Viskijukka — ohjelmisto musiikin reaaliaikaiseen visualisointiin

Lopputyö
Medialaboratorio
Taideteollinen korkeakoulu

Markku Reunanen

28. maaliskuuta 2005

Tiivistelmä

MEDIALABORATORIO
TAIDETEOLLINEN KORKEAKOULU

Reunanen, Markku: Viskijukka — ohjelmisto musiikin reaaliaikaiseen visualisointiin
Lopputyö, 38 sivua, 6 liitesivua

Sähköpostiosoite: marq@iki.fi
Vuosi: 2005
Avainsanat: visualisointi, video jockey, reaaliaika, avoimet ohjelmat, Linux

Viskijukka on ohjelmisto, jota käytetään musiikin reaaliaikaiseen visualisointiin livetilanteissa, kuten konserteissa. Visuaalien ympärille on rakentunut oma alakulttuurinsa, jonka edustajia ovat Video Jockeyt eli VJ:t. Tyypillisesti käytetty kuvamateriaali koostuu videoleikkeistä, joiden toistamisen ja efektien tuloksena saadaan aikaan kiinnostava esitys. Viskijukka lähestyy visuaalien ongelmaa toisesta suunnasta: esitettävä grafiikka tuotetaan reaaliajassa, mikä avaa uusia mahdollisuuksia 3D-grafiikan ja nopean musiikkiin reagoimisen kautta. Lähtökohdan keskeisiä haasteita ovat esityksen säilyttäminen kiinnostavana sekä joustavan ohjelmiston rakentaminen.

Tässä lopputyössä kuvataan Viskijukka-ohjelmistolle asetetut tavoitteet, ohjelmistotekninen toteutus ja lopuksi arvioidaan projektin onnistumista. Vertailu muihin ohjelmiin auttaa lukijaa hahmottamaan VJ-kentän työkaluja ja toimintatapoja. VJ-kulttuurin lisäksi ohjelmisto kytkeytyy vahvasti avointen ohjelmien ideologiaan, jonka tradition mukaisesti koko tuotettu lähdekoodi julkaistaan vapaaseen levitykseen.

Abstract

MEDIA LAB
HELSINKI UNIVERSITY OF ART AND DESIGN

Reunanen, Markku: Viskijukka—Software for Realtime Music Visualization
Master's thesis, 38 pages, 6 enclosure pages

E-mail: marq@iki.fi
Year: 2005
Keywords: visualization, video jockey, real-time, open source, Linux

Viskijukka — Whiskey Jockey — is a software platform that is used for the real-time visualization of music in live situations such as concerts. The concert visuals have given birth to a complete subculture of Video Jockeys (VJ's). Typically the material used consists of video clips that are played with additional effects to produce an interesting performance. Viskijukka approaches the problem from a different angle: the graphics are produced in real-time which allows for new possibilities such as 3D graphics and fast reacting to the music. The largest challenges of the approach are keeping the performance interesting and the programming of the needed flexible software.

This final thesis describes the goals set for the platform, the software architecture and finally the evaluation of the project. Some benchmarking against the existing solutions is provided to help the reader understand the tools and methods used by Video Jockeys. Besides the VJ culture the platform is strongly connected to the open source ideology. Following the open source tradition the full source code produced will be released to the community.

Esipuhe

Valitsin lopputyöni aiheen oman mielenkiintoni perusteella ja toteutin sen vapaa-ajallani ilman ulkopuolista rahoitusta. Työn edistymistä tukivat useat henkilöt, joille esitän tässä kiitokseni:

Kiitän lopputyöni ohjaajia, lehtori Antti Ikosta ja prof. Tapio Takalaa, kommenteista ja tuesta työn tekemisessä. Valtiot. yo. Antti Silvastille esitän kiitokset VJ-kulttuuria koskevista kommenteista ja käytännön näkökulmista. Kiitän työtovereitani DI Tommi Ilmosta ja DI Perttu Hämäläistä hyödyllisistä keskusteluista järjestelmän suunnitteluvaiheessa. Tradenomi Jarkko Rotsténille ja fil. yo. Yrjö Fagerille esitän kiitokseni testisession antoisasta musiikista. Tyttöystävääni Annaa kiitän oikoluvusta ja henkisestä tuesta.

Niin produktio kuin kirjallinen osakin on tehty avoimilla työkaluilla, jotka täyttivät paikkansa erinomaisesti. Tärkeimmät käyttämäni ohjelmat olivat Linux, LyX, LaTeX, FTE, GCC, GNU make, Xfig ja GIMP. Kiitos näiden ohjelmien tekijöille pyyteettömästä työstä yhteisön hyväksi.

Sisältö

Luku 1

Johdanto

Lopputyöni aiheena on Viskijukka, ohjelmisto musiikin reaaliaikaiseen visualisointiin. Ohjelmistoa voidaan käyttää live-konserteissa tuottamaan musiikkiin ja käyttäjän syötteeseen synkronoituja näyttäviä visuaalisia efektejä. Konserttien visualisoinnista vastaava henkilö tunnetaan nimellä VJ — video jockey. Projektin nimi syntyi analogisesti DJ-sanasta: tätä musiikin soittamisesta ja miksaamisesta huolehtivaa henkilöähän on kutsuttu Suomessa kotoisasti tiskijukaksi. Tavoitteena oli tuottaa yhtenäinen ja laajennettava ohjelmointikirjasto, joka toimii pohjana uusille reaaliaikaisille visuaaleille.

Akateemisessa mielessä asetin työlle kolme keskeistä tutkimusongelmaa. Ensimmäinen koski avoimia ohjelmia: *miten avoimia ohjelmia käyttämällä voidaan tuottaa visuaaleja?* Perinteisesti videota, ääntä ja kuvaa on käsitelty kaupallisilla ohjelmilla, jotka ovat usein kalliita ja siten kotikäyttäjän ulottumattomissa. Myös 3D-mallinnusohjelmat ja ohjelmointityökalut ovat suurelta osin kaupallisia. 90-luvun puolivälistä alkaen ns. avoimia ohjelmia kannattava liike on saanut merkittävää jalansijaa ohjelmistotuotannossa ja tuottanut työkaluja myös multimedian käsittelyyn. Tutkimuskysymyksen vastausta lähdin hakemaan toteuttamalla projektin avoimilla työkaluilla, samalla arvioiden niiden käyttökelpoisuutta.

Toinen tutkimusongelma oli edellistä teknisempi: *millainen ohjelmiston rakenne tukee samalla sekä joustavuutta että reaaliaikaisuutta?* Samaa ongelmaa tutkin jo diplomityössäni (Reunanen, 2002) saamatta kuitenkaan tyydyttävää vastausta. Rinnakkaisuutta sisältävän monimutkaisen ohjelmistoalustan suunnittelu ja toteutus ovat haastavia tehtäviä kokeneellekin ohjelmoijalle.

Viimeinen tutkimusongelma oli luonteeltaan viihteellinen: *millaiset visuaalit ovat kiinnostavia katsojalle?* Syvällinen ymmärrys asiaan voi muodostua vasta pitkän ajan kuluessa, mutta jo tähän työhön kartoitin visuaalien viihdyttävyyden olemusta. Eri musiikkityylit vaativat visuaaleilta erilaista ulkoasua sekä rytmiä eivätkä visuaalit saa toistaa itseään liiaksi pitkän konsertin aikana. Miten tätä uudistuvuutta ja jännitettä pidetään yllä?

Viskijukan luonnissa toimin ensin ohjelmistosuunnittelijan ja sitten ohjelmoijan roolissa. Ohjelmistoalustan suunnittelu, vaatimusten määrittely ja toteutus ovat omaa käsialaani. Toteutusryhmän toinen keskeinen jäsen, Antti Silvast, osallistui työhön tarjoamalla käyttööni pitkän kokemuksensa VJ-toiminnasta — alueen oma tuntemukseni on lähinnä teoreettista. Antin kautta pääsin epäsuorasti hyödyntämään myös Amfibio- ja Pseudotoad Laboratories -VJ-kollektiivien kokemuksia. Jarkko Rotstén luovutti projektin käyttöön säveltämäänsä musiikkia.

Kirjallisen esityksen fokus on ohjelmistoalustan vaatimusten ja toiminnan kuvauksessa. Valinta on harkittu ja edustaa omaa osaamistani: historialliset katsaukset ja esteettiset pohdiskelut olisivat vääjäämättä jääneet raapaisun asteelle. Työn merkityksen ja sijoittumisen kannalta keskeiset kontekstit määrittelen luvussa 2. Luvussa 3 esittelen hankkeelle asetetut vaatimukset, jotka osaltaan vaikuttivat valitsemiini toteutusratkaisuihin ja lopputulokseen. Järjestelmän toiminnallinen kuvaus ohjelmistoteknisestä näkökulmasta on luvussa 4. Käytännön testauksella sain tuloksia ohjelmiston toimivuudesta ja samalla ne toimivat pohjana jatkokehitykselle. Testaus on kuvattu luvussa 5. Luku 6 sisältää työstä saadut tulokset ja järjestelmän toimivuuden arvioinnin. Lopun yhteenveto kokoaa vielä yhteen tiivistäen työssä tehdyt havainnot ja oman työn arvioinnin.

Luku 2

Työn konteksti

Valitsemani aiheen sekä työryhmän jäsenten taustojen vuoksi työ kytkeytyi erilaisiin konteksteihin. Aiheen puolesta kontekseja ovat VJ-kulttuuri ja sen piirissä käytetyt ohjelmat. Demokulttuuri on keskeinen konteksti tekijöiden taustasta johtuen ja tuottaa oman värinsä lopputulokseen ja tavoitteisiin. Avoimien ohjelmien kontekstiin työ kiinnittyy tekemieni ideologisten valintojen vuoksi.

2.1 VJ-kulttuuri

DJ eli *disc jockey* on henkilö, joka soittaa valmiiksi tuotettua musiikkia radiossa ja konserteissa. Jo pelkkä soitettavien kappaleiden temaattinen valinta tuottaa kokonaisuuksia. Tämän varioinnin lisäksi tehdään ns. remix-kappaleita, joissa yhdistellään olemassa olevista kappaleista kokonaisia uusia kappaleita. DJ:tä vastaava henkilö konserttien visuaalisen ilmeen luonnissa on VJ eli *video jockey*. Näiden kahden toimijan tehtävät ovat jossain määrin analogisia: siinä missä DJ käyttää mikseriä äänilähteiden yhdistelyyn, VJ yhdistää valmista videomateriaalia videomikserillä. Sittemmin suuntaus on molemmissa mennyt kohti digitaalista tuotantoa ja analogiset laitteet ovat vaihtuneet suurelta osin tietokoneohjelmiin.

VJ-kulttuuri on sikäli uusi ja hajanainen alue, että siitä ei ole olemassa merkittävää tutkimusta tai kirjallista aineistoa. Ensimmäisen kerran termi *VJ* on esiintynyt MTV-kanavalla 1980-luvun puolivälissä (Dekker, 2003). Alueen toimijoita kokoavat yhteen online-yhteisöt kuten keskeinen `VJCentral.com` tai tapahtumat, joista esimerkkeinä kansainvälinen *AVIT* ja kotimainen vastineemme *Pikseliähky*. Varsinaisten alan tapahtumien lisäksi sessioita järjestetään tyypillisesti elektronisen musiikin tapahtumien yhteydessä — VJ-kulttuuri kytkeytyykin elimellisesti klubi- ja teknokulttuureihin.

