä tekevä kuluttaja pystyisi heti äänen toiston alkaessa päättelemään, miten hyvin se sopii hänen tarpeisiinsa, kuuntelematta koko kappaletta (8).

Ääniraitasynkronointi sopii myös animaation hahmojen huulten liikkeen synkronoimiseen puheeseen. Kovin tarkkuutta vaativaa, fotorealistisen 3D-hahmon synkronointia sillä ei voi toteuttaa, koska kasvojen ilmeet ovat liian monimutkaisia, mutta se on riittäväntasoinen piirrostyylisten animaatiohahmojen huulten liikkeen luomiseen. Luodulle hahmolle voidaan siis lisätä toiminnallisuus huulten automaattiseen liikkumiseen puheäänen tahdissa.

Internetissä on lukematon määrä sivustoja, joilla web-sovellusten teknisiä mahdollisuuksia on käytetty vailla päämäärää, esimerkiksi luomalla huomiota kiinnittäviä vilkkuvia elementtejä, äänitervehdyksiä, jatkuvasti liikkuvia navigointikomponentteja ja varoittamatta ilmestyviä ponnahdusikkunoita. Toisinaan tällaiset sivustot ovat saaneet alkunsa ihan kunnioitettavista tavoitteista, mutta lopputulos aiheuttaa lähinnä päänsärkyä ja sovelluksen käytön nopeaa lopettamista. Samalla tavalla ääniraitasynkronoinninkin käyttö voi olla tarpeetonta ja jopa vahingollista. Sen tulisi harvoin olla hallitsevassa asemassa sivulla, ellei sivusto ole juuri musiikkiin keskittynyt.

Syyllinen huonoon lopputulokseen on usein kunnollisen lopputestauksen puute: Jos testauksen suorittaa toteutuksen tehnyt kehittäjä tai projektiryhmä, oman työnsä virheille on lähes väistämättä sokeutunut. Miltei kaikissa web-sovelluksissa pätee kahden viikon peukalosääntö: Kahden viikon kehitystyön jälkeen ongelmakohtien tunnistaminen alkaa vaikeutua huomattavasti, ja mitä pidemmälle mennään, mahdollisten käytettävyysongelmien ja ärsyttävien pikkuvikojen tunnistaminen alkaa muuttua lähes mahdottomaksi.

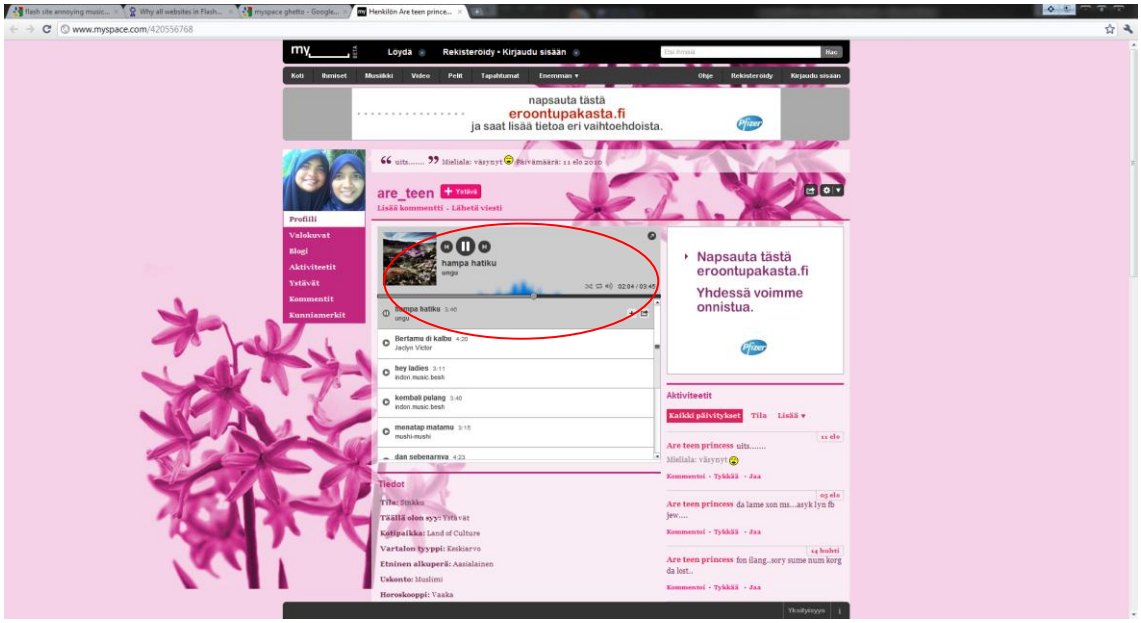


Kuva 2. Selkeä kokonaisuus. Äänisynkronointiin käytetty elementti näkyvillä pienenä oikeassa alanurkassa (9).

Kuva-äänisynkronoinnin käytössä kannattaa olla todella varovainen, jos haluaa pitää tuotoksen modernit käytettävyyden vaatimukset huomioivana. Ääniraitasynkronointi toimii erinomaisesti pienenä visuaalisena lisänä (kuva 2) tai huomionkiinnittimenäkin jos niin halutaan, mutta jos sitä käytetään vain ylimääräisenä huomiota varastavana elementtinä muun sisällön seassa ilman varsinaista tarkoitusta tai ajatusta, se on myös erinomainen tapa saada näkymä näyttämään rakenteeltaan hajanaiselta raakileelta, johon on vain haluttu lisää osia miettimättä kokonaisuutta (10).

Käytettävyysnäkökulmasta on syytä ottaa huomioon, että mitään viestin välityksen kannalta kovin tärkeää informaatiota ei tulisi harvoja poikkeuksia lukuun ottamatta välittää äänisynkrointikomponentin kautta. Yleensä ei ole tarpeellista antaa tälle kom-

ponentille sivun keskeisintä paikkaa, jossa käyttäjä on tottunut odottamaan sivun varsinaisen informaation olevan (kuva 3).



Kuva 3. Runsaasti elementtejä vailla selkeää järjestystä. Äänikomponentti asetettu keskelle sivua. (11).

Riippumatta siitä, sisältääkö elementti bittikarttagrafiikkaa, vektorigrafiikkaa vai videota ja mitä attribuuttia äänen muutoksilla kontrolloidaan, sen käyttö tekstuaalisen sisällön näyttämiseen heikentää lähes väistämättä varsinaisen tiedon välittämistä loppukäyttäjälle (10). Karkeana esimerkkinä voisi ajatella sivun otsikon, joka pomppii äänen tahdissa; harva kuluttaja haluaisi lukea kirjaa, jonka otsikot eivät suostu pysymään paikallaan.

Verkkoympäristössäkin pätevät hyvin pitkälle painomedian nyrkkisäännöt helposta ymmärrettävyydestä ja järkevästä sijoittelusta. Tosin painomedian pitkään kestäneen kehityksen aikana opitut hyvät periaatteet toisinaan unohdetaan Internet-sovelluksissa, koska niiden julkaisu on niin helppoa. Vaikuttavuuden asettaminen tiedonvälittämisen edelle on melkeinpä uusmedia-alan yleisin virhe. Sinänsä se on ymmärrettävää, koska media-alalla kilpailu on kovaa ja vaikuttavalla ulkoasulla pyritään erottumaan kilpailijoista (12).

Samoin on huono ajatus käyttää tekstin taustana äänisynkronoitua animaatiota, ellei kyseessä ole hyvin hailakka tausta. Parhaimmillaan ääniraitasynkronointi on silloin, kun elementtiä käytetään joko yksinomaan ”vau” -efektin luomiseen esittelemällä pelkkä visuaalinen elementti esimerkiksi musiikkikappaleen yhteydessä edes yrittämättä välittää mitään tekstuaalista informaatiota, tai tiedonvälitykseen tarkoitetussa web-sovelluksessa pienenä elementtinä ei-kriittisillä huomioalueilla. Parhaimmillaan elementti on sivustoilla, jotka ovat keskittyneet enemmän musiikkiin kuin tekstisisältöön, sekä animaation osana.

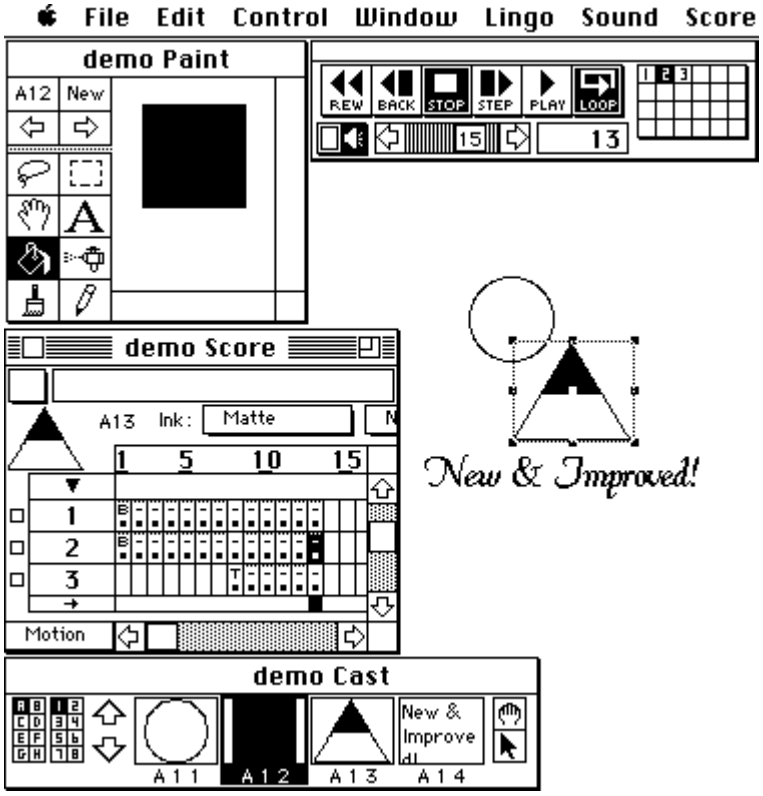
Jos animaatio ei ole oleellisessa osassa, sitä ei tulisi sijoittaa kriittisille alueille, joita web-sovelluksessa ensimmäisenä tarkastellaan (kuva 4). Tällöin se ilmenee käyttäjälle vain mukavana lisänä, ja käyttäjä ei koe sitä ärsykkeenä, joka veisi huomiota pääasiallisesta informaatiosta, jonka välittäminen on verkkosovelluksissa suurimman osan ajasta kuitenkin tarkoituksena.

Kuva 4. Silmän liikkeiden kohdistuminen sivuja luettaessa (13).

