Käsittämättömät koodirivit musiikkina: bytebeat ja demoskenen tekninen kokeellisuus

Ville-Matias Heikkilä
Päätoimittaja
Skrolli-lehti
viznut@low.fi

Abstrakti

Bytebeat on tietokonemusiikin ja kokeellisen ohjelmoinnin muoto, jossa musiikkikappale toteutetaan muutamien kymmenien merkkien pituisena ohjelmointikielen lausekkeena. Artikkeli tarkastelee bytebeat-ilmiön vaiheita ja pyrkii luomaan sen kautta näkökulmaa siihen, kuinka tekninen kokeellisuus toimii demokulttuurissa.

Avainsanat: demoscene, bytebeat, tietokonemusiikki, algoritminen taide, ohjelmointi, hakkerikulttuuri, tilaoptimointi

Johdanto

Kaiken digitaalisen kulttuurin pohjalla ovat tietojenkäsittelyn alkeiselementit: bitit ja niillä operoivat konekäskyt. Valtavirran digikulttuurissa tämä taso hautautuu korkeampien abstraktioiden alle, mutta muutamissa marginaaleissa se on edelleen näkyvissä.

Demoskenestä löytyy monia sopukoita, jotka keskittyvät bittitason alkeiselementtien nyplääämiseen. Pikseligrafiikka sommitellaan yksittäisistä pikseleistä etenkin rajoittuneemmilla alustoilla. Äärimmäisimmät demoteokset puolestaan kootaan yksittäisistä konekäskyistä – etenkin kategorioissa, joissa tilaa tai prosessoritehoa on tarjolla hyvin niukasti. Tällaisia luovan toiminnan muotoja ei voi ymmärtää kunnolla, jos alkeiselementteihin pureutuva tekeminen sivuutetaan.

Tässä artikkelissa keskitytään bytebeatiksi nimettyyn tietokonemusiikin muotoon, joka sai alkunsa demoskenen piirissä syksyllä 2011. Voimakas innostusvaihe kesti vain noin kuukauden, mutta tänä aikana se ehti käydä läpi monia demoskenen teoskategorioille ominaisia kehitysvaiheita. Näin sitä voi käyttää eräänlaisena laboratorioesimerkkinä siitä, kuinka tekninen kokeellisuus ilmenee demoskenessä. Koska idea sai aikaan aktiivista toimintaa myös demoskenen ulkopuolella, voi

ilmiötä tutkimalla tuoda esiin demoskenen ja muun tietokonekulttuurin yhtäläisyyksiä ja eroavaisuuksia.

Artikkeli keskittyy ilmiön kulttuuristen kehitysvaiheiden läpikäyntiin, joten tekniikan esittelyssä keskitytään vain välttämättömään. Ilmiö esitellään yleisesti ja taustoitetaan sekä yleisemmän hakkerikulttuurin että demoskenen kautta. Tämän jälkeen sen vaiheet käydään läpi jotakuinkin kronologisesti etenkin aiheeseen liittyvää Pouet.net-sivuston keskusteluketjua seuraten. Tämän jälkeen kehityskulusta tehdään johtopäätöksia ja peilataan sitä toiseen demoskenen teoskategoriaan, VIC-20-demoihin.

Olen itse yksi Bytebeat-ilmiön alullepanijoista, ja olin aktiivisesti mukana tärkeimmässä kokeiluvaiheessa. Kirjoitin aiheesta vuonna 2011 myös kaksi tekniikkaan ja teoriaan keskittyvää blogikirjoitusta (Heikkilä 2011a, 2011b) ja akateemisemmin muotoillun artikkelin (Heikkilä 2011c). Tämä on tietääkseni ensimmäinen kirjoitus, joka keskittyy ilmiön kulttuurillis-sosiaaliseen puoleen.

Tekninen esittely

Bytebeat sai alkunsa halusta tutkia, kuinka vähillä laskentaoperaatioilla on mahdollista tuottaa musiikilta kuulostavaa ääntä. Ensimmäiset kokeilut olivat C-kielisiä ohjelmia, jotka noudattavat seuraavaa muotoa:

```
main(t){for(;;t++)putchar(LAUSEKE);}
```

Tällainen ohjelma on ikuinen silmukka, joka laskee siihen sisältyvää lauseketta peräkkäisillä muuttujan t arvoilla alkaen yleensä arvosta 1.[1]

Lausekkeen voi ajatella matemaattisena kaavana, joka kuvaa äänisignaalin ajan funktiona. Funktion arvot tulostetaan merkkeinä standardiulostuloon. Tämä tuloste on uudelleenohjattava käsittelijälle, joka vuorostaan tulkitsee merkit PCM-muotoiseksi äänidataksi. Alkuvaiheessa käytettiin käsittelijänä Linuxin laitetiedostoja /dev/audio ja /dev/dsp, jotka käyttävät oletusarvoisesti tarkkuudeltaan 8-bittisiä näytteitä 8 000 hertsin näytteenottotaajuudella.