Musiikkivideot palvelevat lähtökohtaisesti samaa tarkoitusta kuin visuaalit: ne täydentävät musiikkia graafisin keinoin. VJ voi tuottaa videomateriaalia käyttäen täysin samoja menetelmiä ja ohjelmia kuin musiikkivideon tekijä. Ratkaiseva ero näiden kah-

den genren välille tulee esitysvaiheessa: musiikkivideo leikataan valmiiksi kokonaisuudeksi, joka toistuu samanlaisena kerrasta toiseen, kun taas live-tilanteessa VJ on esiintyjä siinä missä muusikotkin. Saman dualismin tuo esille lopputyössään myös Meskanen (2004). Tämä epälineaarisuus asettaa VJ:lle haasteita, mutta toisaalta myös tarjoaa mahdollisuuden osallistumiseen ja vapaaseen jammailuun.

2.1.1 Videomateriaali ja reaaliaika

VJ-toiminta on perinteisesti valmiisiin videoleikkeisiin perustuvaa. Koostamalla valmiista videoleikkeistä pidempiä kokonaisuuksia ja näitä efektoimalla saadaan käyttökelpoista materiaalia tuotettua suoraviivaisesti. Samaa metodia laajentaa reaaliaikaisen videokamerakuvan efektointi, joka tuottaa vaihtelevaa ja tunnistettavaa kuvaa esim. tanssijoista. Videomateriaalilla on teknisesti se etu, että lopullinen materiaali on laitteistolle kevyttä esitettävää ja toimii varmasti tasaisella päivitysnopeudella. Digitaalinen efektointi muuttaa tilannetta sikäli, että se vaatii reaaliaikaista laskentaa, joka voi olla hyvinkin raskasta.

Puhtaasti videomateriaaliin perustuva esitys ei ole omalla kohdallani kiinnostava lähtökohta. Kentällä on jo monta toimijaa ja tekniset ongelmat on jo ratkottu valtaosin, joten todellista uuden löytämistä voi tapahtua enää estetiikan saralla. Reaaliaikainen, ohjelmallisesti generoitu grafiikka tarjoaa periaatteessa paljon vapaammat ilmaisun keinot ja etenkin mahdollisuuden synkronoida visuaalit monimuotoisesti musiikkiin — toisaalta figuratiivisen esityksen luominen voi puolestaan olla vaikeaa. Ohjelmoijan rooli on rajoittunut videoitten esityksessä ja editoinnissa, kun taas generatiivista grafiikkaa käytettäessä ohjelmoija toimii luovana taiteentekijänä. Tämä oli merkittävä tekijä motivaationkin kannalta: en koe tyypillistä jakoa luoviin sisällöntuottajiin ja tekniikan hallitseviin ohjelmoijiin mielekkääksi.

2.2 Demokulttuuri

Demokulttuuri eli ”scene” syntyi 80-luvulla Commodore 64 -kotimikron valtakaudella. Tämä vähän tunnettu underground-taiteen alue on 20-vuotisen olemassaolonsa aikana seurannut aikansa teknistä kehitystä ja tuottanut osaajia niin peliohjelmoinnin kuin muunkin digitaalisen median alalle. Lönnblad (1998) käsittelee pro gradu -tutkielmassaan demokulttuuria musiikin harrastuksen muotona ja tarjoaa samalla haastatteluihinsa perustuvan suppean katsauksen kulttuurin historiaan. Ytimekkään johdatuksen aiheeseen suomalaisesta näkökulmasta ovat esittäneet myös mm. Saarikoski (2001) ja Faler (2001).

Demokulttuuri on tämän projektin keskeisiä kontekseja, sillä kaikki toteutukseen osallistuvat henkilöt ovat alkujaan toimineet juuri tämän kulttuurin piirissä, mikä osaltaan

vaikuttaa vahvasti työtapoihin, esteettisiin ihanteisiin ja arvostuksiin. Näyttävä lopputulos ja tekninen osaaminen ovat demokulttuurin piirissä arvostettuja seikkoja ja siten vähintäänkin taustalla vaikuttamassa. Taiteellinen ilmaisu jää helposti vähemmälle huomiolle, jolloin lopputuloksesta tulee ”eye candyä” — kaunista kuorta ilman sisältöä. Luvussa 5 esitellyt visuaalit ovat valtaosin peräisin omista demoistani. Osa Suomen VJ-hahmoista kytkeytyy demokulttuuriin, joten ei ole liioiteltua väittää, että demoharrastuksella kaikkiaan on oma roolinsa VJ-kulttuurin visuaalisessa ilmeessä ja yhteisöissä.

Ihmisten jakaminen erilaisiin tehtäviin on eräs demokulttuurin piirre. Keskeisimmät roolit ovat ohjelmoija, graafikko sekä muusikko. Näiden lisäksi esiintyy muitakin rooleja, mutta varsinaisessa demotuotannossa nämä kolme joskus päällekkäistä roolia ovat tärkeimmät. Omalla kohdallani tunnistan saman alitajuisen kategorisoinnin: projektiin osallistujia on tapana hahmottaa näiden kolmen roolin kautta, vaikka ne eivät realistisesti vastaakaan VJ:n tehtävien monimuotoisuutta.

2.3 VJ-ohjelmistot

Ennen työhön ryhtymistä kartoitin vastaavat ohjelmistot. Jos näistä olisi löytynyt sovelias alusta, ei oman kirjoittamiselle olisi ollut tarvetta. Seuraavissa alakohdissa esitellään lyhyesti muita järjestelmiä vahvuuksineen ja heikkouksineen. Mikään näistä ei osoittautunut suoraan sopivaksi ja toisaalta itse kirjoitetun ohjelmiston eduiksi voidaan lukea täysi hallittavuus sekä toteutusprosessissa karttunut oma ymmärrys.

2.3.1 Videoefektointiohjelmat

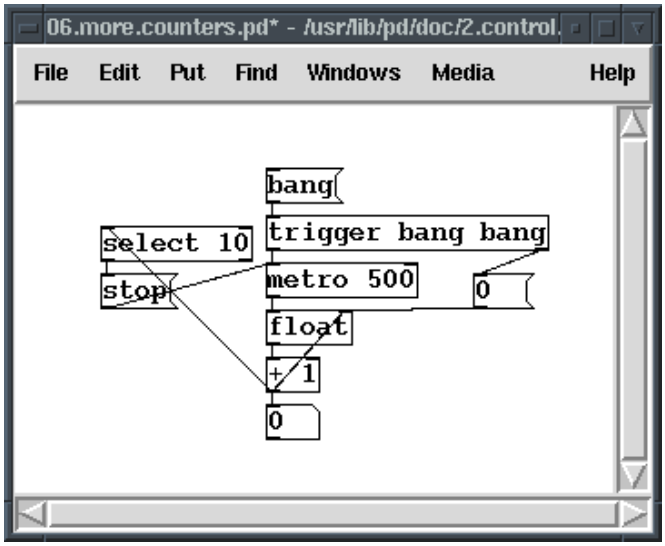
Resolume on Edvin de Koningin ja Bart van der Ploegin kehittämä kaupallinen VJ-työkalu (de Koning ja van der Ploeg, 2005). Ohjelma perustuu videoleikkeiden yhdistelyyn ja efektointiin eri tavoin. Nykyään Resolumessa on mahdollista hyödyntää myös Flash-animaatioita, jolloin sisältö voi olla dynaamisempaa. Reaaliaikaisen kamerakuvan käyttö ja ulkoisten kontrollerien käyttö lisäävät ohjelman käyttömahdollisuuksia. Kaupallisuutensa ja Windows-pohjaisuutensa johdosta Resolumella ei ollut projektille tarjottavaa muuten kuin ideoiden muodossa. Toinen kaupallinen samankaltainen ohjelma on *VJamm Pro*.

Myös avoimia videopohjaisia VJ-sovelluksia on olemassa. Näistä esimerkkeinä *EffecTV*, *FreeJ* sekä *Veejay*. Nämä ohjelmat perustuvat samaan toimintalogiikkaan kuin kaupallisetkin vaihtoehdot, joten niillä saavutetaan samat edut ja toisaalta ollaan samojen rajoitusten alaisia: videomateriaali on lopulta kuitenkin kiinteää ja kaksiulotteista eikä mahdollista täysin vapaata ilmaisua. Efektien määrässä nämä avoimet vaihtoehdot eivät jää jälkeen kaupallisista vastineistaan. Avoimuuden etuna on Viskijukan kan-

nalta se, että tarpeen vaatiessa näiden ohjelmien lähdekoodia voi käyttää suoraan, jos videopohjaiselle käsittelylle ilmenee tarvetta.

2.3.2 Graafipohjaiset video- ja äänisovellukset

Vapaasti kytkettävä *graafi* on Viskijukan perusrakenne. Graafi koostuu solmupisteistä ja näitä yhdistävistä poluista: arkipäivän graafeja ovat esim. tietokoneverkot. Tällainen rakenne sallii itsenäisten *moduulien* eli ohjelmiston osien vapaan yhdistelyn monimutkaisiksi kokonaisuuksiksi. Saman kaltainen rakenne on *Max/MSP*-ohjelmistossa, jota käytetään pääasiassa audion tuottamiseen ja analysointiin. Max/MSP tarjoaa loppukäyttäjälle graafisen käyttöliittymän, joten ohjelmointitaidon puute ei ole este ohjelman käytölle. Lopullinen ääni syntyy monimuotoisesta graafista, joka koostuu esim. äänilähteistä ja suodattimista. *Jitter*-nimisellä laajennuksella Max/MSP mahdollistaa myös graafisen esityksen luomisen tuotetusta äänestä. (Puckette, 1988)



Kuva 2.1: Yksinkertainen graafi Pure Data -ohjelmassa

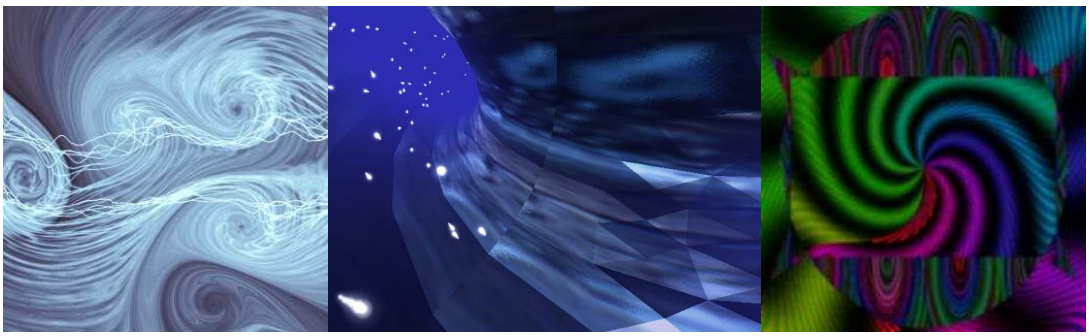
Max/MSP:lle on myös avoin vastineensa, *Pure Data* eli PD (Puckette, 1997). Epäkaupallinen Pure Data perustuu samaan toimintalogiikkaan ja on saatavissa useille eri laitealustoille kuten Windows, Linux, IRIX ja Mac OS X. Kuvassa 2.1 nähdään Pure Datan käyttöliittymä ja siihen ladattu esimerkkigraafi. Max/MSP-ohjelmiston alkuperäinen tekijä, Miller Smith Puckette, on myös tämän hankkeen alullepanija. *GEM*-niminen laajennus mahdollistaa jopa OpenGL-grafiikan käytön sovelluksissa. Harkitsin Pure Datan käyttämistä Viskijukan runkona, mutta täysi hallittavuus, oma kehittyminen ja kovat reaaliaikavaatimukset puolsivat oman alustan rakentamista. Jatkossa voin arvioida käyttömahdollisuuksia uudelleen, jos PD osoittautuu toimivaksi ja re-

aaliaikaiseen käyttöön riittävän tehokkaaksi. Kehittäjien yhteisössä toiminnasta olisi selviä hyötyjä testauksen ja ulkopuolisten tekemien laajennusten muodossa.