3 Äänisynkronoinnin käytön mahdollistavien tekniikoiden kehitys

3.1 Adobe Flash -kehitysympäristö

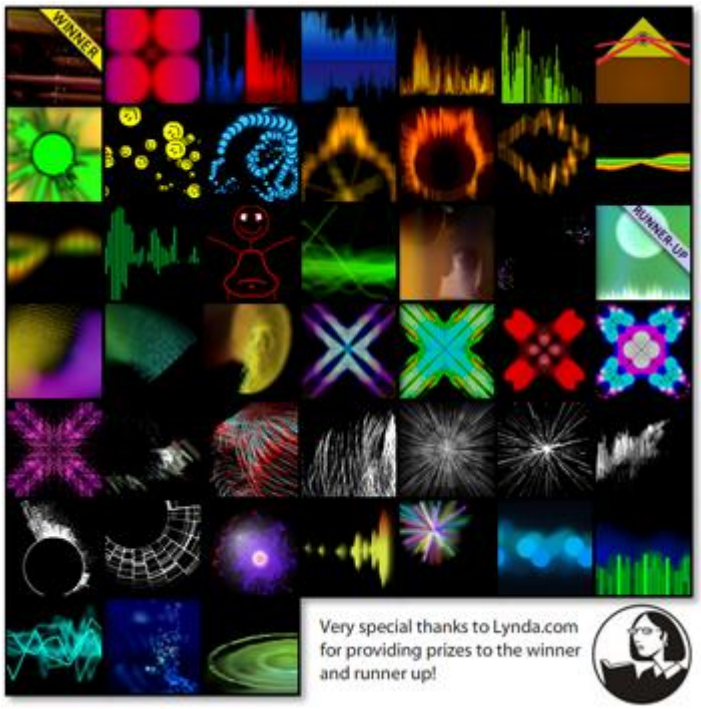
RIA:n (Rich Internet Application) kehityksen historia on kaikki välivaiheet mukaan lukien niin monivaiheinen, ettei tämän työn piirissä voi paneutua siihen kovin syvällisesti. Yleisesti ottaen ääniraitaan synkronoitu animaatio vaatii kuitenkin kahta asiaa: visuaalisen elementin eli vektori- tai bittikarttagrafiikan tuen ja kyvyn toistaa musiikkia jollakin tasolla. Mitä tulee yleisessä käytössä oleviin sovelluksiin, joiden toiminnan yhtenä oleellisena osana on vektoripohjaisen animaation luominen vaivattomasti, ensimmäinen huomattava askel otettiin vasta vuonna 1985. Yhtiö nimeltä MacroMinds julkaisi alkuvaiheessa oleville markkinoille uuden animaatiotyökalun nimeltä VideoWorks, joka avasi oven tämän tyypin ohjelmien jatkokehitykselle. Jo ensimmäinen VideoWorksin kehittäjille esittelemä käyttöliittymä (kuva 5) heijasteli jälkeenpäin tarkasteltuna selvästi tulevaisuutta sellaisille tunnetuille kehitystyökaluille kuin Flash ja Silverlight, jotka kamppailevat nykyään keskenään yleisyydestä käyttäjien selaimissa ja kehittäjien ja loppukäyttäjien suosiosta (1).



Kuva 5. VideoWorksin alkuperäinen käyttöliittymä (14).

Avainkehysanimaation, johon palataan myöhemmissä luvuissa paljon, kehittäjänäkymän visuaalinen esitystapa oli ratkaistu hyvin samalla tapaa kuin nykyisissäkin ohjelmissa ja alkeelliset vektorityökalut oli rakennettu. MacroMind vaihtoi myöhemmin nimensä Macromediaksi ja hankki pienemmältä yhtiöltä, FutureWave Softwarelta, sen kehittämän lippulaivatuotteen FutureSplash Animationin, joka pohjautui käyttöliittymältään pitkälti aiempaan VideoWorksiin. Macromedia vaihtoi tuotteen nimen Flashiksi (14).

Vuonna 1996 julkaistiin Flash 1, ja vuonna 2008 oli päästy versioinnissa Flash 10:een. Välissä oleva kahdentoista vuoden pituinen ajanjakso käsitti teknisen harppauksen suhteellisen yksinkertaisesta animointityökalusta laajaan käyttöympäristöön, joka kykenee nykyään tulkitsemaan suoraan ajon aikana äänitiedoston spektriä (kuva 6).



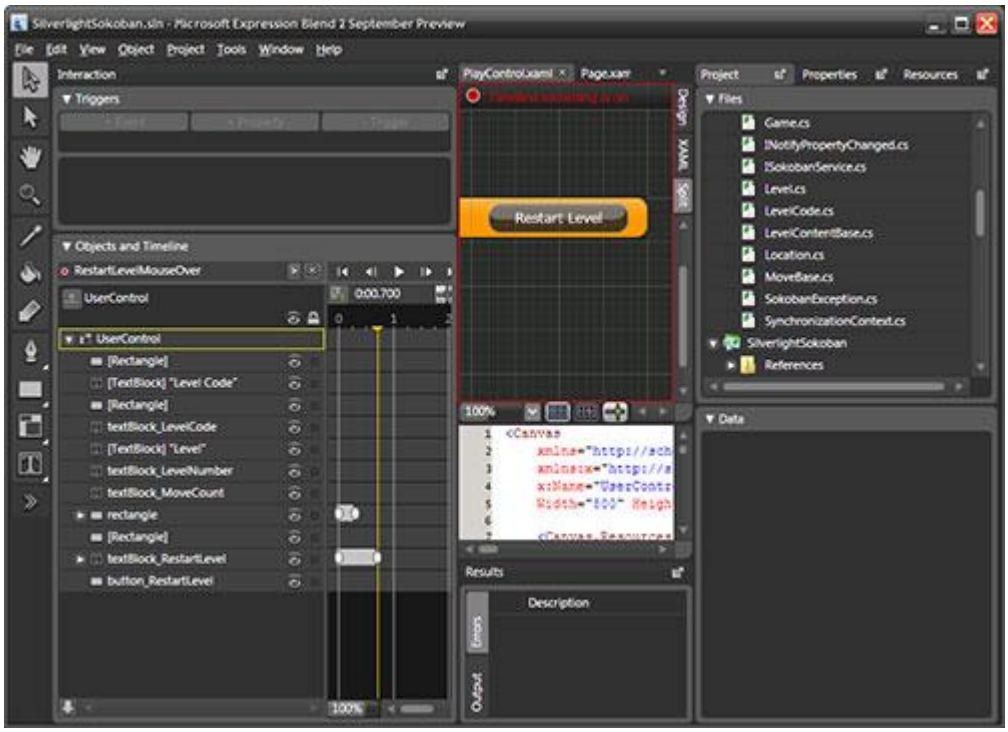
Kuva 6. Flashin tarjoamaa äänispektrin ulostuloa hyödyntäen tehtyjä visualisointeja (15).

3.2 Microsoft Silverlight -kehitysympäristö

Microsoftin kehittämän Silverlightin ensimmäinen versio julkaistiin vuonna 2007, ja siitä ennustettiin RIA-maailmaa käsittelevillä keskustelupalstoilla tulevaisuuden ”flashintap-

pajaa”. Silverlight-ohjelmien visuaalisuus toteutetaan maksuttomalla Microsoft Expression Blend -ohjelmalla, ja ohjelmallinen toiminta tehdään Visual Studiolla, josta myös on saatavissa Silverlight-kehitystyön tarpeisiin riittävä lisenssiltään maksuton erityisversio. Blendin kehittäjät ovat ansiokkaasti ottaneet mallia perinteisten aikajana-animaatioiden tekemiseen käytetyistä ohjelmista (kuva 7).

Blendin perusnäkymä ja animointikäyttö on yhdistelmä sekä Flashia että Autodeskin 3ds Maxin animointitoteutusta, lisänä tietenkin paljon omaa käytettävyyden toteutusta. Project-näkymä noudattaa yleistä ohjelmointityökaluissa käytettyä tiedostonhallinnan tapaa, ja työkalut on intuitiivisesti symboloitu.



Kuva 7. Microsoft Blendin käyttöliittymä.

Taustalla tapahtuva ohjelmallinen toiminta tehdään Visual Basicilla tai C#:llä. Uuden kehittäjän lienee parempi tutustua C#:hen, koska Microsoft ilmoitti vuoden 2008 alussa hylkäävänsä Visual Basicin tuen ja jatkokehityksen.

4 Äänisynkronoinnin käytäntö

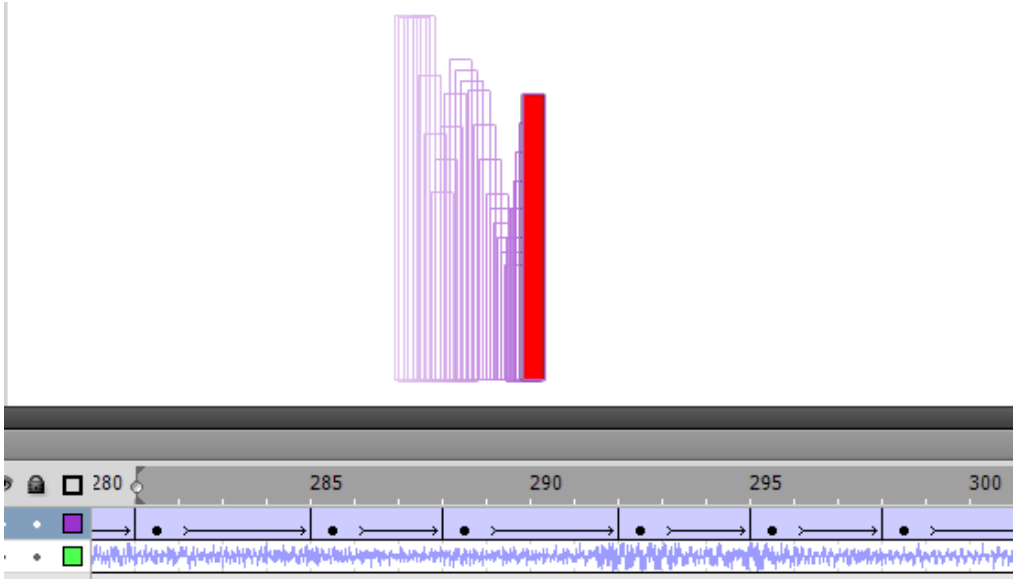
4.1 Manuaalinen synkronointi

Manuaalinen synkronointi tarkoittaa yksinkertaisesti sitä, että manuaalisella työllä animaattori luo visuaalisuuden muutokset näyttämön aikajanalle projektiin valittua ääniraitaa silmällä pitäen, ilman pitkälle automatisoituja ohjelmallisia apuvälineitä. Käytännössä siis animoidaan käsin multimediasovelluksen tapahtumat vastaamaan ääniraidassa tapahtuvia muutoksia, yleensä äänenvoimakkuuden, korkeuden tai tahdin mukaan.