[1] Ohjelmarunko ei ole millään tavoin korrekti oppikirjaesimerkki C-ohjelmasta vaan edustaa merkkimäärältään minimoitua koodia. Muuttuja t on minimointisyistä määritelty main()-funktion argumenttina, jolloin sen tyyppiä ei tarvitse määritellä, vaan se oletetaan suoraan peruskokonaisluvuksi (int, yleensä 32-bittinen). Ensimmäinen main()-funktion argumentti on ohjelman käynnistykseen käytetyn komentorivin argumenttien määrä, joka on normaalitapauksessa 1 (eli komentorivillä on pelkästään ohjelman nimi). Näin ollen muuttujan t alkuarvoksi tulee 1.

Mielenkiintoisia bytebeat-lausekkeita on koottu kolmeen Youtube-videoon, jotka ovat ehkä helpoin tapa saada yleiskäsitys bytebeat-musiikista (Youtube 2011a, 2011b, 2011c). Bytebeatilla on esteettisiä yhtymäkohtia chip-musiikkiin (ks. Marilou Polymeropouloun artikkeli tässä erikoisnumerossa), suositaanhan molemmissa esimerkiksi kantti- ja sahalaita-aaltoja, jotka ovat toteutettavissa hyvin yksinkertaisilla digitaalilogiikan rakenteilla.

Yksinkertaisin jotain ääntä tuottava lauseke on *t*, jota käyttävä ohjelma tulostaa peräkkäiset t:n arvot suoraan: 1, 2, 3, … Sarja ei kuitenkaan kasva loputtomiin, sillä putchar() käyttää saamastaan arvosta yleensä vain alimmat kahdeksan bittiä (eli 256:lla jakamisen jakojäännöksen). Tähän 8 bitin osuuteen eli tavuun viittaa myös bytebeat-sanan etuliite byte. Merkin 255 jälkeen tulostuu siis merkki 0 ja sarja alkaa alusta, joten ohjelma tuottaa tasaista sahalaita-aaltoa, jonka aallonpituus on 256 näytettä. Jos lauseke on esimerkiksi *t**2*, lyhenee aallonpituus 128 näytteeseen, eli sävelkorkeus nousee oktaavilla.

Lausekkeet rakennetaan C-kielen alkeisoperaatioista sekä lukuvakioista ja merkkijonoviittauksista. Laskujärjestystä voi muuttaa sulkumerkeillä. Suurinta osaa kielen ohjaus- ja tietorakenteista ei kuitenkaan voi käyttää – tämä on ehkä merkittävin bytebeatin rajoite.

Toinen tärkeä muotoa-antava bytebeatin piirre rakenteellisen rajoitteen ohella on bittitason operaatioiden runsas käyttö, joka on hyvin epätyypillistä perinteisessä ääniohjelmoinnissa (vrt. esim. Tolonen ym. 1998). Bytebeatin teknistä teoriaa rakentaessani en onnistunut löytämään ainuttakaan aiempaa esimerkkiä bytebeatille ominaisesta tavasta moduloida aaltomuotoja bittioperaatioilla. SuperCollider-musiikkiohjelmointikielikin lisäsi valikoimaansa bittioperaatiot vasta maaliskuussa 2012, uutissivunsa (SuperCollider 2012) koodiesimerkistä päätellen juuri bytebeatin innoittamana.

Bytebeatissa käyttökelpoisia alkeisoperaatioita ovat:

- Tavallinen kouluaritmetiikka eli yhteen-, vähennys-, kerto- ja jakolasku sekä jakojäännös:
  + – * / %
- Bitittäiset perusoperaatiot AND, OR, XOR ja NOT sekä bittisiirrot oikealle ja vasemmalle:
  & | ^ ~ << >>
- Ehdollisen suorituksen mahdollistava ternäärioperaattori:
  ? :
- Vertailuoperaattorit:
  < <= > >= == !=

Bytebeat-lausekkeita kokeiltiin alkuvaiheessa hyvinkin sattumanvaraisesti, ja teoria niiden toimintaperiaatteista kehittyi vasta jälkeenpäin. Tästäkään syystä ei ole kovin mielekästä käsitellä tekniikkaa tämän syvemmin. Tekniikkaan paneudutaan tarkemmin vuonna 2011 kirjoittamissani artikkeleissa (Heikkilä 2011b, 2011c).