Max/MSP:n ja Pure Datan juuret ovat äänen tuottamisessa ja muokkaamisessa, joten niiden grafiikkaominaisuudet ovat suhteessa heikommat kuin ääniominaisuudet. Kuvadatalle vastaava graafipohjainen ohjelma on *EyesWeb*, joka puolestaan soveltuu ensisijaisesti videon käsittelyyn (Camurri et al., 1999). Graafisen käyttöliittymän kautta on mahdollista luoda videokuvaa käsitteleviä sovelluksia. EyesWeb on suljettu ohjelma ja lisäksi sidottu Windows-ympäristöön, joten sen käyttö ei tullut kyseeseen. Uusi kotimainen tulokas graafipohjaisten sovellusten sarjaan on Pauli Ojalan *Edo*, joka soveltuu suoraan VJ-käyttöön ja mahdollistaa kuvien, videon sekä 3D-grafiikan yhdistämisen (Ojala, 2004). Myös Edo on suljettu ohjelma ja toimii ainoastaan Mac OS X -käyttöjärjestelmässä.

2.3.3 Mediasoittimet

Suositut mediasoittimet *WinAmp*, *XMMS* ja *iTunes* mahdollistavat musiikin visualisoinnin laajennusten avulla. Kuvassa 2.2 on esimerkkejä tyypillisistä visuaaleista. VJ-alustan toteuttaminen tällaisena laajennuksena olisi mahdollista. Tämän lähestymistavan hyötynä on valmis tuki erilaisille äänitiedostoille ja suodattimille. Reaaliaikaisuus riippuu kuitenkin mediasoittimen toteutuksesta ja lisäksi järjestelmän toteuttaminen yksittäisen mediasoittimen laajennukseksi hankaloittaa suunnittelua, rajoittaa siirrettävyyttä eikä lopulta edes tarjoa ratkaisevia etuja rajoituksien vastapainoksi.



Kuva 2.2: WinAmp-visuaaleja. Geiss 2 (Ryan Geiss), Lino (Áron Gombás), The Psychedelic Screen Saver (Mike Irvine)

2.4 Vapaat ohjelmat

Toteutusryhmän ideologisista näkemyksistä ja käyttökelpoisen ulkoisen lähdekoodin saatavuudesta johtuen toteutin järjestelmän avoimena ohjelmana (open source softwa-

re). Ohjelman avoimuus tarkoittaa, että sen lähdekoodi julkaistaan vapaasti levitettäväksi. Vastaavia termejä eri vivahtein on muitakin: yleisesti käytetään myös ilmaisua *vapaat ohjelmat* (free software). Ohjelman avoimuuden katsotaan tämän aatesuunnan piirissä parhaiten palvelevan käyttäjiä ja suojaavan sekä tekijän että käyttäjien oikeuksia. Vastakohtana avoimille ohjelmille ovat ns. *suljetut ohjelmat*, joihin lukeutuvat useimmat kaupalliset sovellukset. Raymond (1999) käsittelee hakkerikulttuuria ja avoimien ohjelmien etuja keskeisessä teoksessaan *The Cathedral & the Bazaar*. Raymondin keskeisiä teemoja ovat luovan ongelmanratkaisijan eli *hakkerin* ihanne ja kehittäjäyhteisön merkitys ohjelmien evoluutiossa. Lisäksi hän esittelee palveluihin perustuvia taloudellisia malleja, joilla yrityksetkin voivat tuottaa avoimia ohjelmia ja silti olla kannattavia.

Avoimia ohjelmia yhdistävä piirre on niiden lisensointi. Tyypillisin lisenssi tällä hetkellä on Free Software Foundationin GPL — General Public License (FSF, 1992). GPL rajoittaa lähdekoodin käyttöä pääpiirteissään siten, että koodia on mahdollista käyttää vain GPL:n tai sen kanssa yhteensopivien lisenssien alaisissa ohjelmissa. Tärkein vaikutus rajauksella on se, ettei lähdekoodia voi hyödyntää suljetuissa ohjelmissa. Kuitenkin on tärkeää ymmärtää, että rajaus ei koske kaupallisia avoimia ohjelmia: ohjelmien myyntiä ei ole rajattu, mutta lähdekoodin pitää aina olla saatavilla. Lisenssin LGPL-variantti on ehdoiltaan löysempi ja sallii käännetyssä muodossa olevien komponenttien linkityksen myös suljettuihin ohjelmiin. Tällaisia komponentteja ovat tyypillisesti aliohjelmakirjastot.

Vapaat ohjelmat eivät ole enää pelkästään innokkaiden harrastelijoiden toisilleen kirjoittamia kuriositeetteja, vaan niistä on tullut koko ohjelmistoalaa muokkaava tekijä. Monet suuret ja laajasti käytetyt ohjelmat ovat avoimia, vaikka tavallinen käyttäjä ei tätä usein tiedostakaan. Sittemmin myös osa kaupallisista ohjelmistotaloista on ryhtynyt julkaisemaan tuotteitaan avoimilla lisensseillä. Seuraavassa esimerkkejä merkittävimmistä avoimista ohjelmista:

- Apache. Yleisin Internetin www-palvelinohjelmisto.
- GNU C Compiler (GCC). Optimoiva C-kääntäjä, jolla on tehty mm. Applen Mac OS X.
- LaTeX. Tieteellisen julkaisun perustyökalu.
- Linux. Unixin kaltainen, etenkin palvelimissa käytetty käyttöjärjestelmä.
- Mozilla Firefox. Netscape-selaimen jälkeläinen.
- OpenOffice. Sun Microsystemsin StarOfficeen pohjautuva täysimittainen toimisto-ohjelmisto.

Oman näkemykseni mukaan olemme tällä hetkellä murrosvaiheessa, jossa avoimet ohjelmat ovat tulossa sellaiselle kypsyyden ja monipuolisuuden tasolle, että tietokoneen

käyttäjä voi toimia pelkästään niiden varassa. Erikoistuneilla aloilla kuten palvelimissa näin on jo pitkään tehtykin. Avoimien ohjelmien käyttöönottoa rajoittavia tekijöitä ovat tällä hetkellä niiden vähäinen tunnettuus, käytettävyysongelmat sekä tukipalvelujen puute. Kaupalliset tahot hidastavat muutosta — tarkoituksella tai tahattomasti — suljettujen tiedostomuotojen, patentoinnin ja dokumentoimattoman laitteiston kautta.

Luku 3

Ohjelmistolle asetetut vaatimukset

Seuraavissa kohdissa esittelen järjestelmän suunnittelussa ja toteutuksessa huomioon otetut vaatimukset ja rajoitteet. Osa vaatimuksista, kuten esim. reaaliaikaisuus ovat käyttötilanteen sanelemia ja sitä kautta välttämättömiä, kun taas esim. toimintaympäristö ja työkalut valittiin pikemminkin oman mielenkiinnon ja ideologisten tekijöiden vuoksi. Kautta linjan näkyviä tavoitteita ovat tehokkuus, laajennettavuus ja joustavuus, jotka ovat samalla minkä tahansa hyvän ohjelmiston tunnuspiirteitä.

3.1 Käyttäjäryhmät

Ohjelmiston kehityksessä voidaan erottaa kolme vaihetta, joilla kullakin on omat käyttäjäryhmänsä. Ohjelmiston ensisijaisen käyttäjäryhmän muodostavat sen tekijät, joten suurta työmäärää ei tarvitse ensi vaiheessa uhrata helppolukuisen dokumentaation tai käyttöliittymän tekemiseen. Tekijät tuntevat järjestelmänsä ja osaavat tehdä siihen laajennuksia. Pienen ryhmän keskinäinen kommunikointi on suoraviivaista eikä vaadi hierarkiaa tai muuta hallinnointia.

Tilanne muuttuu jonkin verran siinä vaiheessa kun järjestelmä julkaistaan. Tässä vaiheessa tarvitaan aiempaa parempaa dokumentointia ja yhteydenpito mutkistuu, kun ulkopuoliset tahot tekevät lisäyksiään eivätkä kaikki kontaktit ole enää suoria. Ensimmäisiä potentiaalisia ulkopuolisia käyttäjiä ovat ohjelmointitaitoiset ihmiset, joilla on kykyä muokata ja hyödyntää lähdekoodia. Käyttäjäprofiili on seuraavanlainen:

- Kokemusta VJ-toiminnasta. Käyttäjä tuntee alan terminologian ja pystyy itse kokoamaan laitteistonsa.
- Hyvät ohjelmointitaidot. Käyttäjä osaa kääntää ohjelmia ja muokata niiden lähdekoodia.

- Vankka tekninen osaaminen. Käyttäjä osaa asentaa ohjelmia ja muokata järjestelmänsä asetuksia.
- Englannin kielen taito. Käyttäjä osaa sujuvasti teknistä englantia.

Viimeinen käyttäjäryhmä on ohjelmointia osaamaton ns. suuri yleisö, joka tarvitsee sujuvan käyttöliittymän, käyttöoppaan ja muuta helpottavaa opastusta. Tässä vaiheessa projekti ei ole suunnattu tällaiselle kohdeyleisölle vaan rakentaa ainoastaan teknologista alustaa, joka myöhemmin mahdollistaa loppukäyttäjällekin suunnatun ohjelmiston tekemisen.

3.2 Reaaliaikaisuus

Toimiakseen live-tilanteessa visuaalien täytyy reagoida reaaliaikaisesti musiikkiin. Toinen keskeinen vaatimus on nopea reagointi VJ:n antamiin komentoihin. Jos näitä kahta vaatimusta ei täytetä, alkaa viive musiikin ja grafiikan muutoksen välillä olla havaittava. Näiden *latenssien* minimointi on tärkeä osa järjestelmän suunnittelua ja jos niitä ei saada hyväksyttävälle tasolle, lopputulos ei ole käyttökelpoinen.

Reaaliaikaisuus kytkeytyy paitsi latensseihin, myös ohjelmiston suorituskykyyn. Käytännössä tämä tarkoittaa, että valitulla ohjelmointikielellä täytyy saada aikaan nopeaa ohjelmakoodia. C on ohjelmistoalalla pitkään käytössä ollut kieli, jolle on erittäin tehokkaita kääntäjiä, joten se täyttää tämän vaatimuksen. Lisäksi C-kielen valintaa tukee oma yli kymmenen vuoden kokemukseni sen parissa.

Ohjelmistotekniikassa on määritelty kahdenlaista reaaliaikaisuutta: kovaa ja pehmeää (Haikala ja Järvinen, 1994). Kova reaaliaikaisuus takaa sen, että suoritettava tehtävä valmistuu ennalta määrätyssä ajassa. Esimerkkinä kovista reaaliaikavaatimuksista musiikkiohjelmat: soitettava ääni ei saa katkeilla tai ohjelma on käyttökelvoton. Pehmeä reaaliaikaisuus puolestaan ei takaa tehtävän suoritusta kiinteän ajan kuluessa. Viskijukan tapauksessa vaatimukset ovat lähes kauttaaltaan pehmeitä, sillä mm. grafiikan piirron nopeus on joka tapauksessa vaihteleva. Ainoa kova reaaliaikavaatimus on äänen digitoinnissa: analyysin kannalta on tärkeää, että soitettu musiikki saadaan tarkasti talteen.

3.2.1 Latenssin lähteet

Latenssin suurimmat tuottajat ovat tiedon prosessointi ja laitteiston viiveet. Tiedon prosessointiin kuuluvia tekijöitä ovat äänen analysointi ja suodatus. Prosessointia voi nopeuttaa tehokkaammilla algoritmeilla, epätarkemmalla analysoinnilla ja jos muu ei auta, tehokkaammalla laitteistolla. Tavallisissa syöttölaitteissa kuten näppäimistössä ja hiiressä latenssi ei ole vielä merkittävä, mutta sen sijaan hitaaseen (31,25 kilobittiä

sekunnissa) MIDI-väylään kytkeytyvät laitteet ovat jo selvästi ongelmallisia. Hidas sarjamuotoinen liikenne tukkeutuu, jos tietoa täytyy välittää paljon.