4.1.1 Toteutus Flashilla

Kuvassa 8 Adoben Flash Studiolla on luotu yksi esimerkki, jossa punaisen sata pikseliä korkean suorakulmion vaakasijaintia merkitsevä x-koordinaatti kasvaa jatkuvasti, jotta nähtäisiin animaatiossa selvästi korkeuden vaihtelut, kun Flashin niin kutsuttu sipulinkuorinäkymä on valittu käyttöön esikatselussa. Samaisessa animaattorin käyttöliittymässä ylhäällä olevassa vaaleansinisessä layer-näkymässä näkyvät ylimpänä animaatio- eli tweenkehykset, joissa musta pallo tarkoittaa avainkehystä eli keyframea.

Avainkehykset tarkoittavat animaation muutoskohtia, ja Flash laskee automaattisesti muutokset avainkehysten välille. Alempi layer näyttää sovellukseen sisällytetyn ääniraidan automaattisesti. Punaista ajanvalintapalkkia raahattaessa ääni myös kuuluu oikeassa kohdassa, olettaen että äänen metodiksi on valittu stream eikä event.



Kuva 8. Käsin animoitu palkin pystyvenymä.

Punainen suorakulmio on siis luotu tavanomaisella piirtotyökalulla ja muutettu movieclip-objektiksi, jonka ankkuripisteeksi on määritelty alalaidan keskikohta. Äänitiedoston aikajanalle tuomisen jälkeen palkin korkeus on käsin animoitu saavuttamaan 200 pikselin huippukorkeus.

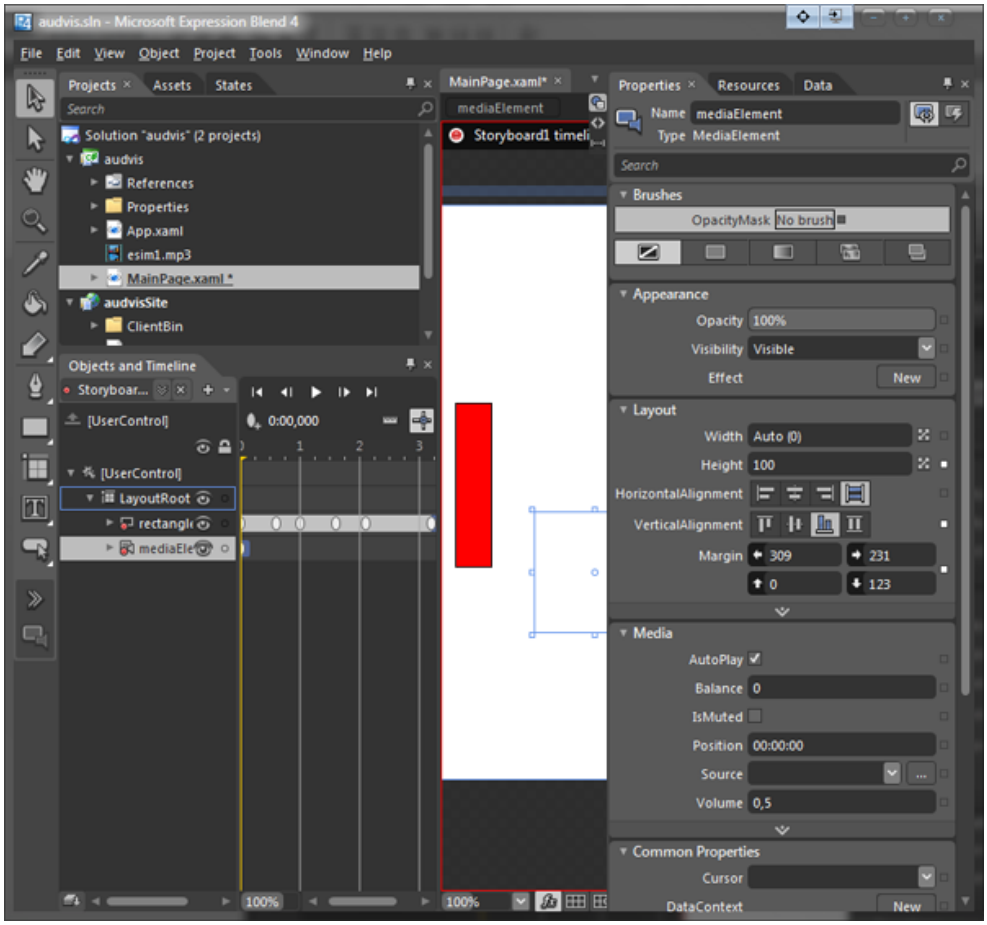
Käsin animoitu tulos voi olla enemmän vaivaa nähden hyvinkin kelvollinen, mutta pitkille äänitiedostoille tämä työtapa ei ole kovin edullinen: kuvassa 8 oleva pätkä on vain sekunnin mittainen, ja käsin animoiden jo yhden elementin attribuuttien muutto pidemmän äänitiedoston muutosten mukana on todella isotöistä, puhumattakaan siitä, että muutettaisiin useampaa elementtiä kerralla.

4.1.2 Toteutus Silverlightilla

Kun käytetään Silverlightia manuaaliseen synkronointiin, huomataan ensimmäiseksi yksi iso kompastuskivi sujuvaa työskentelyä silmälläpitäen: vaikka Silverlight-projektia tehtäessä käytettävä Microsoft Blend -kehitystyökalu tavallaan asettaa äänen samalla tavalla storyboardille, joka on periaatteessa samanlainen kuin Flashin timeline, se ei näytä visuaalisesti äänen muutoksia.

Silverlightin näyttämölle eli storyboardille tuodaan musiikki MediaElement-symbolin sisälle, joka ei millään tapaa viestitä kehittäjälle sitä, mitä juuri sillä hetkellä tuodussa

ääniraidassa tapahtuu (kuva 9): MediaElement on esikatseluvaiheessa hyvin pitkälti kuin tyhjä laatikko, joka ei kerro kehittäjälle ulospäin mitään eikä reagoi storyboardin ajamiseen millään tavalla.



Kuva 9. SilverLightin mediaElement storyboardilla.

Mielenkiintoista ja työtavan kannalta hyvin nurinkurista on, että käytännössä näkymättömästä äänielementistä ja sen ominaisuuksista kyllä esitetään oikealla palkissa varsin tarkasti turhaa tietoa, kuten näkyväisyys, joka ei tunnu vaikuttavan mihinkään äänitiedostoa käytettäessä, ja korkeus ja leveys, joilla ei sinänsä ole mitään tekemistä yksinomaan äänen toistoon käytetyn elementin kanssa.

Ilmeisesti MediaElementin toiminnallisuutta luotaessa kehittäjätiimi on ajatellut sitä etupäässä ideon helppoon ja nopeaan julkaisemiseen tarkoitettuna objektina eikä vaativampien äänen muutosten seuraamiseen pohjautuvien animaatioiden toteutustyökaluna. Esimerkkinä käytetyn punaisen suorakaiteen korkeuteen tehdyt muutokset näky-

vät selkeästi storyboardilla, ja valkoiset soikeat kuviot taas symboloivat avainkehyksiä, joiden välille ohjelma laskee itse arvoja odottamatta jokaiselle kehykselle käsinmääriteltyä attribuuttia. Mikäli samantyyppinen esitystapa olisi käytössä myös MediaElementissä äänen käyrän kanssa, kyseessä olisi tavallinen ja järkevä työtapa.

Vasta julkaistaessa nähdään, onko arvioimalla tehty manuaalinen visualisointi osunut siihen kohtaan kuin pitäisi. Selaimessa toistuu sekä palkin animaatio että musiikkitiedosto. Tämä tosin tarkoittaa sitä, että jotta kehittäjä pystyisi tekemään tämäntyyppistä synkronointia, hänen täytyy jatkuvasti julkaista sovellusta uudestaan selaimessa tarkasteltavaksi.

Yksi käytännönläheinen vaihtoehto olisi myös käyttää ulkoista musiikkisoitinta tai äänieditoria ja seurata suoraan siitä ääniraidan muutoksia. Sekään ei tosin ole kovin kätevää, ja ylimääräisen ohjelman tarve tämäntyyppisessä työssä laskee Silverlightin pisteitä. Silverlight on tosin vastoin markkinointiaan osoittautunut ennemminkin tietokantaan liitettävien sovellusten tekemiseen hyväksi, eikä niinkään Flashin visuaalisten ominaisuuksien kanssa kilpailijaksi, kuten Microsoft aiemmin mainosti julkaisuvaiheessa.

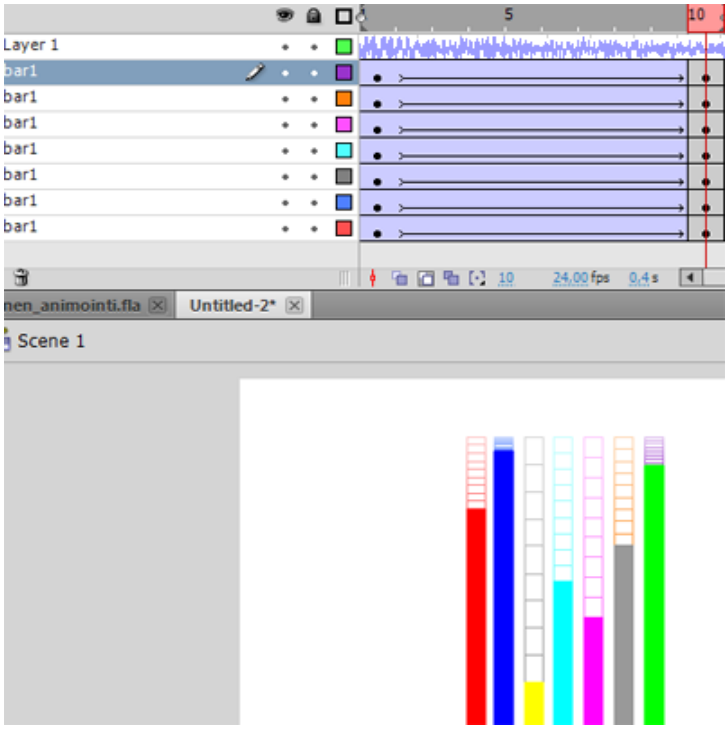
4.2 Äänielementistä riippumaton manuaalinen visualisointi

Toisinaan Internet-sivuilla on saatettu käyttää äänisynkronoinnin kaltaisen efektin hakuun equalizer-elementtiä, jossa siis liikkuu rivi animoituja palkkeja ainakin näennäisesti taustamusiikin mukaisesti. Tarkkaavainen käyttäjä kuitenkin huomaa nopeasti, etteivät elementin palkit liiku tahdissa äänen kanssa. Tätä toteutustapaa käytettiin sivustoilla usein musiikkisoittimen yhteydessä antamaan sille elävämpi ilme. Esimerkiksi MySpace käytti sitä käyttäjäsivujensa soittimessa, kunnes se Flash-lisäosan tuoreempien versioiden yleistyessä pystyttiin korvaamaan luvussa 4.4 esitettävällä ajonaikaisella synkronoinnilla.