Bytebeat laajeni melko pian alkuperäisestä C-kontekstistaan myös muihin kieliin, kun nettiin ilmestyi lausekkeiden kokeilemiseen tarkoitettu sivu. Toteutuksessa käytetyn JavaScriptin lausekesyntaksi ja laskentalogiikka eroavat jonkin verran C:stä, ja melko pian alkoikin ilmestyä lausekkeita, jotka toimivat oikein vain JavaScriptillä. Bytebeat-idea on poikinut myös sovelluksia, jotka käyttävät C-tyylisten lausekkeiden sijaan Forth-tyylistä pinosyntaksia. Tässä artikkelissa bytebeat rajataan kuitenkin käsittämään vain C-tyyliset lausekkeet, jotka tuottavat ääntä joko Javascript-sivulla ajettuna tai edellä annetussa C-rungossa.

Seuraavassa vielä näytteen vuoksi muutama erilainen bytebeat-lauseke, joihin ei tässä paneuduta sen syvemmin. Neljä ensimmäistä toimii sekä C-kielisenä että Javascriptinä, viimeinen vain Javascriptinä.

- t&t>>8
- (t*5&t>>7)|(t*3&t>>10)
- (t&t%255)-(t*3&t>>13&t>>6)
- (t/8)>>(t>>9)*t/((t>>14&3)+4)
- (3e3/(y=t&16383)&1)*35 +(x=t*”6689″[t>>16&3]/24&127)*y/4e4 +((t>>8^t>>10|t>>14|x)&63)

Lyhyen ohjelmakoodin magia

Jo ennen mikroprosessoriaikaa laadittiin hyvin lyhyitä tietokoneohjelmia, *hackeja*, jotka tekevät kokoonsa nähden hämmästyttäviä asioita. Muutamia näistä on kirjattu *HAKMEMiin* eli MIT:n tekoälylaboratorion muistioon 239 (Beeler ym. 1972). Joukossa on mm. grafiikkaohjelmia eli *display hackeja* ja kaksi ääniohjelmaa (mt, kohdat 145 ja 168). Ääniohjelmat tuottavat melko lyhyitä, toistuvia äänisarjoja, jotka muuttuvat koneen ohjauspaneelin bittikytkinten mukaan. Niiden sovittaminen C-kielelle paljastaa, etteivät ne muodosta muita kun mikrotason musiikillisia rakenteita, mutta teknisesti niitä voisi silti pitää bytebeatin varhaisena esimuotona.

Hakkeriperinnettä dokumentoiva *Jargon File* (Raymond 1996, 2004) selittää *hackeihin* kohdistuvaa kiinnostusta käsitteellä *hack value*. Hakkerit arvostavat ohjelmissa *hack valueta* ja pyrkivät

synnyttämään sitä omiin tuotoksiinsa. Vaikka käsitteen sanotaankin olevan enemmän kokemusperäinen kuin tiukasti määriteltävä, *display hackien* tapauksessa siihen vaikuttavat tekijät annetaan hyvinkin tarkasti:

> The hack value of a display hack is proportional to the esthetic value of the images times the cleverness of the algorithm divided by the size of the code (Raymond 1996, 154).

Korkein *hack value* on siis ohjelmilla, jotka nerokkaan algoritmiikkansa avulla tuottavat eniten kauneutta lyhimmällä koodilla. Tähän ideaaliin sopivat demoskenen piirissä etenkin tiukkaan kokorajaan ahdetut teokset. Myös bytebeat-ohjelmissa pyritään samantapaiseen ihanteeseen: niissäkin tärkeitä tekijöitä ovat ohjelman tiiviys, sen tuottaman äänen estetiikka ja uusien teknisten temppujen löytäminen.