Keskeinen latenssin tuottaja laitteistossa on äänipiiri. Äänen digitointi tapahtuu pieninä lohkoina eikä digitoinnin aikana vielä voida reagoida musiikissa tapahtuneisiin muutoksiin mitenkään. Lohkon koon pienentäminen on ilmeinen ratkaisu, mutta sekin tuottaa sivuvaikutuksia: pienten lohkojen käsittely kuormittaa järjestelmää enemmän eikä lyhyestä lohkosta saada yhtä paljon analyysille tärkeää tietoa kuin pitkästä. Käyttöjärjestelmä, äänipiirin ajurit sekä laatu rajaavat osaltaan käytettävää lohkon kokoa, sillä valmistajat eivät yleensä katso tarpeelliseksi minimoida latensseja kuin ainoastaan ammattilaistuotteissaan.

3.2.2 Grafiikan latenssi

Grafiikan osalta latenssia aiheuttavat piirtämiseen kuluva aika ja lisäksi näyttölaitteeseen synkronointi. Visuaalin piirtävältä ohjelman osalta kuluu aikaa piirtokäskyjen lähettämiseen grafiikkapiirille ja vastaavasti grafiikkapiiriltä kuluu aikaa piirtämiseen. Jotta ihminen tulkitsisi erilliset kuvat animaatioksi täytyy päivitystaajuuden olla vähintään luokkaa 15–20 kertaa sekunnissa. Esim. mykkäelokuvien tyypillinen päivitystaajuus on 16 Hz ja äänielokuvien 24 Hz (Parent, 2002, s. 493). Itse pyrimme vielä suurempaan taajuuteen pehmeämmän liikkeen aikaansaamiseksi.

Kuvan virkistys tapahtuu videotykeillä ja monitoreilla tyypillisesti 50–100 kertaa sekunnissa. Jos tietokoneen piirtämää kuvaa ei synkronoida mitenkään tähän virkistykseen, tulee esitettävään grafiikkaan häiritsevää vapinaa. Tavallisin ratkaisu tähän ongelmaan on se, että piirto tapahtuu vuorotellen kahteen puskuriin, joista toista näytetään ja toiseen piirretään (ns. *double buffering*). Puskurien vaihto piirron valmistuttua synkronoidaan ruudun virkistykseen.

3.3 Toimintaympäristö ja siirrettävyys

Ensisijainen tavoite on, että alusta toimii Linux-ympäristössä. Linuxin ohjelmointirajapinnat ovat peräisin Unix-ympäristöistä, joten ohjelmakoodista tulee samalla siirrettävää muihin Unix-ympäristöihin ilman eri vaivaa. Mac OS X on kohdealustana järjestyksessä toinen: Unix-pohjaisuutensa ansiosta se ei vaadi lähdekoodiin suuria muutoksia ja myös käytettävät ohjelmointityökalut ovat samat. Siirrettävyyden kannalta C on käyttökelpoinen valinta ohjelmointikieleksi, sillä mihin tahansa realistiseen kohdeympäristöön on olemassa C-kääntäjä.

Siirrettävyyttä ja avoimuutta tukee myös avoimien ohjelmointirajapintojen käyttö. Ehdoton vaatimus on, että 3D-grafiikka piirretään OpenGL-rajapintaa (Shreiner et al., 2003) käyttäen. Syöttölaitteiden ja ikkunoinnin käsittelyyn valitsin Simple DirectMedia Layer -kirjaston (Lantinga, 2005) sen siirrettävyyden ja monipuolisuuden vuoksi.

SDL mahdollistaa myös äänen soittamisen mutta ei sen digitointia, joten digitointiin käytetään siirrettävää Portaudio-kirjastoa (Bencina ja Burk, 2001). Muiden käytettävien kirjastojen tulee niin ikään olla avoimia ja sopivalla lisenssillä julkaistuja.

3.4 Avoimuus

Avoimen ohjelmistonkehityksen idean mukaisesti järjestelmän lähdekoodi tulee julkaista avoimella lisenssillä. Tämä asettaa rajoituksia käytettävien komponenttien suhteen: kaupallista tai rajoittavalla lisenssillä julkaistua lähdekoodia en voi käyttää. Valinnasta ei koidu sanottavia ongelmia, sillä projektin puitteissa ei ole muutenkaan varaa tai tarvetta ostaa ohjelmia. Sen sijaan Internetistä saatavien valmiiden avoimien komponenttien käytöstä on merkittävää hyötyä. Koska itselläni ei ole taloudellisia sidosryhmiä projektin suhteen, ei lähdekoodin julkaisu ole ongelmallista. Jotta julkaistusta alustasta olisi mahdollisimman suuri hyöty kansainväliselle yhteisölle, täytyy lähdekoodin kommentointi ja dokumentointi tehdä englanniksi.

3.5 Viihdyttävyys

Konsertissa visuaalien rooli on ennen kaikkea täydentävä ja viihteellinen. Pohdin seuraavissa alakohdissa viihdyttämisen olemusta ja sen merkitystä ohjelmiston suunnittelun ja toteutuksen kannalta.

3.5.1 Mielenkiinnon säilyttäminen

Viihtymisen, kiinnostavuuden ja hauskuuden asettamia vaatimuksia lähdin hahmottamaan Ulyaten ja Bianciardin esittämän kymmenen käskyn listan mukaan. Tätä listaa käytettiin suunnitteluperiaatteena rakennettaessa SIGGRAPH'98-konferenssiin laajamittaista interaktiivista installaatiota nimeltä *The Interactive Dance Club* (IDC). IDC:ssä keskeisenä ajatuksena oli tarjota tanssijoille yksinkertaista ja viihdyttävää toimintaa, joka tuotti välitöntä visuaalista palautetta. Kuvassa 3.1 on esimerkkejä IDC:n eri alueista. (Ulyate ja Bianciardi, 2002)

Ulyaten ja Bianciardin kymmenen käskyä olivat seuraavat:

1. Käyttöliittymän ja sisällön tulee rohkaista liikkumaan ja palkita siitä.
2. Osallistujan toimitaan vastataan välittömällä ja tunnistettavalla reaktiolla.
3. Ohjeita ei sallita. *Toiminnan tulee olla niin intuitiivista, ettei käyttöohjeita tarvita.*

Kuva 3.1: The Interactive Dance Clubin alueita. Orbs, Stomp ja Tire-o-mania.

4. Osallistujan ei tarvitse olla asiantuntija.
5. Ajattelua ei sallita. *Aktivoidaan kehollista eikä älyllistä toimintaa.*
6. Toiminnasta seuraa esteettisesti miellyttävä palaute.
7. Toiminnan on oltava yksinkertaista, välitöntä ja hauskaa.
8. Reagointi on tärkeämpää kuin esityksen laatu.
9. Ajattele modulaarisesti. *Ohjelmiston tulee koostua komponenteista.*
10. Tarkkaile ja opi. *Käytännön havainnointi hyödyttää jatkokehitystä.*

Vaikka listaa ei ole alkujaan tarkoitettukaan VJ-käyttöön, tarjoaa se hyvän perustan viihdyttävyyden ja interaktion ongelmien lähestymiseen. Toistuvia ja korostettuja teemoja ovat erityisesti ärsykkeisiin reagointi ja toiminnan luonnollisuus. Visuaaleihin sovellettuna nämä merkitsevät musiikkiin ja ohjaukseen kytkettyä välitöntä ja tunnistettavaa reagointia. On kuitenkin tärkeää havaita, että esitetty lista on suunniteltu nopean tanssimusiikin tukemiseen, joten sen soveltaminen suoraan muihin musiikin lajeihin ei ole välttämättä mielekästä.

3.5.2 Viihdyttävyys ohjelmiston kannalta

Viihdyttävyyden abstraktit vaatimukset täytyy toteutusta varten pystyä ilmaisemaan tarkkoina teknisinä vaatimuksina. Ohjelmiston tulee mahdollistaa visuaalien liikkeen, värien ym. ominaisuuksien monipuolinen ohjaus esitystilanteessa: osaa tapahtumista kontrolloi musiikki, osaa VJ manuaalisesti. Ohjelmiston kannalta tämä vaatimus merkitsee sitä, että järjestelmän osien välisen kommunikoinnin on tapahduttava yhtenäisellä tavalla. Yksittäisen visuaalin toteuttajan tulee puolestaan tarjota visuaalille säädettäviä parametreja, jotta esitystä olisi mahdollista jatkuvasti varioida.

Järjestelmän osien välisen hierarkian on niin ikään tuettava joustavuutta. Vaihtelevan esityksen takaamiseksi osia on pystyttävä reaaliajassa kombinoimaan uusilla, odottamattomilla tavoilla. Ohjelmiston on täten koostuttava itsenäisistä moduuleista, kuten myös Ulyate ja Bianciardi (2002) esittivät. Dynaamisesti muuttuva järjestelmä on toteutukseltaan ohjelmistoteknisesti staattista haastavampi — erityisesti luotettavuuden täytyy olla kunnossa tai muutoin koko esitys voi vaarantua ohjelmiston vikaantumisen takia.

Luku 4

Ohjelmiston kuvaus

Tässä luvussa kuvaan järjestelmän rakenteen ja käytännön toiminnan ohjelmistotekniseltä kannalta. Esitys on tarkoituksella lyhyt ja yleisluontoinen, jotta sen osuus ei nousisi kirjallisessa osassa epäoleellisen suureksi ja jotta tekniikkaa tuntematonkin lukija siitä hyötyisi. Käsiteltävät asiat ovat ohjelmiston yleinen rakenne, dynaaminen laajentaminen ja ajonaikainen toiminta.

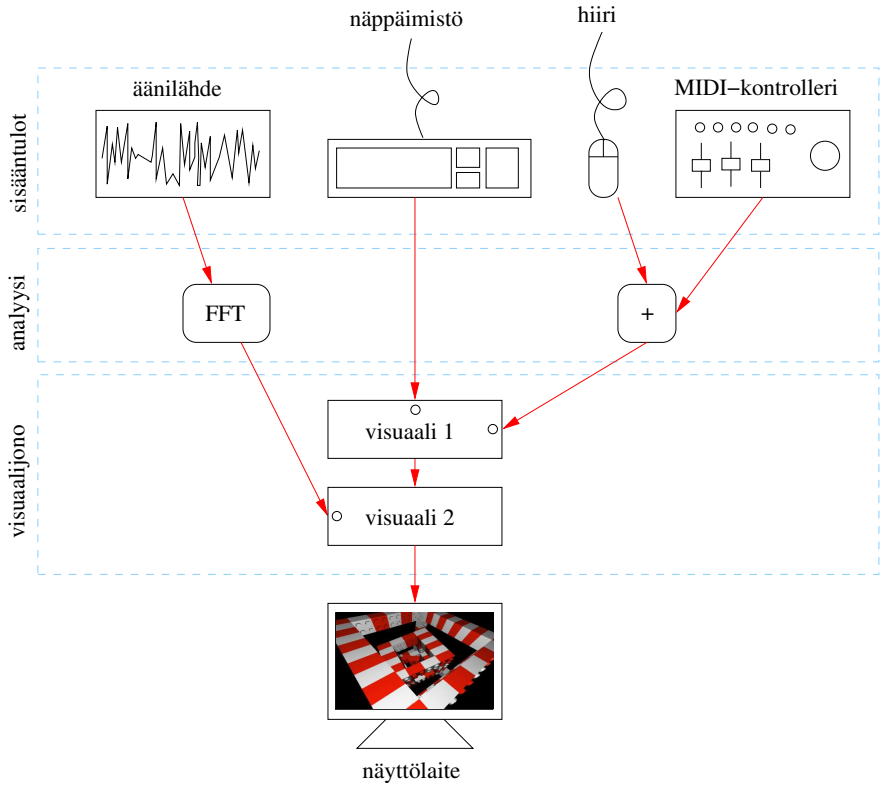
4.1 Järjestelmän osat

Ohjelmiston rakennetta ja toimintaa voi hahmottaa erilaisten kriteerien kautta: jaottelua voi tehdä esim. kontrollin ja tiedon välittymisen, tilasiirtymien tai ohjelmiston loogisten kokonaisuuksien kautta. Tässä kohdassa valittu lähestymistapa perustuu ohjelmiston loogisiin kokonaisuuksiin siten, kuin ne näyttäytyvät ohjelmoijalle. Kontrollin ja tiedon välittyminen selkenee valtaosin alakohtien 4.1.2 ja 4.3 perusteella.