Kyseessä ei siis ole tarkalleen ottaen ääniraitasynkronointi, vaikka samansuuntaisesta halutusta lopputuloksesta on kyse. Metodi on kuitenkin halpa, koska palkkeihin ja toimintaan ei välttämättä tarvitse tehdä muutoksia, vaikka käytettyä äänitiedostoa vaihdettaisiinkin useasti.

4.2.1 Toteutus Flashilla

Kuvassa 10 nähdään Flash Studiolla manuaalisesti toteutettu näennäinen ääniraidan visualisointi. Studion käyttöliittymästä on laitettu ”onion skinning” päälle tapahtuvien muutosten helpompaa tarkastelua varten työstönäkymässä.



Kuva 10. Äänen näennäinen visualisointi manuaalisesti.

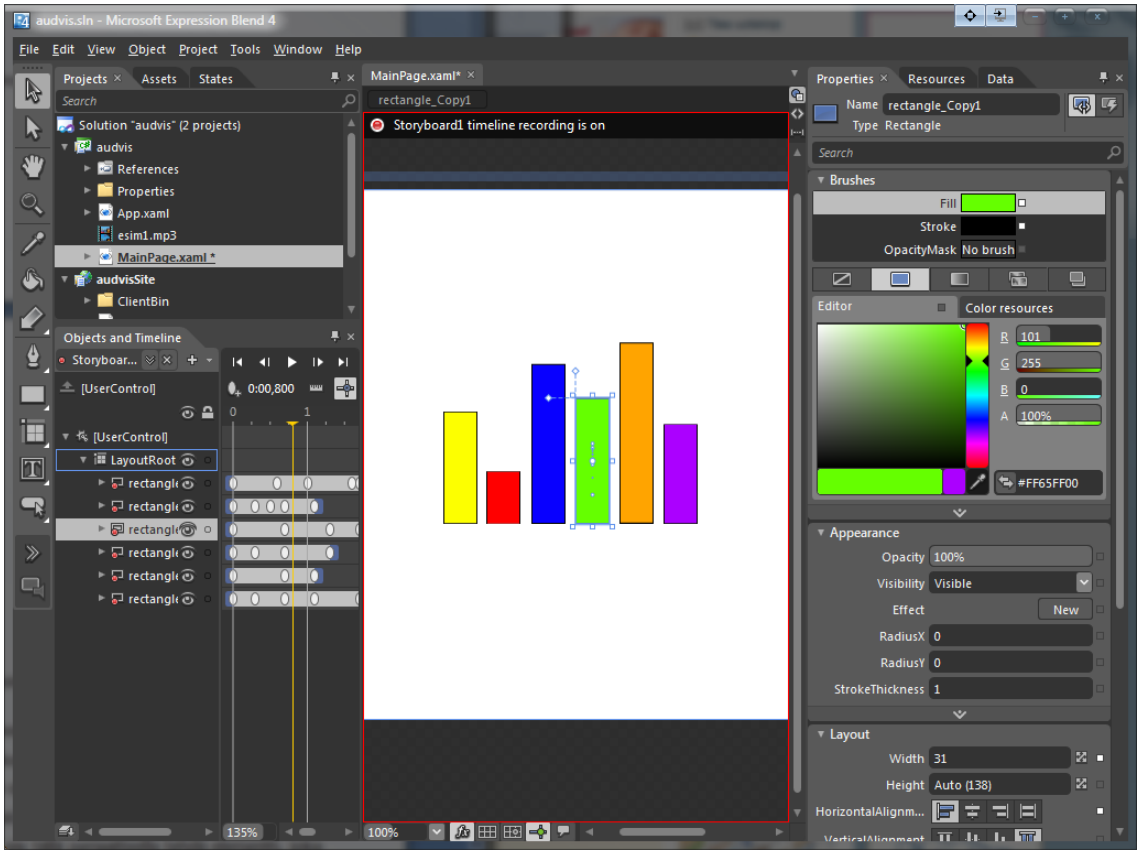
Palkit liikkuvat täysin sattumanvaraisesti, eikä sillä ole väliä mikä sisään ladattu ja toistettu äänitiedosto on kyseessä. Tämäntyyppinen ratkaisu oli aiemmin käytössä muun muassa MySpacen käyttäjäsivuilla, ennen kuin Flash 9:n levinneisyys käyttäjien selaimissa nousi tarpeeksi korkealle ja mahdollisti suoran dynaamisen synkronoinnin käytön uudistetulla versiolla ActionScriptista.

4.2.2 Toteutus Silverlightilla

Kun toteutetaan samaa näennäistä visualisointityyppiä Silverlightilla, työnkulussa ei ole juurikaan eroa. Alussa luodaan elementit palkeille ja värjätään ne. Sen jälkeen kytketään storyboardin automaattinen nauhoitus päälle ja muutetaan palkkien korkeutta

suhteessa aikaan. Automaattinen nauhoitus on todella pätevä ja mukava käyttää, ja se tekee tällä toimintatavalla animoinnista nopeampaa kuin vastaava toteutus Flashilla. Riittää, kun se on päällä ja selaa aikaa eteenpäin, muuttaa jotakin hivenen, selaa eteenpäin, muuttaa ja niin edelleen.

Automaattinen nauhoitus tarkoittaa sitä, että kuvassa 11 näyttämön vasemmalla puolella nähtävälle storyboardille ilmestyvät avainkehykset automaattisesti aina, kun palkkeihin tehdään mikä tahansa muutos.

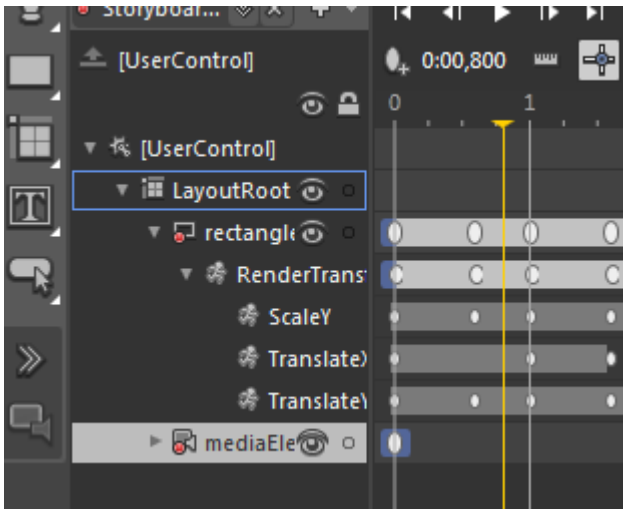


Kuva 11. Näennäinen visualisointi Silverlightilla

Ei ole siis laisinkaan tarpeen luoda manuaalisesti avainkehystä aina, kun kehittäjä toteaa, että tässä kohdassa olisi hyvä olla jokin visuaalinen muutos. Tekemällä muutoksia storyboardille tapahtuu sekä muutos että sitä edeltävä avainkehyksen automaattinen lisäys.

Erinomainen lisäys tähän animaatiotyöskentelyyn on myös tapa, jolla Blend esittää storyboardiin tehdyt muutokset tarkemmin niin haluttaessa: avattavat alavalikot näyttävät

suoraan, mitä muutoksia elementtiin on tehty sillä ajanhetkellä verrattuna alkuperäiseen (kuva 12).



Kuva 12. Blendin luettelo muutetuista attribuuteista.

Esimerkiksi pystyskaalan säätäminen näkyy omina avainkehyksinään kohdassa ScaleY, ja sieltä pystytään poistamaan avainkehyksiä erikseen vaikuttamatta muiden attribuuttien avainkehyksiin. Kyseessä on näppärä ominaisuus, jonka soisi yleistyvän muissakin animaatiota käsittelevissä työkaluissa.

4.3 Ajoa edeltävä automatisoitu synkronointi

Kun synkronointi toteutetaan, ennen kuin loppukäyttäjä varsinaisesti on vielä alkanut käyttää sovellusta, se tarkoittaa periaatteessa sitä, että visuaaliseen synkronointiin tarvittava tieto pitää olla hankittu valmiiksi ja tallennettu staattisessa muodossa etukäteen. Toisin sanoen äänen hetkellinen voimakkuus analysoidaan ja tallennetaan käyttökelpoiseen muotoon etukäteen siihen soveltuvilla työkaluilla.

Yksi edellä mainitun kaltainen työkalu oli nimeltään FlashAmp, jonka jatkokehitys on nykyään jo loppunut, siitä ymmärrettävästä syystä, että Flash 8.0 oli käytännössä viimeinen versio, jossa työkalusta oli enää mahdollista hyötyä markkinoilla. Sen jälkeen tarvittava äänen analysointi on voitu toteuttaa suoraan ajonaikaisella koodilla, jota käsitellään luvussa 4.4. Toiminnaltaan samantyyppisiä lisäosia on kirjoitettu ainakin Ado-

be After Effectsille. FlashAmpin jatkokehitys on lopetettu, ja ohjelman alkuperäinen valmistaja tarjoaa sitä ilmaiseksi halukkaiden käyttöön kotisivuillaan (16).

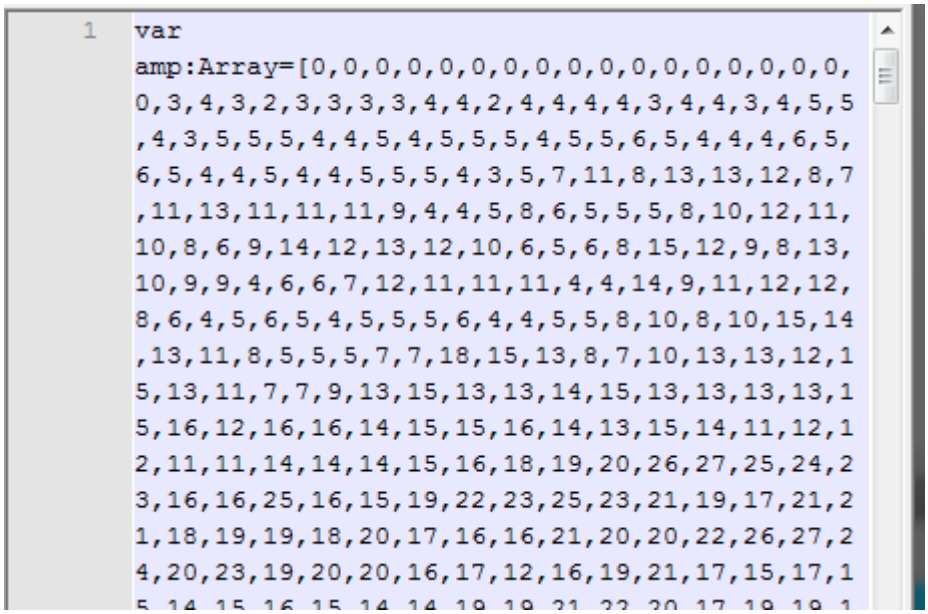
4.3.1 Toteutus Flashilla

Käytännössä luvussa 4.3 esitellyt työkalut palauttavat äänen voimakkuuden ajanhetkellä numerosarjana, joka voidaan joko säilyttää suoraan multimediasovellukseen muuttujan sisällä tai laittaa ulkoiseksi tiedostoksi, joka ladataan sovellusta ladattaessa (liite 1).