Demoskenessä ei juuri käytetä *hack value* -käsitettä – sanaparin haku Pouet.netistä tuottaa tätä kirjoitettaessa vain kaksi osumaa. Alakulttuurin pyrkimys aina vain näyttävämpiin ja kekseliäämpiin teoksiin aina vain pienemmissä kokokategorioissa osoittaa kuitenkin samankaltaisen ihanteen olemassaolon. Lisäksi *display hackit* on tunnustettu demojen esimuodoksi niin demoskenen kuin alkuperäisen hakkerikulttuurinkin piirissä.[2]

Hackit ovat usein vaikeaselkoisia – kokeneenkin ohjelmoijan voi olla vaikea ymmärtää niiden toimintaperiaatteita. Tämä voi kuitenkin olla jopa toivottava piirre *hack valuen* kannalta. Käsite *magic* viittaa *Jargon Filessä* vaikeastiselitettävyyteen – äärimmäisessä tapauksessa jopa siihen, ettei ohjelman toimintaa ymmärrä kukaan (*black magic*) (Raymond 1996, 72). Samalla *magic* kuitenkin määritellään kaiken insinöörityön ja kehityksen korkeimmaksi päämääräksi viitaten Arthur C. Clarken kolmanteen lakiin (Clarke 1973, 21): ”Any sufficiently advanced technology is indistinguishable from magic.” Käsittämätön koodi kutkuttaa hakkerien mielikuvitusta ja antaa vaikutelman tekniikan äärirajojen puskemisesta. Vuodesta 1984 järjestetty *International Obfuscated C Code Contest* kannustaa vaikeaselkoisen ohjelmakoodin kirjoittamiseen itsetarkoituksellisesti.

Bytebeateihin on alusta asti viitattu yhden rivin ohjelmina, onelinereina. Näiden ohjelmien kulttuurihistoriaa valottaa Nick Montfortin ja tämän kollegoiden kirja *10 PRINT*

[2] Nykyisen *Jargon Filen* (Raymond 2004) ja *Wikipedian* määritelmät termille *display hack* mainitsevat perinteen jatkajaksi demoskenen. Samoin ohjelmalliseen taiteeseen keskittyvän displayhacks.org-sivuston takana on tunnettu demontekijä Bent Stamnes (Gloom/Fairlight), joka lainaa sivuston esittelyssä *Jargon Filen* määritelmää.

CHR$(205.5+RND(1)); : GOTO 10, jonka nimi on itsessään Commodore 64:llä toimiva yhden rivin grafiikkaohjelma. Hyvin lyhyet ohjelmat kiehtoivat jo 1960-luvulla APL-ohjelmointikielen ohjelmoijia, ja myöhemmin suosittuja toteutuskieliä ovat olleet mm. BASIC ja Perl. Yhden rivin ohjelmat on myös nähty taianomaisina: esimerkiksi Commodoren koneisiin erikoistunut *RUN*-lehti julkaisi onelinerinsä *Magic*-nimisellä palstalla. (Montfort ym. 2012, 148-152.)

Hyvin lyhyitä musiikkiohjelmia on julkaistu ennen bytebeat-ilmiötä esimerkiksi *The Wire* -musiikkilehden *SuperCollider 140* -kokoelmassa vuonna 2009. Kukin kokoelman 22 kappaleesta tuotetaan Twitter-mittaisella eli enintään 140-merkkisellä SuperCollider-ohjelmalla. SuperCollider on erityisesti musiikkiohjelmointiin tarkoitettu kieli, jossa on runsaasti nimenomaan äänen ja musiikin käsittelyyn tarkoitettuja ominaisuuksia. Kokoelman ohjelmat käyttävät näitä ominaisuuksia laajasti hyväkseen, joten ne ovat bytebeatmaisesta kokoluokastaan huolimatta tekniseltä perustaltaan varsin erilaisia.

Miniatyyrimusiikkiohjelmointi demoskenessä

Demoskenessä bytebeat kytkeytyy etenkin pienimpiin teoksiin, joita Markku Reunanen käsittelee artikkelissaan *Neljän kilotavun taide*. Neljän kilotavun introt olivat alkuvuosinaan yleensä äänettömiä, mutta musiikki yleistyi niissä vuosituhannen vaihteessa. Nykyisten PC-alustojen 4k-introjen musiikki tuotetaan yleensä teoksiin integroiduilla ohjelmallisilla syntetisaattoreilla eli niin kutsutuilla *softasynilla*. (Reunanen 2013.)