Vaatimusten mukainen joustava perusrakenne ohjelmassa on mahdollisimman vapaasti kytkettävä graafi. Ohjelma muodostuu moduuleista, joita yhdistetään toisiinsa tiedon ja kontrollin välittämistä varten ns. *kanavilla*. Järjestelmän yleisrakenne näkyy kuvassa 4.1. Tällä hetkellä graafi on käytössä vain ohjelmiston sisäisenä rakenteena eikä vielä käyttöliittymän tasolla.

4.1.1 Ydin

Ydin on järjestelmän osa, joka huolehtii keskitetysti kontrollin etenemisestä, lataa moduulit ja tarjoaa muille osille yhteisiä palveluja. Ohjelman käynnistyessä kontrolli on juuri ytimellä ja se alustaa muut osat käyttökuntoon. Ytimen tarjoamat palvelut ovat myös ohjelmoijalle pääasiallinen rajapinta, jolla Viskijukkaa käytetään.



Kuva 4.1: Järjestelmän rakenne

4.1.2 Kanavat ja viestit

Jokaiseen moduuliin kuuluu joukko kanavia, joilla moduuli sekä lukee itselleen saapuvaa tietovirtaa että välittää sitä edelleen eteenpäin. Yhden moduulin ulostulo kytketään toisen sisäänmenoon, jolloin voidaan rakentaa pitkiä ketjuja, jotka kokonaisuuksina muokkaavat dataa monimutkaisilla tavoilla. Tyypillinen esimerkki tästä on äänimoduulilta peräisin oleva äänidata, jota suodatetaan, analysoidaan ja jaetaan muiden moduulien kesken. Yleinen rakenne muistuttaa pitkälti diplomityössäni suunnittelemani äänijärjestelmän rakennetta (Reunanen, 2002).

Kanavissa välittyvää tietopakettia kutsutaan nimellä *viesti*. Viesti sisältää lähteestä riippuen erilaista tietoa, kuten lukuarvon, ääninäytteitä tai kuvan. Standardoitujen lukuarvojen käyttö on tarpeen vapaan kytkennän vuoksi: jos hyvinkin erilaiset kontrollerit ja analysaattorit tuottavat samanlaisia arvoja, voidaan niitä kytkeä vapaasti visuaalien eri sisäänmenoihin. Keskeiset lukualueet ovat [0..1] ja [-1..1]. Näistä ensimmäinen soveltuu esimerkiksi liukusäätimen tai äänenvoimakkuuden välittämiseen, kun taas jälkimmäistä voidaan käyttää mm. ääninäytteille ja ohjaussauvan suunta-arvoille. Jatkuva-arvoisen datan lisäksi järjestelmässä on tarpeen välittää *komentoja*, joita aiheutuu vaikkapa hiiren nappien painamisesta.

4.1.3 Laajennusten lataaja

Tämän ohjelmiston osan tehtävänä on ladata moduulit tiedostoista osaksi järjestelmää. Toiminta on suoraviivaista: latauspyynnön saatuaan lataaja lukee levyltä moduulin ja liittää sen osaksi järjestelmää. Käyttöjärjestelmäkohtaiset ominaispiirteet on tarkoituksella piilotettu rajapintaa käyttävältä ohjelmoijalta, jotta lähdekoodi pysyisi mahdollisimman samana ympäristöstä toiseen. Laajennusten lataajan muut velvollisuudet ovat moduulien instantiointi ja poisto järjestelmästä. Instantiointi kahdentaa järjestelmässä olevan moduulin, jolloin sitä voidaan käyttää riippumattomasti kahdessa ohjelmiston eri osassa.

4.2 Dynaaminen laajentaminen

Vaatimuksissa esitetty laajennettavuus on toteutettu jakamalla ohjelmakoodi dynaamisesti ladattaviin laajennusmoduuleihin eli *plugineihin*. Tällaisesta kapseloinnista seuraa jonkin verran ylimääräistä työtä, sillä moduulien kommunikointi kanavien kautta on hankalampaa ja hitaampaa kuin suora funktioiden kutsuminen. Toisaalta ratkaisu mahdollistaa järjestelmän ajonaikaisen laajentamisen ja moduulien vapaan yhdistelyn. Yksittäisen laajennusmoduulin voi kirjoittaa ja kääntää ilman, että muuhun järjestelmään tarvitaan muutoksia. Tarpeettomat laajennukset voidaan jättää lataamatta, jolloin säästetään muistia.

Jokaisen laajennuksen täytyy sisältää vähintään alustusfunktio, joka saattaa moduulin toimintakuntoon sekä ajofunktio, joka käsittelee moduulin saaman tietovirran ja välittää sen eteenpäin. Näiden kutsuminen tapahtuu kaikille laajennuksille yhtenäisellä tavalla, joten ytimen ei tarvitse tietää erikseen niiden erityispiirteistä. Liitteessä B on yksinkertaisen laajennuksen lähdekoodi, josta ohjelmointitaitoinen voi nähdä, että omien laajennusten kirjoittaminen ei ole monimutkaista eikä työlästä. Esittelen järjestelmän toiminnan kannalta keskeisimmät moduulit seuraavissa alakohdissa.

4.2.1 Äänen digitoija

Digitoija lukee tietokoneen äänisisääntulosta äänidataa ja välittää sitä eteenpäin muulle järjestelmälle standardimuotoisina viesteinä. Moduulin keskeinen ominaisuus on säädettävä äänilohkon koko, joka vaikuttaa latenssiin ja äänianalyysin laatuun (katso alakohta 3.2.1). Digitoijan on toimittava itsenäisesti omassa säikeessään tai keskeytyksessä, jotta äänidataa saadaan luettua järjestelmään katkeamatta. Varsinainen äänen analysointi ei tässä toteutuksessa sijoitu digitoijaan, jotta rakenne säilyisi paremmin modulaarisena, mutta jatkossa tilanne voi muuttua, jos latenssin minimointi katsotaan tärkeämmäksi kriteeriksi.

4.2.2 Syöttölaitteet

Näppäimistö ja hiiri ovat tyypillisimmät syöttölaitteet ja aina saatavilla. Näppäimistö soveltuu hyvin komentojen antamiseen ja siitä huolehtiva moduuli tuottaakin komentoviestejä. Hiirimoduuli tuottaa sekä koordinaatteja että komentoja (liike ja painikkeet). Testausvaiheessa ja pienen mittakaavan visualisoinnissa näppäimistö ja hiiri riittävät VJ:n käyttöön, mutta parametrien kontrolli on vapausasteiden vähyyden vuoksi työlästä ja rajoittunutta.

Tehokkaampia syöttölaitteita ovat joypad ja MIDI-ohjain. Molemmat tuottavat sekä komentoja että jatkuva-arvoista dataa. Joypad tuottaa komentoja painikkeistaan ja koordinaatteja ohjaussauvoista sekä liipaisimista. MIDI-ohjaimissa on monenlaisia säätimiä kuten liukusäätimiä, potentiometreja, painikkeita ja koskettimia, joilla saadaan aikaan monimuotoinen kontrolli. Laitteelta saadut standardimuotoiset MIDI-viestit täytyy tulkita moduulissa, jotta niistä saadaan järjestelmälle merkityksellisiä arvoja. Ohjelmoijalle monipuoliset syöttölaitteet näyttäytyvät samanlaisina: moduuleissa on useita erillisiä kanavia, jotka välittävät lukuarvoja ja komentoja. Näin yksittäinen säädin voidaan vapaasti kytkeä suoraan yksittäisen visuaalin haluttuun parametriin: esimerkkinä vaikkapa grafiikan RGB-arvon säätäminen kolmella liukukytkimellä.

4.2.3 Visuaalit

Visuaalit eivät poikkea muista moduuleista kuin sen osalta, että ne eivät tyypillisesti välitä saamiaan viestejä mihinkään vaan käyttävät niitä ainoastaan parametreinaan. Yksittäinen visuaali piirtää ruutupuskuriin parametriensa määräämän grafiikan, jonka jälkeen kontrolli palaa kutsujalle. Useita visuaaleja peräkkäin ajettaessa voidaan edellisen ruutumuistiin jättämää grafiikkaa käyttää syötteenä seuraavalle, jolloin lopputuloksena saadaan yksinkertaisillakin visuaaleilla monimutkainen esitys.

Visuaalit tuottavat näyttölaitteella esitettävän grafiikan käyttäen OpenGL-rajapintaa (Shreiner et al., 2003). OpenGL:n monimutkainen tilanhallinta aiheuttaa sen, että edellinen visuaali saattaa jättää seuraavalle väärät asetukset, jolloin lopputulos on vaikeasti ennakoitavissa. Moinen virhetoiminta voi tuottaa mielenkiintoisia, odottamattomia efektejä, mutta vähintään yhtä suurella todennäköisyydellä myös huonon tuloksen tai pahimmillaan grafiikkaohjelmoijille tutun mustan ruudun. Virheellisen toiminnan välttämiseksi visuaalin täytyy tallentaa ja palauttaa kaikki käsittelemänsä tilat — yksinkertaisimmin ja luotettavimmin tämä onnistuu kaikkien OpenGL:n tilojen palauttamisella.

4.3 Ajonaikainen toiminta

Aivan aluksi järjestelmä on tarpeen alustaa, jotta globaalit asetukset saadaan oikeiksi. Alustuksen jälkeen ladataan laajennukset yksitellen. Ladattu moduuli ei vielä itsessään ole käyttökelpoinen, vaan toimii aihiona ajon aikana käytettäville. Kun graafiin halutaan lisätä uusi moduuli, luodaan se käyttämällä aihiota pohjana, jonka jälkeen moduuli on käyttövalmis. Tämän jälkeen on vielä tarpeen kytkeä uuden tulokkaan sisäänmeno- ja ulostulokanavat, jotta moduuli tulisi ylipäänsä ajetuksi. Tämä kanavien kytkentä muodostaa lopulta kokonaisen käyttöön valmiin graafin.

Kuvassa 4.1 esitetyn rakenteen mukaisesti osa moduuleista on *visuaalijonossa* tai laajemmin ottaen aktiivisten moduulien listassa. Kun järjestelmän graafia ryhdytään käymään läpi (traversoimaan), kontrolli etenee järjestyksessä näistä moduuleista. Aktiivisten moduulien saamat viestit selvitetään siten, että niiden sisääntuloista edetään rekursiivisesti yksi kerrallaan kohti lehtisolmuja eli käytännössä syöttölaitteita ja äänisisääntuloa. Kun lehtisolmut on saavutettu, kontrolli palaa käänteisessä järjestyksessä: hierarkiassa aiempi moduuli ajetaan ja se tuottaa seuraavalle sisääntulokanavaan viestejä. Lopulta kontrolli palaa alkuperäiseen moduuliin, jonka jälkeen toimitaan vastaavasti seuraavalle aktiiviselle moduulille kunnes ne loppuvat.