Työtavan kannalta ei ole juurikaan väliä, kumpaa tapaa käyttää. On erittäin todennäköistä, että FlashAmpin tai muun vastaavan äänenvoimakkuusdatan generointiin tarkoitetun ohjelman käyttäjä on myös tuotettavan sovelluksen pääasiallinen ohjelmoija. On toki mahdollista tehdä soitin, joka lataa ulkopuolelta sekä äänitiedoston että äänenvoimakkuusdatan esimerkiksi XML-tiedoston sisältä. Mutta todennäköisesti jos koostaja on hankkinut itselleen FlashAmpin, hän omistaa myös sovelluksen julkaisuun vaadittavan työkalun, eikä datan ohjelmoiminen sovelluksen sisäiseksi osaksi ole näin ongelma. Saadun voimakkuusdatan säilyttäminen ulkoisessa XML-tiedostossa ei tietenkään ole itseisarvo.

Hieman lähempänä todellisuutta olisi vaihtoehto, jossa käytetään esimerkiksi Adobe After Effectsin skriptausmahdollisuutta äänenvoimakkuusdatan saamiseen, mutta ainakin yrityskäytössä on hyvin usein tapana ostaa isompi kokoelma Adoben tuotteita kuin pelkkä After Effects, jolloin julkaisutyökalu olisi todennäköisesti joka tapauksessa ääniraitadatan alkuperäisen luojan käytössä.

Kuvassa 13 on ote FlashAmpilla äänitiedostosta generoidusta sarjasta äänenvoimakkuusdataa ajanhetkellä. Sarja on sisällytetty suoraan multimediasovellukseen taulukkomuodossa eli arrayna.



Kuva 13. Äänitiedoston äänenvoimakkuus ajanhetkellä.

Taulukon alussa on ensimmäiseksi tasainen kaistale nollaa, koska kappaleessa, josta taulukko on generoitu, on ollut alussa hiljaisuutta hetken aikaa, kuten musiikkitallenteissa on tapana. Sen jälkeen alkaa odotetusti äänenvoimakkuuksien vaihtelu, jota seuraamalla saadaan toteutettua muutokset visuaalisiin elementteihin.

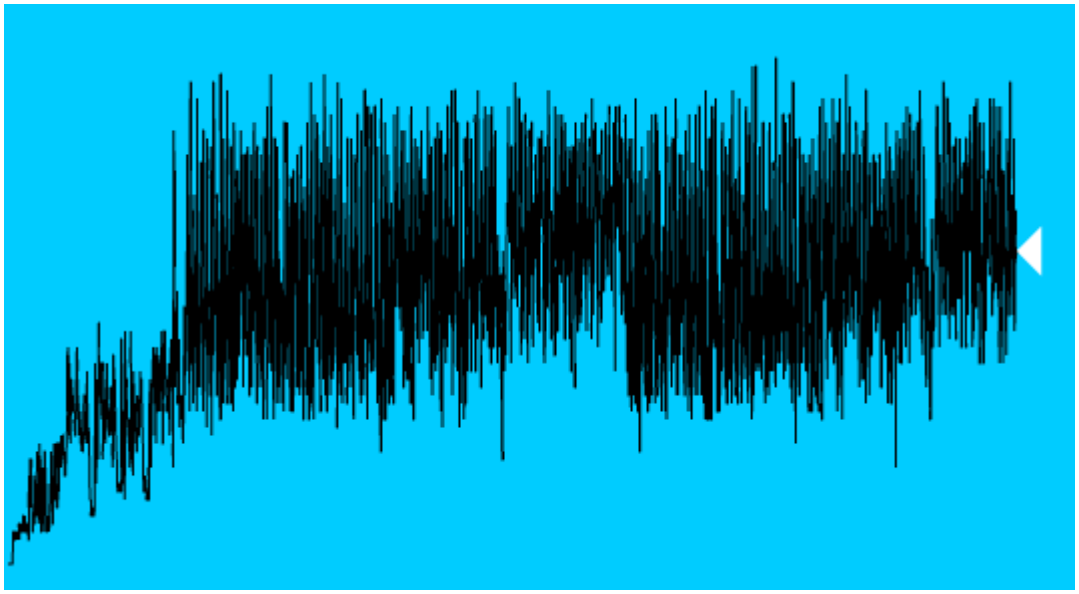
Toteutuksessa liitetään näyttämölle kirjastosta musiikkikappale (koodiesimerkki 1), jonka äänenvoimakkuusdata nähtiin kuvassa 13.

```
3       var s:Sound = new Sound();
4       s.attachSound("audio1");
5
6   toista.onPress = function() {
7       _root.viiva.removeMovieClip();
8       s.start();
9       _root.createEmptyMovieClip("viiva",1)
10      viiva.lineStyle(1,0x000000);
11      viiva.moveTo(20,300);
12  }
13  stoppaa.onPress = function(){
14      s.stop();
15
16      }
17
18  this.onEnterFrame = function(){
19      var perc:Number = s.position/s.duration;
20      var index:Number = Math.floor(amp.length*perc);
21      ball._x = 400*2*perc+20;
22      ball._y = 300-amp[index]*4;
23      viiva.lineTo(ball._x, ball._y);
24  }
```

Koodiesimerkki 1. Generoidun äänenvoimakkuusdatan piirtäminen näyttämölle.

Funktiossa onEnterFrame käydään läpi jatkuvasti, missä kohtaa taulukkoa mennään, ja siirrytään x-akselilla sen mukaan eteenpäin ja haetaan viivan piirtämiselle oikeaa y-koordinaattia jatkuvasti äänenvoimakkuuden sisältävästä taulukosta. Näin jatkuvasti laskemalla saadaan joka hetki uusi piste, johon viivan piirto siirtyy lineTo-komennolla onEnterFrame-funktion viimeisellä rivillä.

Näin saadaan aikaiseksi saumattomasti piirtyvä käyrä, joka kasvaa tarkalleen samaan tahtiin äänitiedoston toiston edetessä (kuva 14).



Kuva 14. Ajan myötä piirtyvä hetkellisen äänenvoimakkuuden kuvaaja.

4.3.2 Toteutus Silverlightilla

Flashilla toteutettu ohjelmallinen prosessi on myös täysin monistettavissa Silverlight-tekniikkaan. Storyboardilla oleva materiaali luodaan XAML:lla, ja C#-ohjelmointikieleen kopioidaan aiemmin toteutettu logiikka. Aiemmin luotu prosessi on toteutettavissa melko lailla millä tahansa ohjelmointiympäristöllä, joka sallii äänen toistamisen ja noudattaa tiettyä vaihtumatonta kehysnopeutta eli frameratea.

FlashAmpilla tuotettu avainkehysdata on käyttökelpoista materiaalia ohjelmointiympäristöstä melko lailla riippumatta: tarvitsee vain ohjelmallisesti monistaa samanlainen toiminta, ja lopputulos on useimpiin tarpeisiin riittävä.

Silverlightin edusta on tehty XAML:lla, jolla ei tähän tapaukseen tarvinnut luoda kuin Rectangle-elementti kuvaamaan äänen muutoksia ja MediaElement, johon ladataan sama aiemmin käytetty äänitiedosto (koodiesimerkki 2). MediaElement on suorakulmainen alue, johon voidaan ladata videota tai ääntä. Ääntä käytettäessä siinä ei näy mitään visuaalista ulospäin muuta kuin kehitystyökaluja käyttäessä, ei loppukäyttäjille (17).

```
100%
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5       xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6       mc:Ignorable="d"
7       d:DesignHeight="300" d:DesignWidth="400">
8
9       <Grid x:Name="LayoutRoot" Background="White">
10          <Rectangle x:Name="Rect3" Fill="#FFF4F4F5" HorizontalAlignment="Left" Margin="150,66,0,132" Stroke="Black" Width="32"/>
11          <MediaElement x:Name="player" HorizontalAlignment="Right" Margin="0,66,88,134" Width="100" Source="/esim1.mp3"/>
12
13      </Grid>
14  </UserControl>
15
```

Koodiesimerkki 2. XAML:lla luotu näyttämö.

Oletusarvoisesti sisään ladattua mediaa aletaan toistaa saman tien, odottamatta erillistä play-kutsua. Tästä syystä onkin järkevämpää asettaa kuuntelijat latauksen valmistumiselle, kuin ohjelmoida erillistä toistamistoiminnallisuutta. XAML:ssa olevia elementtejä kontrolloidaan C#:lla kirjoitetulla luokalla.

Data-taulukko luodaan, ja palkin lähtökorkeudeksi asetetaan 10. Mediaelementille annetaan tarvittava Handler, joka seuraa, mitä mediaelementissä tapahtuu. Tällä kertaa erityisesti seurataan sitä, kun tiedosto on latautunut ja sitä aletaan soittaa funktiossa _mediaElement_MediaOpened (koodiesimerkki 3).