Reunanen tuo esiin demoskenessä vallitsevan paineen siirtyä aina vain tiukempiin kokoluokkiin: esimerkiksi 4k-sarjan vaativuuden kasvaessa ovat yhden kilotavun teokset nostaneet päätään niiden rinnalla. Tämän kehityksen myötä aina vain pienempiin teoksiin halutaan myös musiikkia, ja musiikin toteuttamiseen on löydettävä uudenlaisia menetelmiä. Neljän kilotavun kokoluokassa käytetään yleensä melko perinteiseen tapaan rakennettuja soittorutiineja, joissa esimerkiksi synteesi, ohjauslogiikka ja nuottidata ovat selkeästi erillisiä moduuleja. Reunasen esittelemät *4klang* ja *Syna* on toteutettu näin. Perinteinen rakenne muuttuu kuitenkin haastavaksi kokoluokkien pienentyessä – tilaa kovin moniosaiselle kokonaisuudelle ei yksinkertaisesti enää ole.

Bytebeat haastaa vallitsevat käsitykset siitä, kuinka soittorutiini on rakennettava: jo muutaman alkeisoperaation yhdistelmä pystyy hoitamaan kaikki tarvittavat osa-alueet ainakin hyvin karkealla tavalla. Tämä tekee siitä demoskenen kannalta mielenkiintoisen paitsi itsenäisenä kategorianaan, myös uusien tekniikoiden etsimisvälineenä.

Umpimähkäiset ensikokeilut

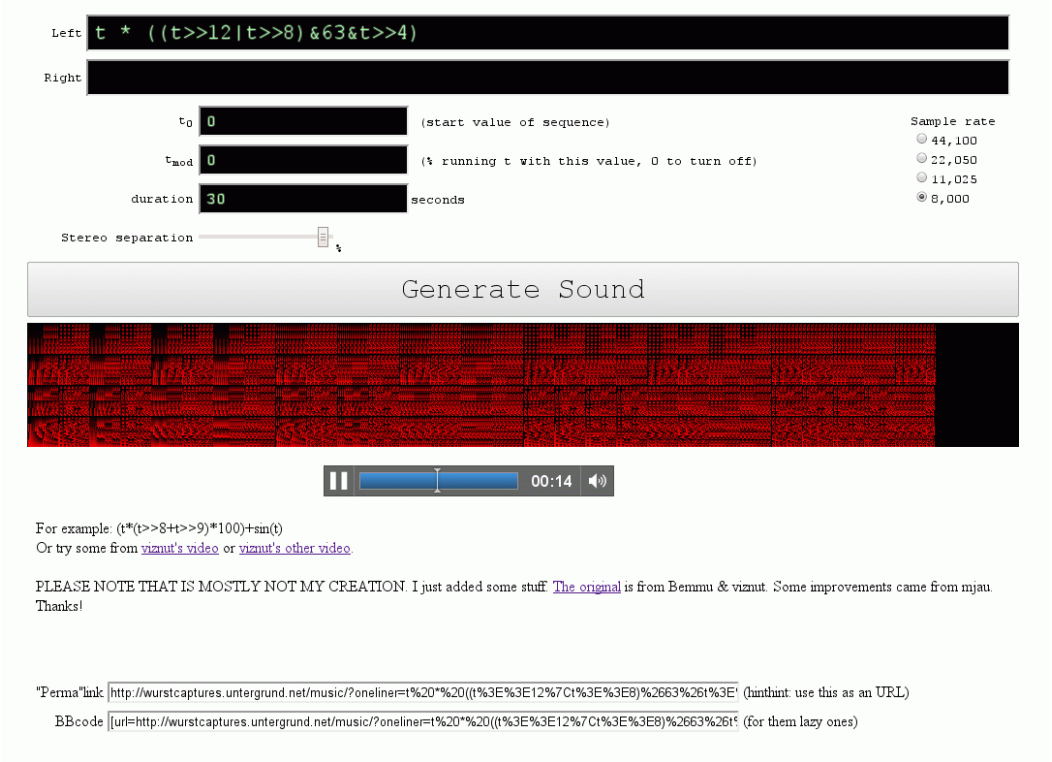
Bytebeat sai alkunsa PWP-demoryhmän IRC-kanavalla 18. päivä syyskuuta 2011, kun keskusteluun osallistuneet kokeilivat yksinkertaisten C-ohjelmien tuottamia ääniä. Alkuperäisenä innoittajana kokeilulle toimi toukokuussa 2011 julkaistu 23 tavun mittainen Commodore 64 -teos, Ate bit -demoryhmän *4mat*-nimimerkin *Wallflower*, joka tuottaa yllättävän monimuotoisen audiovisuaalisen rakenteen muutamalla peruskonekäskyllä. (Heikkilä 2011a.)