Live-tilanteessa moduuleja lisätään ja poistetaan dynaamisesti visuaalijonosta, jolloin graafinen lopputuloskin muuttuu. Samoin kanavien kytkentöjä muutellaan, jolloin äänianalyysin ja VJ:n toimintojen vaikutus kytkeytyy uudella tavalla visuaaleihin ja näiden parametreihin.

Luku 5

Käytännön testaus

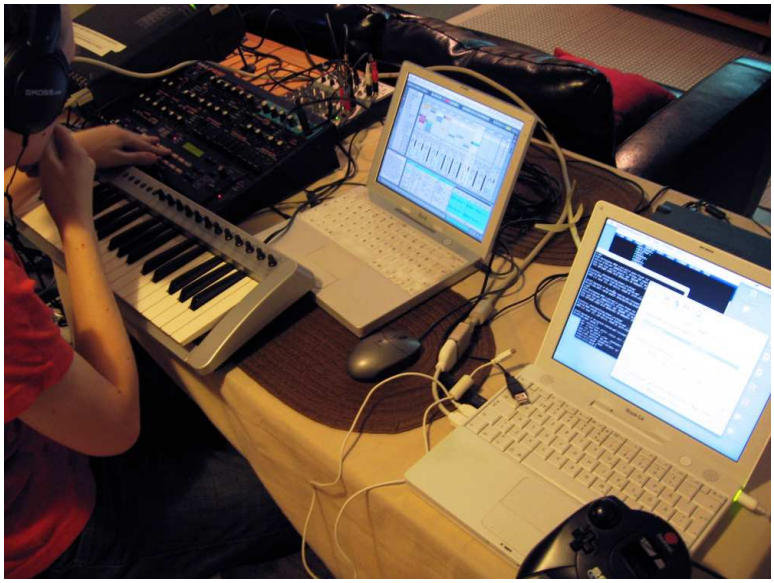
Ohjelmiston testauksella pyrin arvioimaan sen luotettavuutta ja soveltuvuutta käyttötarkoitukseensa. Itse suorittamani testauksen lisäksi kokeilin ohjelmaa myös käytännön tilanteessa: pelkässä laboratoriotestauksessa saadut tulokset eivät vastaa livetilanteen yllätyksiä. Esityksen kiinnostavuuden arviointi vaatii sekin yleisöä.

Projektin kiireisen aikataulun vuoksi en voinut järjestää laajamittaista testitilanteiden sarjaa, joten pitkän linjan kokemukset järjestelmän käytöstä jäivät puuttumaan. Ohjelman kehittyminen on joka tapauksessa iteratiivinen prosessi ja esiintyjäksi harjaantuminen vaatii aikaa. Nämä rajoitteet huomioiden testauksesta saamani havainnot olivat kaikkiaan hyödyllisiä ja ne tarjosivat omalta osaltaan ideoita jatkokehitykseen.

5.1 Testitilanteen kuvaus

Ohjelmiston ensimmäinen käytännön testaus tapahtui 11.3.2005. Testitilanne oli pienimuotoinen 40 minuuttia kestänyt groove/house-tyylinen jammailu, jossa esiintyi kaksi muusikkoa. Jarkko Rotstén toimi sessiossa DJ:nä ja vastasi taustoista. Yrjö Fager soitti efektoitua MIDI-kitaraa. Itse tuotin session visuaalit käyttäen pelkästään Viskijukan tarjoamia mahdollisuuksia.

Ohjelmistoa ajettiin kannettavalla Apple iBook G4 -tietokoneella, jonka kuva ohjattiin videotykille. Tietokoneessa ei ole äänisisäänmenoa, joten digitointiin käytettiin Creative Labsin valmistamaa ulkoista Extigy-äänikorttia, joka liitettiin koneeseen USB-liitännän avulla. Äänen käsittelystä USB-väylän yli syntyi hieman ylimääräistä latenssia verrattuna sisäänrakennettuun äänikorttiin. Visuaalien kytkeminen päälle ja pois tapahtui näppäimistöltä ja muu ohjaus pääosin USB-väylään kytketyllä Sega Dreamcastin peliohjaimella. Peliohjaimessa oli viisi painonappia, yksi analoginen sauva, kaksi analogista liipaisinta sekä digitaalinen ristikko-ohjain. Kuvassa 5.1 näkyy testitilanteessa käytettyä laitteistoa. Lisää kuvia liitteessä C.

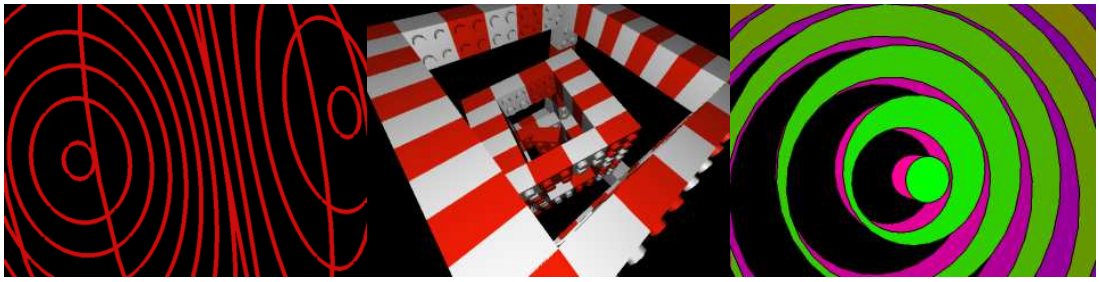


Kuva 5.1: Testitilanteessa käytettyä laitteistoa

5.2 Esimerkkejä visuaaleista

Kuvassa 5.2 on esimerkkejä testitilanteessa käyttämistäni visuaaleista, joita oli mahdollista näyttää joko yksitellen tai toistensa kanssa päällekkäin. Kaikille yhteinen ominaisuus oli pyöriminen, jota käytettiin erilaisina variaatioina 3D-kappaleiden ohjaamattomassa liikkeessä. Visuaaleista monipuolisimmaksi osoittautui *Sphere*, koska sillä pystyi tuottamaan ulkoasultaan selvästi vaihtelevinta jälkeä. Spherellä on lukuisia parametreja kuten katseluetäisyys, värisävy, kirkkaus ja pyörimisakseli. *Cubes* ja *Spiral* olivat puolestaan ulkomuodoltaan realistisia ja tunnistettavia, mutta niissä ei ollut reaaliajassa säädettäviä parametreja. Tämä muokkausmahdollisuuden puute oli niiden käytössä ilmeinen ongelma. Pelkkä digitoidun äänikäyrän näyttäminenkin osoittautui yllättävän toimivaksi keinoksi, sillä sen reagointi musiikkiin on selvästi tunnistettavaa ja välitöntä.

Varsinaisten visuaalien lisäksi oli käytössä kaksi efektiä, joilla lopputulosta sai edelleen muokattua. Niinkin triviaali efekti kuin edellisen piirretyn kuvan ruudulle jättäminen tarjosi jo muuntelumahdollisuuksia, kun visuaaleista jäi ruudulle tunnistettavaa jälkeä. Toinen käytetty efekti oli yksinkertainen tekstuurin kopiointiin perustuva *Shine*, jonka avulla grafiikkaan sai vaikutelman hehkumisesta. Kuvassa 5.3 on esimerkkejä Sphere-visuaalin muokkaamisesta näillä kahdella efektillä.



Kuva 5.2: Esimerkkejä visuaaleista. Sphere, Cubes ja Spiral.

Kuva 5.3: Efektien vaikutus. Shine, päällepiirto ja näiden yhdistelmä.

5.3 Välittömät havainnot

Ohjelmisto toimi luotettavasti. Vakavia vikoja kuten kaatumista tai hidastumista ei ilmennyt ja lisäksi havaitsemani ongelmat olivat pieniä ja helposti korjattavia. Viestit kulkivat kanavissa, visuaalit toimivat odotetusti ja kaikkiaan VJ:n toiminta välittyi tunnistettavasti lopulliseen esitykseen. Omien havaintojeni ja saamani palautteen mukaan sessio oli hauska ja tunnelmaltaan rentoutunut.

40 minuuttia osoittautui pitkäksi ajaksi. Vaikka ajantajuni katosikin hetkittäin jammailun aikana, niin esityksen pitäminen jatkuvasti kiinnostavana oli haastavaa. Parhaita apuja tämän ongelman ratkaisuun olivat parametrien säätely (etenkin Spheren tapauksessa), visuaalien uusi yhdistely ja lopputuloksen efektointi. Musiikkityyli tarjosi säännöllisyytensä puolesta sopivia synkronointimahdollisuuksia kuten visuaalien kytkeminen päälle ja pois tahtien mukana. Äänikäyrän näyttämistä lukuun ottamatta kaikki synkronointi tapahtui käsiohjauksella, sillä nopeasti toteutettu äänenvoimakkuuteen perustuva ohjaus tuotti räpsyvän ja epämiellyttävän tuloksen. Musiikin ennalta kuuntelu olisi helpottanut esityksen suunnittelua, mutta vapaan improvisoinnin tapauksessa moinen ei ollut mahdollista.

Peliohjain ei tarjonnut riittävästi vapausasteita monipuoliseen kontrollointiin ja lisäksi sen yhteiskäyttö näppäimistön kanssa oli hetkittäin vaikeaa, sillä ohjain vaati kahden

käden otetta. Liipaisimet ja analoginen sauva kytkeytyivät sinänsä luonnollisesti visuaalien parametreihin, mutta liikealueen pienuus haittasi tarkkaa ohjaamista.

Luku 6

Tulokset ja niiden arviointi

Ohjelmiston suunnittelu, toteutus ja käytännön testaus tuottivat tuloksia luvussa 1 esittelemiini tutkimusongelmiin. Seuraavissa kohdissa kokoan saamani tulokset yhteen ja arvioin niiden merkitystä. Prosessin aikana karttunut näkemys avasi uusia näköaloja kentän ongelmiin, mikä puolestaan loi pohjaa jatkokehityksen suunnitteluun.

6.1 Vastauksia tutkimusongelmiin

Tutkimusongelmat olivat keskeinen motivaattori koko projektin tekemiselle: niihin saadut vastaukset olivat henkilökohtaisestikin kiinnostavia. Rajallisen ajan puitteissa en saanut asetettamiini laajoihin tutkimuskysymyksiin kattavia vastauksia, vaan pikemminkin vastauksen osia sekä ajatuksia siitä, miten ongelmia voisi jatkossa lähestyä.

6.1.1 Avoimet ohjelmat ja reaaliaikainen media

Oman roolini johdosta avoimet ohjelmat täyttivät tarpeeni erinomaisesti. Sekä Linux- että Mac OS X -ympäristössä käytössäni oli sama GNU C Compiler (GCC), joka toimi luotettavasti ja tuotti tehokasta ohjelmakoodia. Kyseinen kääntäjäympäristö on itselleni tuttu jo seitsemän vuoden ajalta, joten oppimiskynnystä ei ollut. Ohjelmointityökalujen osalta sama kuvio toistuu suuremmassakin mittakaavassa: ohjelmointiin on saatavilla suuri määrä avoimia työkaluja kuten kääntäjiä, tulkkeja, editoreita, apuohjelmia ja aliohjelmakirjastoja.