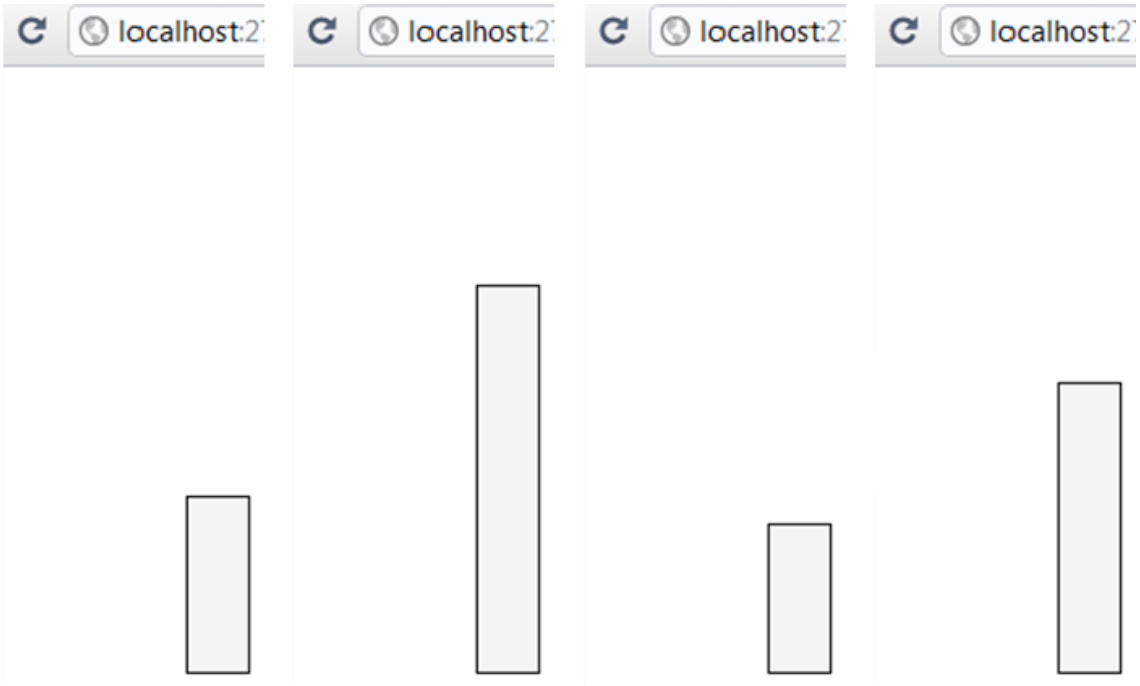
SilverlightApplication1.MainPage

```
int[] amp = new int[] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 4, 3, 2, 3, 3, 3,
int count = 0;
double deci = 0;

public MainPage()
{
    InitializeComponent();
    Rect1.Height = 10;

    MediaElement player = new MediaElement();
    player.MediaOpened +=new RoutedEventHandler(_mediaElement_MediaOpened);
    LayoutRoot.Children.Add(player);

    System.Windows.Threading.DispatcherTimer dt = new System.Windows.Threading.DispatcherTimer();
    dt.Interval = new TimeSpan(0, 0, 0, 0, 30); // 500 means 500 Milliseconds
    dt.Tick += new EventHandler(dt_Tick);
    dt.Start();

}

void dt_Tick(object sender, EventArgs e)
{
    count++;
    Rect3.Height = amp[count] * 10;
}

void _mediaElement_MediaOpened(object sender, RoutedEventArgs e)
{
    count = 0;
    deci = 0;
    player.Play();


}
```

Koodiesimerkki 3. C#:lla toteutettu ohjelmointi.

Funktion sisällä asetetaan muuttujat nolliksi, jotta taulukkoa tulkittaisiin oikeasta kohtaa. Mediaelementin asettamisen jälkeen luodaan uusi Interval-toiminta nimeltä dt, joka 30 millisekunnin välein päivittää palkin korkeuden taulukosta saatavan datan mukaiseksi äänitiedoston toistamisen edetessä.

Lopputulokseksi saadaan ohjelmalliselta toiminnaltaan melko lailla samanlainen toteutus kuin Flashilla aiemmin (liite 3). Näyttämöllä on identifierilla korvamerkitty suorakulmio, jonka korkeus vaihtelee ääniraidassa tapahtuvien muutosten mukaan automatisoidusti (kuva 15). Kuten aiemmin Flashissa, tätäkin voi jalostaa pidemmälle esimerkiksi konvertoimalla seurattavaksi dataksi sekä vasemman että oikean kanavan erikseen tai eri taajuusalueita useampaan eri palkkiin.



Kuva 15. XAML:lla ja C#:lla toteutettu äänen muutoksia seuraava palkki eri hetkillä.

4.4 Ajonaikainen automatisoitu synkronointi

Ajonaikaisessa synkronoinnissa äänen muutokset tunnistetaan multimediasovellusta ajettaessa, ja kuvan muutokset tehdään näiden äänen muutosten mukaisesti. Flash 8:lla tämä ei ollut vielä mahdollista äänityökalujen vajavaisuuden vuoksi. Kun Adobe julkisti uuden ActionScript 3.0 -kielen, siihen oli lisätty työkalut suoraa äänispektrin käsittelyä varten, mikä teki käytännössä mahdolliseksi ajonaikaisen automatisoidun synkronoinnin.

Yksinkertaisesti tiivistäen voidaan siis todeta, että tällä periaatteella voidaan rakentaa ohjelmallinen elementti, joka kykenee automaattisesti seuramaan minkä tahansa elementtiin ladatun MP3-muotoisen äänitiedoston spektrin muutoksia. Vaikka MP3-tiedosto valittaisiin sovelluksessa toistettavaksi vasta sovelluksen julkaisun jälkeen, ajonaikainen synkronointi analysoi sen oikein. Näin pystyttäisiin siis luomaan esimerkiksi musiikkisoitin, joka visualisoi näyttämölle mitä tahansa ääntä, jota sillä valitaan soitettavaksi, ei vain ennalta analysoituja äänileikkeitä, kuten oli ajoa edeltävän synkronoinnin tapauksessa.

4.4.1 Toteutus Flashilla

Toteutettaessa äänisynkronointia Flashilla pyrittiin seuraamaan stereokanavaisen MP3-tiedoston kummankin puolen muutoksia visuaalisesti. Vaikka Flash Studio tukeekin useamman eri ääniformaatin importointia aikajanalle, dynaamisesti lataamisessa voi käyttää vain MP3-formaattia. Se ei todennäköisesti muodostu usein ongelmaksi, koska MP3 on äänitiedostona todella yleinen ja ilmaisia käännösohjelmia muiden formaattien muuttamiseen MP3-muotoon löytyy paljon. (18; 19.)

Ensimmäiseksi sisään ladattiin ActionScript 3.0:n URLRequest-metodilla esim1.mp3:ksi nimetty äänitiedosto projektikansion omasta juuresta. Ladatulle MP3-tiedostolle lisättiin SoundTransform-luokka, jota käytetään yleisesti ActionScript 3.0:lla toteutetuissa Flash-projekteissa ladatun äänen ajonaikaiseen muuttamiseen johonkin haluttuun lopputulokseen. SoundTransform-luokka sisältää ominaisuuksia ja funktioita, joilla asetetaan ja luetaan attribuutteja äänenvoimakkuudelle ja panoroinnille (20). Tässä tapauksessa käytettiin pelkästään äänenvoimakkuuden ja vasenoikea-kanavan säätelyä, jotta pystyttiin todentamaan ääniraitojen visualisoituvan oikein ja erillään. Asetettiin siis toiston äänenvoimakkuudeksi 1, eli ohjelmallisesti sama kuin 100 %, ja vasenoikea-kanavaa säädeltiin koetarkoituksessa vasemmasta laidasta –1:stä oikeaan laitaan +1:een verraten tuloksia, ja lopuksi jätettiin nollaan, jolloin kanavat ovat alkuperäisessä tasapainossa.

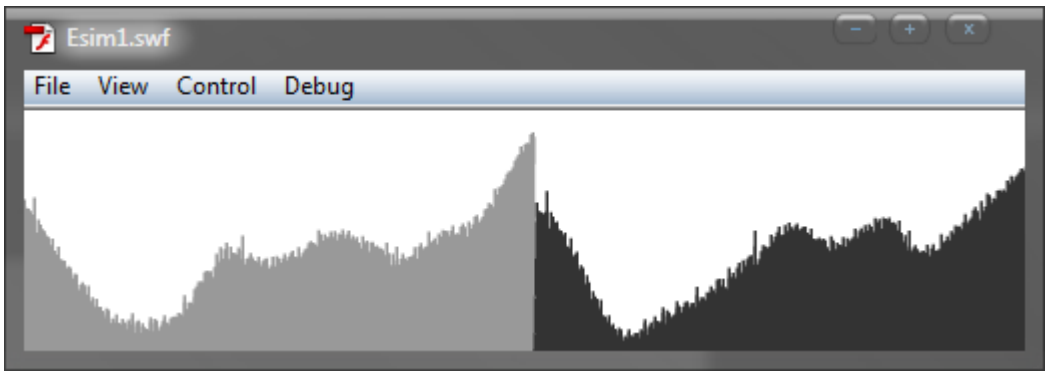
Näyttämölle lisättiin EventListener, jonka tehtävä oli toistuvasti kuunnella muutosta kehyksen vaihtuessa ja ajaa automaattisesti paivitys-nimistä funktiota, joka tulisi huolehtimaan ääntä kuvaavan graafin piirrosta (koodiesimerkki 4). Graafin piirtoon käytettiin Flashin Sprite-luokkaa. Sekä vasemmalla että oikealla kanavalla oli kummallakin 256 saraketta, joissa data muuttuu ajanhetkellä, joten piirtäjän piti käydä läpi 256 yksittäistä saraketta kahteen kertaan, eri kohdista.

```
38              public function paivitys(e:Event):void {
39
40
41                  var aanispektrumi:ByteArray = new ByteArray();
42                  SoundMixer.computeSpectrum(aanispektrumi);
43                  graaf.graphics.clear();
44
45
46                  graaf.graphics.moveTo(0,120);
47                  //Vasen kanava
48                  for(var i:int = 0; i <256; i++){
49                      spektrumi = 20 + aanispektrumi.readFloat() * 20;
50                      graaf.graphics.lineStyle(1,0x999999);
51                      graaf.graphics.lineTo(i-1, 120);
52                      graaf.graphics.lineTo(i, Math.abs(spektrumi)*3);
53
54                      addChild(graaf);
55                  }
56                  graaf.graphics.moveTo(254,120);
57
58                  for(var j:int = 0; j <256; j++){
59                      spektrumi = 20 + aanispektrumi.readFloat() * 20;
60                      graaf.graphics.lineStyle(1,0x333333);
61                      graaf.graphics.lineTo(j-1+256, 120);
62                      graaf.graphics.lineTo(j+256, Math.abs(spektrumi)*3);
63
64                      addChild(graaf);
65                  }
66
67
68              }
```

Koodiesimerkki 4. Kuviota jatkuvasti päivittävä funktio.

Yhden ajanhetken läpikäynnin jälkeen piirtäjä palasi esimerkin alimpaan käytettyyn y-koordinaattiin 120, jotta saatiin tasainen piirretty alue pelkän viivan sijaan. Flash Studiossa koordinaatit lähtevät vasemmasta ylänurkasta, eli y-koordinaatti 120 tarkoittaa pistettä, joka on 120 pikselin päässä näyttämön ylälaidasta.