Kokeilun tuloksena syntyneet ohjelmat lähtivät leviämään eri verkkopalveluihin (Twitter, IRC, Google Plus), jossa ne herättivät kiinnostusta. Ohjelmista koostettu video *Experimental music from very short C programs* ladattiin Youtubeen 26. päivänä (Youtube 2011a). Seuraavana päivänä Pouet.net-demoskenesivustolle ilmestyi uusi keskusteluketju, jonka otsikko oli sama kuin videolla (Pouet 2011).

Keskusteluketjun alussa keskitytään etsimään videon esittämille lyhyille musiikkiohjelmille demoskenerelevanssia. Ketjun aloittanut nimimerkki *elfan* ehdottaa näille omaa musiikkiohjelmakilpailuaan (*executable music compo*), jossa esimerkiksi lähdekoodin merkkimäärää olisi rajoitettu. Vaihtoehtoisena ideana esitetään perinteisempää, ajettavan tiedoston 256 tavun kokorajaan perustuvaa kilpailua, jollainen on järjestetty aiemmin ainakin Commodore 64:lle.

Ensimmäisenä päivänä vain muutama kokeilee tekniikkaa: ketjuun ilmestyy yksi uusi C-kielinen ohjelma, ja eräs videon ohjelmista ehditään sovittaa Atari 2600 -pelikonsolille. Monille kynnys ajaa ja muunnella ohjelmia itse on turhan korkea, sillä esimerkiksi Windows-käyttöjärjestelmä vaatii erilaista lähestymistapaa. Pioneerivaihe jää kuitenkin hyvin lyhyeksi, sillä jo 28.9. ketjussa julkaistaan linkki sivulle, joka mahdollistaa lausekkeiden kokeilun www-selaimessa.

Tämän käännekohdan jälkeen huomattava osa ketjun viesteistä on bytebeat-lausekkeita ja kommentteja niihin. Monet selvästikin kirjoittavat lausekkeita suhteellisen umpimähkäisesti – muutamat jopa toteavat, etteivät ymmärrä lainkaan mitä tekevät. Tätä käsitystä vahvistavat monissa lausekkeissa esiintyvät tekniset järjettömyydet kuten nollan tai yli sadan pituiset bittisiirrot. Myös lausekkeiden kokeiluavaruus laajenee: ensimmäisessä videossa esiintyvät lausekkeet lähinnä varioivat samaa perusrunkoa, mutta uudet ovat muodoiltaan monipuolisempia.



Kuva 1. Selainpohjainen testaustyökalu. Lähde: http://wurstcaptures.untergrund.net/ music/

Tekniikan kesyttäminen

Syyskuun 29. päivänä ruvetaan ketjussa ihmettelemään, kuinka bytebeat-lausekkeet oikeastaan toimivat. Bittioperaatioita vähemmän käyttänyt ohjelmoija haluaa tutustua niihin paremmin. Eräs keskustelija toteaa, että vaikka hän ymmärtääkin bittioperaatiot, niin hän ei käsitä niiden roolia lausekkeissa. Lausekkeiden määrätietoisempaan rakentamiseen annetaan myös vinkkejä: *xpansive* kertoo, että XOR- tai AND-operaatiota kannattaa käyttää useamman lausekkeen yhdistämiseen.

Samana päivänä ketjuun ilmestyy myös lausekkeita, jotka eivät edes yritä olla C-kielisiä vaan kokeilevat rohkeasti www-testaussivun käyttämän JavaScriptin mahdollisuuksia. Viimeistään tässä vaiheessa testaussivusta on tullut konkreettinen pääalusta, eikä C-yhteensopivuudesta enää välitetä. Liukulukujen ja trigonometrian käyttö ei kuitenkaan miellytä joitakin keskustelijoita, jotka haluaisivat rajata ne pois kokeilun piiristä: ”sin/cos/floats are no valid micro-music functions.”