Kohdassa 2.3 käsittelin VJ-ohjelmia, joista osa oli avoimia. Vielä tämän tason käyttökelpoisia työkaluja on olemassa, mutta kaikille suurille kaupallisille ohjelmille ei ole vastinetta. Erityisesti tämä pitää paikkansa pikseligrafiikan, videon käsittelyn, 3D-grafiikan ja äänituotannon osalta. Harrastajavoimin ei ole toistaiseksi pystytty saavuttamaan parhaiden kaupallisten ohjelmien tasoa. Pikseligrafiikan alueella vahvin avoin

ohjelma on *GIMP* (GNU Image Manipulation Program), jota voi realistisesti verrata muutaman vuoden takaiseen *Adobe Photoshopiin*. 3D-grafiikan alalla vastaavassa asemassa on *Blender*, joka on alkujaan kaupallinen ohjelma. Ääni- ja etenkin videotuotantoon soveltuvat ohjelmat ovat tällä hetkellä joko ominaisuuksiltaan rajallisia tai vasta kehitysvaiheessa.

Kun sovelluksia tehdään ei-teknisille loppukäyttäjille, nousee käytettävyys keskeiseen rooliin. Laajamittainen käytettävyystestaus puolestaan on kallista ja vaatii erityisosaamista, joten avoimien ohjelmien tekijöillä on harvoin mahdollisuutta sen suorittamiseen. Nichols ja Twidale (2003) käsittelevät artikkelissaan tätä ongelmaa ja toteavat, että heikko käytettävyys on merkittävä este avoimien ohjelmien laajalle käyttöönotolle. He esittävät useita eri lähestymistapoja avointen ohjelmien käytettävyyden parantamiseen, mutta mikään niistä ei ole yksinään riittävä. Käyttäjien tarpeiden tunteminen, käytettävyyden merkityksen tunnustaminen ja toteuttajien asiantuntemuksen lisääminen ovat menetelmistä riippumatta ne keinot, joilla ohjelmia voidaan parantaa.

Avointen ohjelmien soveltuvuus reaaliaikaisen median tuottamiseen riippuu käyttäjän tarpeista: ohjelmointiin, grafiikan ja äänen käsittelyyn on olemassa käyttökelpoisia ohjelmia, mutta niiden rajoitukset täytyy tiedostaa ja lisäksi varautua osoittamaan itse aktiivisuutta. Kaupallisen ohjelman mukana saa käyttöönsä tukipalvelut, joiden vastineena avoimien ohjelmien puolella ovat kehittäjien erilaiset foorumit. Suuria kaupallisia projekteja ei välttämättä ole mahdollista toteuttaa täysin avoimilla ohjelmilla, mutta pienimuotoisemmassa käytössä jo pelkät kustannussäästöt voivat olla riittävä peruste niiden hyödyntämiseen.

6.1.2 Ohjelmiston rakenne

Suunnattu graafi osoittautui testauksen perusteella joustavaksi rakenteeksi, joka lisäksi tukee erinomaisesti moduulien uudelleenkäytettävyyttä. Tämän perusteella ei ole yllätys, että sama rakenne on käytössä alakohdassa 2.3.2 esitetyissä ja lukuisissa muissakin reaaliaikaisen median ohjelmissa. Käyttöliittymän tasolla graafi on luonnollista esittää kuvien 2.1 ja 4.1 tapaan. Graafirakenteen varjopuolena on toteutuksen haastavuus: tehokas kanavia pitkin kommunikointi vaatii hyvää suunnittelua ja dynaamisesti muuttuva rakenne on virhealtis.

Laajennusmoduulien käytöstä jäi niin ikään positiivinen vaikutelma. Lähdekoodin tiukka kapselointi edistää sen uudelleenkäytettävyyttä ja vähentää järjestelmän osien välisiä potentiaalisia konflikteja kuten päällekkäisiä funktioiden ja tietotyyppien nimiä. Kaupallisen ohjelmiston kohdalla laajennuksista olisi se lisähyöty, että niitä voitaisiin levittää ilman lähdekoodia — Viskijukan tapauksessa tämä ei tietenkään ole relevanttia. Laajennuksien käytön ainoaksi merkittäväksi haitaksi osoittautui se, että kaikille saman, geneerisen rajapinnan kautta on työlästä välittää moduuleille juuri niiden tarpeisiin räätälöityä dataa. Jos visuaalit olisi toteutettu kiinteinä järjestelmän osina, niiden kanssa kommunikointi olisi ollut mutkattomampaa.

6.1.3 Visuaalien kiinnostavuus

Kiinnostavuuden arviointi tuotti jatkokysymyksen: miten kiinnostavuutta voi pitää yllä kohtuullisella työmäärällä? Esityksen säilyttäminen mielenkiintoisena edellyttää jatkuvaa uudistumista, mutta pitkän esityksen koostaminen täysin erillisistä lyhyistä osista merkitsee paitsi suurta työmäärää, myös merkittävää luovaa haastetta.

Uudistumista tukivat havaintojeni mukaan seuraavat ominaisuudet: visuaalien runsas parametrointi, visuaalien yhdistely ja grafiikan efektointi. Esimerkkivisuaaleista juurikin *Sphere* toimi yksinkertaisuudestaan huolimatta pisimpään, koska siinä oli eniten ajon aikana muunneltavia parametreja. Ulkoasun abstraktius korreloi jossain määrin parametroinnin kanssa: figuratiiviseksi tarkoitettu grafiikka ei tarjoa lähtökohtaisesti yhtä paljon vapautta ulkoasun suhteen kuin abstrakti. Tätä havaintoa ei ole syytä tulkita yksinkertaistaen, ikään kuin abstraktit visuaalit olisivat ylivoimaisia figuratiivisiin verrattuna. Figuratiivisella grafiikalla on mahdollista välittää merkityksiä ja tarjota yleisölle tarttumapintaa esim. viittausten kautta.

Visuaalien yhdistely ja jälkikäsittely luovat valtavasti uusia, potentiaalisesti kiinnostavia lopputuloksia. Tämä on osoitettavissa jopa laskennallisesti: jos käytössä on n kappaletta moduuleja, joita voidaan kytkeä päälle ja pois, niin yhdistelmiä on vähintään 2^n. Jo kuudella moduulilla yhdistelmien lukumäärä on 64, eikä lukemassa ole edes huomioitu näytettävien visuaalien uudelleenjärjestelyä. Kaikki yhdistelmät eivät välttämättä poikkea toisistaan merkittävästi tai ole esteettisesti käyttökelpoisia, mutta tästä huolimatta yhdistely on kiistämättä voimakas keino.

Kiinnostavuudesta saadut tulokset toistavat alakohdassa 3.5.1 käsiteltyjä Ulyaten ja Bianciardin (2002) kymmentä käskyä. Reagoinnin, modulaarisuuden ja yksinkertaisuuden totesin käytännössä itsekin hyviksi asioiksi. Aivan kaikille käskyille ei ole suoraa vastinetta jo siitäkin syystä, että yleisön osallistumista esitykseen ei ole kokeiltu.

6.2 Jatkokehitys

Ohjelmisto on luonteeltaan avoin ja sitä on tarpeen laajentaa, jotta eri tyyppisten käyttötilanteiden vaatimukset voidaan täyttää. Tämä monimuotoisuus aiheuttaa sen, että järjestelmä ei koskaan saavuta lopullista, muuttumatonta tilaa: jatkokehitys on aina mahdollista. Yhden henkilön voimin ei ole mitenkään mahdollista toteuttaa edes kaikkia tähän mennessä mieleen tulleita lisäominaisuuksia, joten priorisointi on tarpeen. Esittelen seuraavissa alakohdissa tärkeimmät jatkokehitysajatukset.

6.2.1 Ohjelmiston iteratiivinen kehittäminen

Tämänhetkinen rajapinta ei mahdollista riittävän joustavaa kommunikointia pääohjelman ja moduulien välillä. Kanavien kautta tapahtuva kommunikointi on tähän tarkoi-

tukseen työläs tapa. Myös kanavien ja viestien toteutusta täytyy kehittää, jotta kanavissa olisi mahdollista välittää nykyistä monipuolisempia tietotyyppejä. Eräs realistinen mahdollisuus on valmiin viestiprotokollan, kuten Max/MSP- ja Pure Data -ohjelmien laajennuksena toimivan *Open SoundControlin* (Wright ja Freed, 1997) toteuttaminen. Jo olemassa olevan protokollan kautta oman järjestelmän voisi yhdistää suoraviivaisesti muihin järjestelmiin. Näen tämän kehityksen iteratiivisena prosessina, jossa uusi toteutus johtaa samalla uusiin jatkokehitysajatuksiin.

6.2.2 Graafinen käyttöliittymä

Graafin koostaminen ohjelmoimalla ei ole intuitiivista eikä sitä voi tehdä kesken esityksen. Interaktiivinen käyttöliittymä mahdollistaisi kytkentöjen muuttelun reaaliajassa ja ennen kaikkea toisi ohjelmiston laajemman käyttäjäkunnan ulottuville. Optimaalinen ratkaisu olisi tukea kahta näyttöä siten, että toisella olisi näkyvissä käyttöliittymä ja toisella reaaliaikainen grafiikka. Käyttöliittymän lisääminen järjestelmään ei ole vielä realistisesti mahdollista: ohjelmarunkoon on tarpeen tehdä lisäyksiä, jotta ohjelman sisäinen logiikka voidaan esittää loppukäyttäjälle ymmärrettävässä muodossa. Esimerkkinä tästä kanavat: tällä hetkellä kanaviin viitataan joko niiden numeroilla tai muistiosoitteilla. Käyttäjälle ymmärrettävämpiä olisivat symboliset nimet kuten ”Mouse x coordinate” ja ”Sound Output”.

6.2.3 Uudet moduulit

Uusien visuaalien suunnittelu ja toteutus on keskeinen tehtävä, jotta ohjelmisto soveltuisi todelliseen käyttöön. Tällä hetkellä on olemassa lähinnä vain vanhojen, demoihin suunniteltujen efektien versioita. Kokonaan uusien visuaalien teossa on mahdollista huomioida projektin aikana saadut kokemukset ja täten tehdä niistä suoraan VJ-käyttöön sopivia. Kohina, grafiikan peilaus, kuvamosaiikki ym. yksinkertaiset jälkikäsittelyefektit ovat ensimmäisinä toteutettavien listalla.

Muiden moduulien osalta ensimmäinen fokus lienee äänianalyysissä: nykyisessä toteutuksessa en hyödyntänyt tätä mahdollisuutta vielä nimeksikään. Signaalinkäsittelyn menetelmin on mahdollista tunnistaa musiikista lukuisia piirteitä kuten iskut ja eri taajuuskaistojen aktiivisuus. Pelkän tunnistamisen lisäksi ohjaus vaatii myös pehmentävää suodatusta, sillä äkillinen ohjaus vaikuttaa luonnottomalta. Tähän tarkoitukseen on tarpeen käyttää ulkoisia kirjastoja ja mahdollisesti pyytää apua asiaa paremmin tuntevilta henkilöiltä, sillä oma äänenkäsittelyn tuntemukseni on vähäinen.

Uusille syöttölaitteille tarvitaan omat moduulit paremman kontrollin saavuttamiseksi. MIDI-pohjaisten laitteiden käsittely on loogista toteuttaa kerroksittain: ensimmäinen moduuli lukee järjestelmäriippuvaisella tavalla MIDI-dataa ja välittää sen eteenpäin

seuraavalle, joka puolestaan tulkitsee viestit ja muuttaa ne muun järjestelmän ymmärtämiksi viesteiksi. Jatkon kannalta mielenkiintoinen mahdollisuus olisi hajauttaa prosessointi usean tietokoneen välille, jolloin myös graafi kanavineen sijaitsisi osin eri laitteilla. Tästä lähestymistavasta olisi etua sekä prosessorikuorman keventämisen että kytkettävien laitteiden määrän kannalta. Luotettava hajautus ei kuitenkaan ole triviaali tehtävä ja hyvinkin toimivan lähiverkon latenssit voivat olla liian suuret etenkin musiikin ja kuvan synkronointiin.