Lopputulokseksi saatiin visualisointi, jossa näytetään vaaleanharmaalla värillä vasemman kanavan spektri ja tummempana oikean kanavan (kuva 16). Visualisointi muuttuu jatkuvasti äänen toiston jatkuessa, rasittamatta suoritinta kovasti. Kyseessä on siis suhteellisen helppo, prosessoritaloudellinen ja uudelleenkäytettävä metodi (liite 2).



Kuva 16. Ajonaikaisen synkronoinnin lopputulos, vasen ja oikea kanava.

4.4.2 Toteutus Silverlightilla

Valitettavasti ainakaan toistaiseksi Microsoft Silverlight ei tarjoa mahdollisuutta saman toiminnallisuuden kopiointiin. Silverlight kyllä tietää, missä kohtaa toistettavaa äänitiedostoa liikutaan, mutta se ei saa ulos mitään muuta hyödyllistä tietoa, pelkästään senhetkisen ajan (17). Näin ollen työtapa, joka pohjautuu kokonaan äänispektrin ajonaikaiseen tulkintaan, ei ole ainakaan nykyisellä Silverlight-ympäristöllä toteutettavissa tekniikan puutteiden vuoksi. Ei kuitenkaan ole mitenkään mahdoton ajatus, että se tultaisiin siihen lisäämään tulevaisuudessa, koska Silverlight on nopeasti päivittyvä suhteellisen tuore teknologia.

5 Tulosten tarkastelu

Työssä toteutetuista äänisynkronoinnin tavoista käytännöllisimmäksi osoittautui luvussa 4.4 esitelty äänen ajonaikainen automatisoitu synkronointi. Se on erinomaisen nopea toteuttaa, onnistuu alusta loppuun ilman Flash Studion ulkopuolisia työkaluja ja toimii millä tahansa MP3-muotoisella äänitiedostolla ilman konvertoinnin tai äänispektrin datan ulkoisen analysoinnin tarvetta. Se vaatii loppukäyttäjältä vähintään Flash 9 -lisäosan, mutta sen levinneisyys selaimissa on tasolla 99 % länsimaissa, joten voitaneen olettaa sen olevan suhteellisen luotettavasti saatavilla, mikäli käyttäjäkohderyhmä sisältää lähinnä eurooppalaisia tai amerikkalaisia (1). Valitettavasti samanlaisen elementin luomiseen ei ollut mahdollisuutta Silverlight-tekniikalla ainakaan kirjoitushetkellä.

Kun ajonaikaiseen automaattiseen synkronointiin yhdistetään esimerkiksi 3D-kirjastojen toiminnallisuutta, saadaan lopputulokseksi helposti uudestaan käytettävää, visuaalisesti vaikuttavan näköistä materiaalia. Sillä voi toteuttaa esimerkiksi myös piirroshahmojen huulten synkronointia puhetta sisältävään äänitiedostoon tai luoda musiikkisoittimeen visuaalisen osuuden. Ylipäänsäkin kaikissa tilanteissa, joissa äänen toistaminen on sovelluksessa suuressa roolissa, tämäntyyppisellä synkronoinnilla voidaan luoda visuaalisia osia, jotka lisäävät sovelluksen vaikuttavuutta. Musiikkia käyttävissä animaatioissa ainakin osittain pystytään hyödyntämään ajonaikaista synkronointia, vaikka todella tarkkuutta vaativat kohdat kannattaa toteuttaa manuaalisella synkronoinnilla. Esimerkiksi musiikin tahtiin leijailevat lehdet, huojahtelevat puut ja muut ei-inhimilliset elementit toimivat hyvin automatisoidun synkronoinnin kanssa.

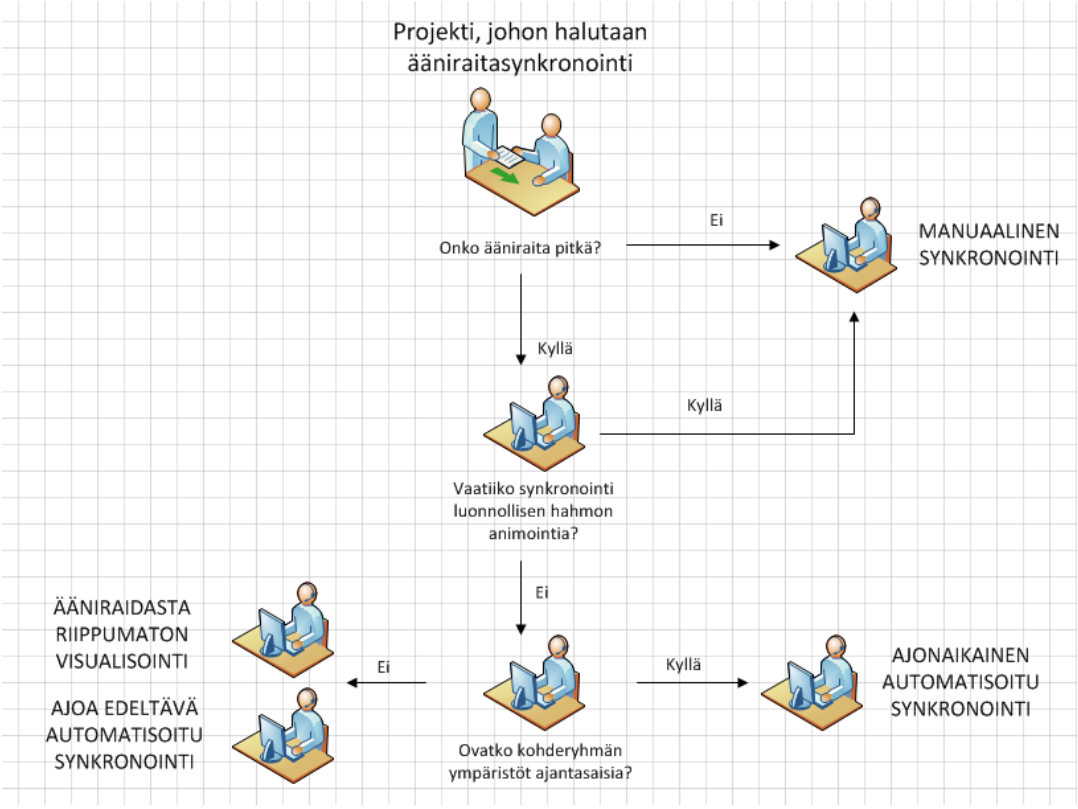
Manuaalinen synkronointi on suositeltavaa, mikäli tehdään todella tarkkuutta vaativaa animointia, esimerkiksi pyritään saamaan monimutkaiset ihmisenkaltaiset luurankoon sitoutetut animaatiohahmot liikehtimään musiikin tahdissa mahdollisimman luonnollisen näköisesti.

Ihmisenkaltaisen realistisen hahmon animointiin ohjelmallinen synkronisointi ei millään kykene inhimillisten aistinvaraisten tulkintojen puutteen vuoksi, vaan animointityö pitää suorittaa perinteisesti käsin ja verrata animaation liikkeitä suoraan käytettävään ääniraitaan. Mitään muuta varsinaista suositusta käytetylle työkalulle ei ole, kuin se mihin tekijä on tottunut. Subjektiivisia näkemyksiä animaatiotyökalujen paremmuudesta löytyy toki.

Tekijän oman mukavuuden tärkeys koskee samalla tapaa myös äänielementistä riippumatonta manuaalista visualisointia: kun synkronointi ei ole automatisoitu, tekijän tottumus nousee isoon asemaan työkalun valinnassa. Lisäksi tulee ottaa huomioon oletetut loppukäyttäjän selaimeen asennetut lisäosat. Äänielementistä riippumaton visualisointi ei ole varsinaisesti synkronointia, mutta sen toteuttamisen helppous tosiaan riippuu totutusta työkalusta. Jos toteutettava projekti on määritelty alhaisen version lisäosalle pienellä budjetilla, hyvin pian huomataan, että heikoilla resursseilla ei saa vahvaa jälkeä. Tällöin riippumaton visualisointi on vaihtoehto.

Ajoa edeltävä automatisoitu synkronointi toimii molemmilla tutkituilla tekniikoilla, sekä Flashilla että Silverlightilla, aika lailla samalla tapaa. Sen sijaan on kyseenalaista, onko tällä metodilla enää mitään arvoa, ellei muista syistä ole valittu Silverlightia jo aiemmin toteutustekniikaksi. Flashilla osoittautui huomattavasti yksinkertaisemmaksi toteuttaa ajonaikainen synkronointi.

Yksinkertaistaen vaikuttaa siltä, että useimmissa tapauksissa Flashilla toteutettu ajonaikainen ääniraitasynkronointi, joka työn kuluessa toteutettiin valmiiksi käyttökuntoisuuteen asti, on monistamiskelpoinen ja riittävän helposti laajennettava vastaamaan useimpiin tarpeisiin (kuva 17).



Kuva 17. Ääniraitasynkronoinnin metodin valinta.

Erityistapauksissa voi työn luonteesta riippuen harkita muita metodeja. Esimerkiksi jos projekti on jo muutenkin luotu alusta alkaen Silverlightilla, ei ole järkevää vaihtaa ekosysteemiä vain ääniraidan kylkeen lisättävien efektien vuoksi. Edelleen jos on syytä olettaa, että Flashiin pohjautuvassa projektissa kohdeyleisön laitteissa ei ole käytössä

vielä AS3:a ymmärtävää Flash-lisäosaa, ajonaikaisen synkronoinnin teko on mahdotonta ilman palvelinpään ohjelmointia.

6 Yhteenveto

Insinöörityössä pyrittiin selvittämään yksinkertainen keino internet-multimediasovelluksen efektien muutosten synkronointiin ääniraidan kanssa. Tämä tehtiin kahta eri RIA-tekniikkaa käyttäen, Adobe Flashia ja Microsoft Silverlightia.

Valitut tekniikat olivat ajankohtaiset ja sopivat, tosin HTML5:n alkaessa vallata markkinaosuutta on syytä tutustua sen tarjoamiin mahdollisuuksiin samalla alueella. Flash osoittautui muuntautumiskykyisimmäksi, ja sillä toteutettu ajonaikainen synkronointi vastaa todennäköisesti suurimmilta osin niiden projektien tarpeita, joissa tämäntyyppistä ääneen sitouttamista tarvitaan. Silverlightilla samaa metodia ei saanut nykyisellä lisäosan versiolla toteutettua, mutta aika näyttää, mikä on äänirajapinnan tulevaisuus jatkuvasti kehittyvässä tekniikassa. Ajoa edeltävä synkronointi toimii yhtä lailla sekä Flashilla että Silverlightilla, ja käytettävää metodia valittaessa tulee huomioida lisäosien yleisyys kohdeyleisön selaimissa ja ennen muuta luotavan animaation luonne. Eri metodit eivät yleisluontoisesti vastaa aivan samoihin tarpeisiin, vaan yhdessä animaatiossa voi olla järkevää hyödyntää esimerkiksi sekä ajonaikaista että manuaalista synkronointia.