Bytebeat-ilmiö on siirtynyt ensikokeiluista uuteen vaiheeseen, jossa tekniikkaa yritetään oppia ymmärtämään, jotta sitä voitaisiin hyväksikäyttää määrätietoisemmin. Kutsun tätä vaihetta tekniikan kesyttämisvaiheeksi. Kaavat eivät ole enää niin umpimähkäisiä kuin aiemmin, vaan usein hyvinkin tarkoitushakuisesti rakennettuja. Ketjuun alkaa ilmestyä myös kommentteja, joissa asetetaan kehitykselle konkreettisia tavoitteita ja visioidaan sen mahdollisuuksia:

> OMG, that's sooo cool :D. I've been waiting for someone to create one that actually features harmonic progression of some kind. Now we've got cool basslines, percussion, melodies and harmonic progression. Won't be long now until we have music in multiple parts that features all the elements of a "normal" song :). (*elfan*, 1.10.2011.)

Lokakuun toisena päivänä julkaisen ilmiötä ja sen taustoja ja tekniikkaa käsittelevän blogikirjoituksen (Heikkilä 2011a.) Linkki kirjoitukseen leviää Reddit- ja Hacker News -uutissivustoille. Siinä missä demoskeneyhteisö innostui pelkästä koodia ja ääntä esittävästä videosta, laajempi hakkeriyhteisö nähtävästi tarvitsi perusteellisemman alustuksen ennen läpimurtoa. Heti seuraavana päivänä hackaday.com-blogin aiheena on AVR-mikrokontrollerilla toimiva bytebeat-ohjelma. Bytebeat on alkanut selvästi elää omaa elämäänsä myös demoskenen ulkopuolella.

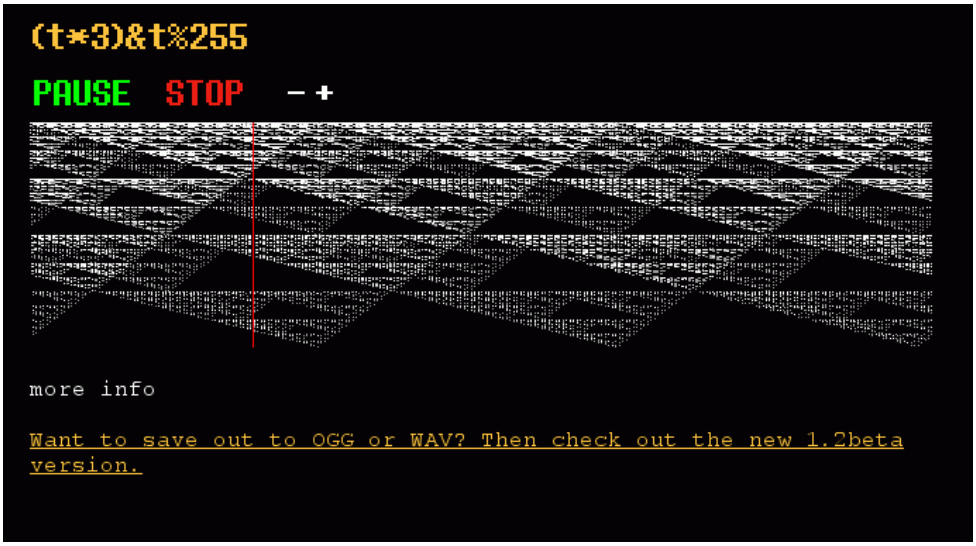
Lokakuun neljäntenä alkaa ilmestyä tarkoitushakuisemmin rakennettuja bytebeat-lausekkeita, joihin on tallennettu lyhyitä nuottisarjoja luku- ja merkkijonovakioina. Eräs kokeilija pohtii, rikkooko tämä koko kokeilun henkeä, mutta on kuitenkin sitä mieltä, että tietoisen kontrollin määrää on hyvä lisätä:

> i'm not sure if the shift-register-as-step-sequencer violates the spirit of the whole thing, but making it a bit more controllable seems like a good move overall. (*ryg*, 4.10.2011.)

Nimimerkki *mu6k* julkaisee parinsadan merkin pituisen lausekkeen, johon kuuluu harmoninen progressio bassolinjoineen, melodia ja rumputausta. Muut keskustelijat optimoivat pian lausekkeen pituuden alle 100 merkkiin. Analyysini perusteella tämä lauseke yhdistää sattumanvaraisella kokeilulla löytyneitä ja tietoisesti rakennettuja elementtejä. (Heikkilä 2011b.)

Kahden päivän päästä *mu6k* julkaisee yli 600 merkkiä pitkän lausekkeen, joka sisältää osittaisen sovituksen Conspiracy-ryhmän *Chaos Theory* -teoksen musiikista. Kyseinen 64k-intro on julkaisunsa jälkeen uudelleentoteutettu kaksikin kertaa 4k-kokoluokassa. Se oli ehkä tämän historiansa vuoksi houkutteleva lähdemateriaaliksi.