Luku 7

Yhteenveto

Työtä tehdessäni hain vastauksia tutkimusongelmiin, mutta vielä lopuksikaan en onnistunut niitä kattavasti ratkomaan. Etenkin esityksen kiinnostavuus osoittautui valtavaksi kysymykseksi, johon löysin vain rajallisia keinoja, kuten visuaalien yhdistely ja parametrointi. Työn fokuksesta johtuen visuaalien esteettinen ulottuvuus ja tarinankerronta jäivät lähes tyystin arvioimatta, vaikka ne ovatkin kriittisiä tekijöitä kiinnostavuudessa. Yhtä oikeaa vastausta asettamiini kysymyksiin ei ole, joten saamani tulokset toimivat parhaiten kokonaisuuden rakennusosina, eivätkä niinkään valmiina vastauksina.

Aloitin ohjelmiston suunnittelun jo vuoden 2003 lopulla, mutta toteutukseen tartuin toden teolla vasta vuoden 2005 alussa. Osasyynä viivästymiseen olivat työkiireet, mutta vielä suuremmassa määrin psyykkiset tekijät. Ylisuunnittelun ja korkealle asetetun riman vuoksi kohtasin *tabula rasa* -ongelman: käytännön toteutukseen oli vaikea tarttua. Vuonna 2004 aloitin järjestelmän kirjoituksen kahteenkin kertaan, mutta itseluottamuksen ja motivaation puuttuminen aiheuttivat noiden versioiden nopean hylkäämisen. Itse ohjelmiston kirjoitus eteni suoraviivaisesti kunhan pääsin alkuun: teoreettisen suunnittelun aikana kohtaamani konseptuaaliset ongelmat muuttuivat konkreettisiksi ja samalla helpommin ratkaistaviksi. Tutuilla työkaluilla ohjelmointi ei vaatinut opettelua, joten saatoin kohdistaa energiani oleelliseen. Työkalujen hallinta lisäsi luottamusta siihen, että kohdatut haasteet ovat ylipäänsä voitettavissa.

Oman arvioni mukaan Viskijukan tekoprosessi oli varsin opettavainen. Eniten katson hyötyneeni ohjelmiston suunnittelusta ja toteutuksesta, jonka aikana sain uutta näkökulmaa reaaliaikaisen median käytännön ongelmiin. Oman työni, käymieni keskustelujen ja muihin järjestelmiin tutustumisen kautta sain tietoa erilaisista ohjelmistoteknisistä ratkaisuista, joilla näihin ongelmiin on tähän mennessä vastattu. Toteutuksen kankean alun jälkeen projekti muuttui mielekkääksi ja lopputuloksena syntyi toimiva järjestelmä, joka tarjoaa hyvän pohjan jatkokehitykseen. Toistaiseksi olen suunnitellut ohjelmistoa VJ-käyttöön, mutta avoin rakenne mahdollistaa tarvittaessa laajentamisen myös muiden samankaltaisten sovellusalueiden tarpeisiin.

VJ-kulttuuri oli työlle mielenkiintoinen konteksti. Näkemykseni VJ:n toiminnasta karttui projektin aikana, mutta opittavaa on edelleen runsaasti. Mielenkiintoisten visuaalien rakentaminen ja ylipäänsä esiintyjänä toimiminen vaativat lisää harjaantumista, jota voi saada vain käytännön kautta. Jatko on tältä osin vielä avoinna ja riippuu täysin saamastani palautteesta: uusien kehittäjien mukaan saaminen ja live-tilanteissa esiintyminen toisivat motivaatiota projektin edelleen jatkamiseen.

Lähteet

Ross Bencina, Phil Burk, 2001. Portaudio — an Open Source Cross Platform Audio API. In *Proceedings of The International Computer Music Conference 2001*, Havana, Cuba.

Antonio Camurri, Matteo Ricchetti, Riccardo Trocca, 1999. EyesWeb — Toward Gesture and Affect Recognition in Dance/Music Interactive Systems. In *Proceedings of The IEEE Multimedia Systems '99*, Firenze, Italy.

Annet Dekker, 2003. Synaesthetic Performance in the Club Scene. In *Proceedings of Cosign 2003: 3rd Conference on Computational Semiotics for Games and New Media*.University of Teesside, Middlesbrough, UK.

Matti Faler, 2001. Johdatus demosceneen. Teoksessa Tanja Sihvonen (toim.), *Sähköä, Säpinää, Wapinaa: Risteilyjä teknologian kulttuurihistoriassa*. Turun yliopiston historian laitoksen julkaisuja nr. 79.

Free Software Foundation, 1991. GNU General Public License, Version 2. Verkkojulkaisu, `http://www.fsf.org/licensing/licenses/gpl.html`, ladattu 5.3.2005.

Ilkka Haikala, Hannu-Matti Järvinen, 1994. *Käyttöjärjestelmät.* Modeemi Ry:n julkaisu C3.

Edvin de Koning, Bart van der Ploeg, 2005. Resolume VJ software. Verkkojulkaisu, `http://www.resolume.com/`, ladattu 5.3.2005.

Sam Lantinga, 2005. Simple DirectMedia Layer. Verkkojulkaisu, `http://www.libsdl.org/`, ladattu 5.3.2004.

Hanna Lönnblad, 1998. Tietokonedemot kulttuurina ja musiikin harrastuksen muotona. Pro gradu -tutkielma, Helsingin yliopisto.

Mika Meskanen, 2004. Nazca | Lume — The Orchestration of Live Visuals in a Synaesthetic Rock Performance. Taiteen maisterin lopputyö, Taideteollinen korkeakoulu.

David M. Nichols, Michael B. Twidale, 2003. The Usability of Open Source Software. First Monday, Volume 8, Number 1. Verkkojulkaisu, `http://www.firstmonday.org/issues/issue8_1/nichols/index.html`, ladattu 13.3.2005.

Pauli Ojala, 2004. Edo. Verkkojulkaisu, `http://www.anioni.com/edo/`, ladattu 20.3.2005.

Rick Parent, 2002. *Computer Animation*. Morgan Kaufmann Publishers.

Miller Puckette, 1988. The Patcher. In *Proceedings of The International Computer Music Conference 1988*, San Francisco, USA, pp. 420–429.

Miller Puckette, 1996. Pure Data: another integrated computer music environment. In *Proceedings of The Second Intercollege Computer Music Concerts*, Tachikawa, Japan, pp. 37–41.

Eric S. Raymond, 1999. *The Cathedral & the Bazaar*. O'Reilly & Associates Inc.

Markku Reunanen, 2002. Äänijärjestelmän ohjelmistoarkkitehtuuri. Diplomityö, Tampereen teknillinen korkeakoulu.

Petri Saarikoski, 2001. Valtavirtaa vastaan — Demoscene suomalaisen kotimikroilun historiassa. Lähikuva 3/2001, s. 54–65.

Dave Shreiner, Mason Woo, Jackie Neider, Tom Davis, 2003. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.4*. Addison-Wesley Publishing Company.

Ryan Ulyate, David Bianciardi, 2002. The Interactive Dance Club: Avoiding Chaos in a Multi-Participant Environment. Computer Music Journal, Volume 26, Issue 3, pp. 40–49.

Matthew Wright, Adrian Freed, 1997. Open SoundControl: A New Protocol for Communicating with Sound Synthesizers. In *Proceedings of The International Computer Music Conference 1997*, Thessaloniki, Greece.

Liite A

Tekninen sanasto

aihio	Eng. *template*. Olio, joka toimii pohjana uusien olioiden luomisessa.
GPL	General Public License. Yleisin avoimen lähdekoodin ohjelmien lisenssi.
graafi	Ohjelmistotieteessä: solmupisteistä ja niitä yhdistävistä poluista muodostuva verkkomainen rakenne.
Hz	Hertsi. Yksikkö, jolla ilmoitetaan tapahtumien määrä sekunnissa.
instantiointi	Uuden itsenäisen olion luominen alkuperäisestä aihiosta.
laajennus	Eng. *plugin*. Dynaamisesti ladattava ohjelman osa, joka laajentaa ohjelman toiminnallisuutta.
latenssi	Viive
Linux	Avoin Unixin kaltainen käyttöjärjestelmä.
MIDI	Musical Instrument Digital Interface. Standardoitu menetelmä, jolla liitetään digitaalisia soittimia toisiinsa sekä tietokoneisiin.
moduuli	Loogisesti itsenäinen ohjelmiston osa.
mikseri	Laite, jolla yhdistetään eli *miksataan* useita ääni- tai kuvalähteitä yhdeksi.
OpenGL	Silicon Graphics Inc:n vuonna 1992 julkaisema 3D-grafiikan ohjelmointirajapinta.
synkronointi	Tapahtumien tahdistaminen toisiinsa.
Unix	The Open Groupin tavaramerkki. Arkikielessä tarkoittaa joukkoa samankaltaisia käyttöjärjestelmiä, joissa on yhteinen ohjelmointirajapinta.

Liite B

Esimerkkilaajennus: dummy

```
/*
    An example plugin that does nothing special.
*/

#include <stdio.h>
#include <stdlib.h>
#include <vj.h>

int create(vj_plugin *p)
{
    /* Set some values not set by the core */
    p->outputs=1;
    p->inputs=1;
    p->name="Dummy plugin";
    printf("A new instance generated\n");

    return(VJ_OK);
}

void run(vj_plugin *p)
{
    vj_message  *put,*get;

    printf("I’m running here...\n");

    /* Read incoming integer messages if any */
    while(vj_message_pending(p->in_channel[0]))
    {
        get=vj_message_read(p->in_channel[0]);
```

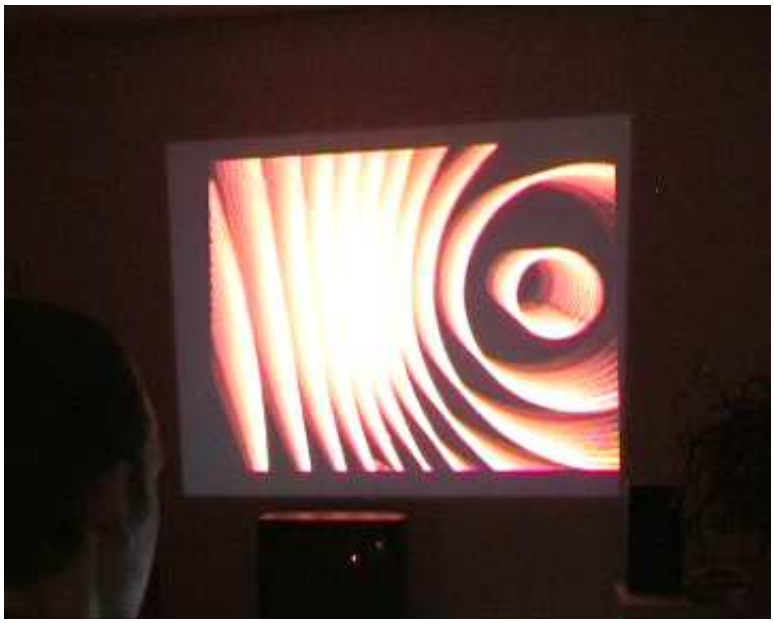
```
        if(get->types&VJ_INT)
            printf("Read: %d\n",get->i);
    }

    put=vj_message_get(VJ_INT);
    put->i=rand()%10;
    vj_message_write(p->out_channel[0],put);
    printf("Wrote: %d\n",put->i);
}

void delete(vj_plugin *p)
{
    /* Do additional cleanup if necessary */
    printf("Good bye\n");
}
```

Liite C

Valokuvia testitilanteesta



Kuva C.1: Sphere-visuaali

Kuva C.2: Ääniaalto ja Shine

Kuva C.3: Sphere, ääniaalto ja edellisen kuvan päälle piirto

Kuva C.4: Cubes ja Shine

Kuva C.5: VJ työssään