Ajonaikaiselle synkronoinnille toteutettu metodi on riittävän muuntautumiskykyinen ja helppotajuinen toimiakseen projekteissa, joissa visuaalinen näkymä on äänestä riippuvainen. Tällaisia projekteja olisivat esimerkiksi verkossa musiikin esittelyyn tarkoitettujen sivustojen soittimet tai mahdollisesti reaaliaikaisesti tuotetut musiikkivideot klubikäyttöön.

Lähteet

1 Rich Internet Application Statistics. Verkkodokumentti. DreamingWell.com. <http://riastats.com/>. Automatisoidusti päivittyvä. Luettu 19.1.2011.

2 ECMAScript dialects. Programmer's Wiki. 2009. Verkkodokumentti. Wikia. <http://code.wikia.com/wiki/Category:ECMAScript_dialects>. Luettu 15.1.2011.

3 Top 10 Reasons why HTML 5 is not ready to replace Silverlight. 2010. Verkkodokumentti. Silverlight Hack. <http://silverlighthack.com/post/2010/02/08/Top-Reasons-why-HTML-5-is-not-ready-to-replace-Silverlight.aspx>. Luettu 12.1.2011.

4 Sound Visualization in ActionScript 2.0. Stackoverflow. 2009. Verkkokeskustelu. <http://stackoverflow.com/questions/1632200/sound-visualization-in-actionscript-2-0> Luettu 25.3.2011.

5 Adobe Flash Music Visualization Help. 2008. Verkkokeskustelu. Armor Games. <http://armorgames.com/community/thread/1933113/adobe-flash-music-visualization-help/page/1> Luettu 25.3.2011.

6 Swift MP3. Verkkodokumentti. Swift-Tools. <http://www.swift-tools.net/swift-mp3.php> Luettu 25.3.2011.

7 Papervision 3D Sound Spectrum. Verkkodokumentti. Papervision3D. <http://pv3d.org/2008/12/06/papervision3d-sound-spectrum/>. Luettu 2.2.2011.

8 ExtremeMusic.com – The Bad Boys of Production Music. Verkkodokumentti. ExtremeMusic. <http://www.extrememusic.com/#album/1914> Luettu 25.3.2011.

9 Wiggins, Matt. Home. Verkkodokumentti. <http://www.mattwiggins.net/>. Luettu 20.1.2011.

10 McMullin, Jess ja Skinner, Grant. 2011. Usability Heuristics for Rich Internet Applications. Verkkodokumentti. <http://www.boxesandarrows.com/view/usability_heuristics_for_rich_internet_applications>. Luettu 17.2.2011.

11 Myspace-käyttäjäsivu. Verkkodokumentti. <http://www.myspace.com/420556768>. Luettu 20.1.2011.

12 Follett, Jonathan. 2007. Interfaces That Flow: Transitions as Design Elements. Verkkodokumentti. <http://www.uxmatters.com/mt/archives/2007/04/interfaces-that-flow-transitions-as-design-elements.php>. Luettu 17.2.2011.

13 Nielsen, Jakob. F-shaped Pattern For Reading Web Content. 2006. Verkkodokumentti. <http://www.useit.com/alertbox/reading_pattern.html>. Luettu 3.2.2011.

14 Bitter, Raju. 2009. The Timeline from Videoworks in 1985 to Silverlight and JavaFX. Verkkodokumentti. <http://openfuture.rajubitter.com/2009/07/07/the-timeline-from-videoworks-in-1985-to-silverlight-and-javafx/>. Luettu 5.2.2011.

15 Brimelow, Lee. 2006. The Flash Blog. Verkkodokumentti. <http://blog.theflashblog.com/?p=197>. Luettu 6.2.2011.

16 FlashAmp. Verkkodokumentti. Marmalade Media. <http://www.marmalademedia.com.au/download/index.html> Luettu 28.3.2011.

17 MediaElement. Verkkodokumentti. MSDN Library. <http://msdn.microsoft.com/en-us/library/bb980132(v=vs.95).aspx> Luettu 22.2.2011.

18 Basics of working with sound. Flash CS4 Resources. Verkkodokumentti. Adobe. <http://help.adobe.com/en_US/ActionScript/3.0_ProgrammingAS3/WS5b3ccc516d4fbf351e63e3d118a9b90204-7d27.html> Luettu 29.3.2011.

19 Hakutulos ”MP3 Converter”, freeware-lisenssillä. Verkkodokumentti. Download.com. < http://download.cnet.com/1770-20_4-0.html?query=mp3+converter&searchtype=downloads&rpp=10&filter=licenseName=Free|platform=Windows&filterName=licenseName=Free|platform=Windows&tag=pe-searchFacetsTile;navForm> Luettu 29.3.2011.

20 SoundTransform, Adobe Language Reference. Verkkodokumentti. Adobe. < http://www.adobe.com/livedocs/flash/9.0/ActionScriptLangRefV3/flash/media/SoundTransform.html>. Luettu 22.2.2011.

FlashAmpilla konvertoitu äänitiedoston avainkehysdata kohdissa 0–50

[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,3,4,3,2,3,3,3,3,4,4,2,4,4,4,4,3,4,4,3,4,5,5,4,3,5,5,5,
4,4,5,4,5,5,5,4,5,5,6,5,4,4,4,6,5,6,5,4,4,5,4,4,5,5,5,4,3,5,7,11,8,13,13,12,8,7,11,13,11,
11,11,9,4,4,5,8,6,5,5,5,8,10,12,11,10,8,6,9,14,12,13,12,10,6,5,6,8,15,12,9,8,13,10,9,9
,4,6,6,7,12,11,11,11,4,4,14,9,11,12,12,8,6,4,5,6,5,4,5,5,5,6,4,4,5,5,8,10,8,10,15,14,13
,11,8,5,5,5,7,7,18,15,13,8,7,10,13,13,12,15,13,11,7,7,9,13,15,13,13,14,15,13,13,13,13
,15,16,12,16,16,14,15,15,16,14]

Ääniraitadatan lukeminen ja piirtäminen ajonaikaisesti Flashilla

Esim1.as

```
package  {
        import flash.events.Event;
        import flash.utils.ByteArray;
        import flash.net.URLRequest;
        import flash.display.Sprite;
        import flash.display.StageAlign;
        import flash.media.Sound;
        import flash.media.SoundChannel;
        import flash.media.SoundMixer;
        import flash.media.SoundTransform;

        public class Esim1 extends Sprite {
                private var aani1:Sound;
                private var kanava:SoundChannel;
                private var spektrumi:Number;
                private var graaf:Sprite = new Sprite();

                public function Esim1 () {

                stage.align = StageAlign.LEFT;
                var trans:SoundTransform
                trans = new SoundTransform(2, 0);

                aani1 = new Sound(new URLRequest("esim1.mp3") );

                kanava = aani1.play(0,1,trans);


                stage.addEventListener(Event.ENTER_FRAME, paivitys);
```

```
                }

                public function paivitys(e:Event):void {


                var aanispektrumi:ByteArray = new ByteArray();
                SoundMixer.computeSpectrum(aanispektrumi);
                graaf.graphics.clear();

                graaf.graphics.moveTo(0,120);
                //Vasen kanava
                for(var i:int = 0; i <256; i++){
                        spektrumi = 20 + aanispektrumi.readFloat() *
20;
                        graaf.graphics.lineStyle(1,0x999999);
                        graaf.graphics.lineTo(i-1, 120);
                        graaf.graphics.lineTo(i,
Math.abs(spektrumi)*3);

                        addChild(graaf);
                }
                graaf.graphics.moveTo(254,120);

                for(var j:int = 0; j <256; j++){
                        spektrumi = 20 + aanispektrumi.readFloat() *
20;
                        graaf.graphics.lineStyle(1,0x333333);
                        graaf.graphics.lineTo(j-1+256, 120);
                        graaf.graphics.lineTo(j+256,
Math.abs(spektrumi)*3);

                        addChild(graaf);
                }}}}
```

Ääniraitadatan lukeminen ja piirtäminen ajoa edeltävällä datalla Flashilla

```
var amp:Array=[0,0,0,0,0] // Lyhennetty generoitu data, katso Liite 1.

var s:Sound = new Sound();
s.attachSound("audio1");

_root.viiva.removeMovieClip();
s.start();
_root.createEmptyMovieClip("viiva",1)
viiva.lineStyle(1,0x000000);
viiva.moveTo(20,300);

this.onEnterFrame = function(){
                var perc:Number = s.position/s.duration;
                var index:Number = Math.floor(amp.length*perc);
                ball._x = 400*2*perc+20;
                ball._y = 300-amp[index]*4;
                viiva.lineTo(ball._x, ball._y);
}

stop();

ball._x=20;
ball._y=300
```

Ääniraitadatan lukeminen ja piirtäminen ajoa edeltävällä datalla Silverlightilla

MainPage.xaml

```
<UserControl x:Class="SilverlightApplication1.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400">

    <Grid x:Name="LayoutRoot" Background="White">
            <Rectangle x:Name="Rect3" Fill="#FFF4F4F5" HorizontalAlign-
ment="Left" Margin="150,66,0,132" Stroke="Black" Width="32"/>
            <MediaElement x:Name="player" HorizontalAlignment="Right" Mar-
gin="0,66,88,134" Width="100" Source="/esim1.mp3"/>

    </Grid>
</UserControl>
```

MainPage.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
```

```
namespace SilverlightApplication1
{
    public partial class MainPage : UserControl
    {
    int[] amp = new int[] { 0, 0, 0, 0, 0, 0, 0,  }; // Lyhennetty generoitu data, katso
Liite 1
    int count = 0;
    double deci = 0;
   public MainPage(){
   InitializeComponent();
   MediaElement player = new MediaElement();
   player.MediaOpened +=new RoutedEventHandler(_mediaElement_MediaOpened);
   LayoutRoot.Children.Add(player);
System.Windows.Threading.DispatcherTimer dt = new            Sys-
tem.Windows.Threading.DispatcherTimer();
        dt.Interval = new TimeSpan(0, 0, 0, 0, 30); // 500 means 500 Milliseconds
        dt.Tick += new EventHandler(dt_Tick);
        dt.Start();

        }
        void dt_Tick(object sender, EventArgs e)
       {
        count++;
        Rect3.Height = amp[count] * 10;
       }
      void _mediaElement_MediaOpened(object sender, RoutedEventArgs e)
       {
       count = 0;
       deci = 0;
       player.Play();
} }
}
```