Osa keskustelijoista alkaa spontaanisti optimoida *Chaos Theory* -sovitusta merkkimäärältään lyhyemmäksi. Koodin tilaoptimointi on yleistä demoskenellä, mutta se on hyvin harvoin näin yhteisöllistä ja julkista. Seuraavana päivänä lausekkeen pituus on saatu 300 merkin pintaan laadun juuri kärsimättä.



Kuva 2. Flash-pohjainen vaihtoehtoinen testaustyökalu. Lähde: http://entropedia.co.uk/generative_music/

Innostuspiikin jälkeen

Chaos Theory -optimoinnin jälkeen keskustelutahti hiljenee vähitellen. Mahdollisesti rima nousi sen myötä jo niin korkealle, että sen ylittämisen tai ylipäätään uusien temppujen löytämisen otaksutaan vaativan jo pitkäjänteisempää kehitystyötä. Ketjuun ilmestyy edelleen uusia lausekkeita, mutta niissä on harvoin kovin hätkäyttäviä niksejä verrattuna aiempiin tuotoksiin.

Ajatus träkkerityylisestä bytebeat-musisointityökalusta nousee esiin. Samoin pinnalle palaa ensimmäisten viestien idea musiikkikilpailun järjestämisestä demopartyilla. Tällaista kilpailua ei kuitenkaan tietääkseni ole tähänkään mennessä järjestetty missään tapahtumassa, ja tiedossani ei myöskään ole kuin jokunen pikkuintro, jossa tekniikkaa on käytetty tunnistettavasti. Yksi näistä on oma 125-tavuinen MS-DOS-teokseni *Express Train 125*, jonka tarkoitus on lähinnä osoittaa, että bytebeat-tekniikkaa pystyy käyttämään 256 tavun kokoluokassa.

Suurin osa kiinnostavasta bytebeatiin liittyvästä kehityksestä on siirtynyt tässä vaiheessa demoskenekontekstin ulkopuolelle. Tekniikka saa nimen *bytebeat* 8. päivänä joulukuuta, ja Kragen Javier Sitaker kirjoittaa pian sen jälkeen yhteenvedon ilmiön senhetkisistä rönsyistä: monet ovat käyttäneet bytebeatia elektroniikkarakenteluprojekteissaan, ja Applen mobiililaitteille on julkaistu joulukuussa bytebeat-tekniikkaa käyttävä *Glitch Machine* -sovellus (Sitaker 2011). Myös useita muita tekniikkaa käyttäviä musiikkiohjelmia ilmestyy myöhemmin mobiili- ja työpöytäkäyttöjärjestelmille (*Bitwiz Audio Synth*, *Pytebeat*, *Droidbeat Synth*). Bytebeatia on

käytetty pohjana myös musiikkiteknologian tutkimuksessa, jossa uusia lausekkeita tuotetaan automaattisesti geneettisten algoritmien avulla (Kaliakatsos-Papakostas 2012).

Pouet-keskustelu hiljenee joulukuun 9. päivän jälkeen ja herää seuraavan kerran henkiin vasta kahden kuukauden päästä, jolloin *mu6k* esittelee uuden kokeilunsa. Pian tämän jälkeen keskustelu hiljenee taas. Koska ketjuun lisätyt lausekkeet saavat kuitenkin innostuneita kommentteja myös muiden uudelleenheräämisten yhteydessä, näyttäisi kiinnostus tekniikkaa ja sen kehitystä kohtaan olevan tallella. Sitä seurataan kuitenkin mieluiten sivusta, ja näyttämölle jää vain muutama tekniikalle pitkäjänteisemmin omistautunut tekijä.

Ilmiön luonne ja vaiheet

Bytebeat poikkeaa monin tavoin tyypillisestä demoskenen teoskategorian kehityskaaresta. Tavanomaisiin kategorioihin julkaistaan teoksia lähinnä kilpailujen yhteydessä, ja niiden kehitystyö tapahtuu poissa julkisuudesta. Bytebeat sen sijaan ehti käydä huippuhetkensä läpi ennen kuin ainuttakaan kilpailua ehdittiin järjestää, ja kehityksen kulku on jopa yksityiskohtiin asti seurattavissa julkisista viesteistä. Yhteisönlaajuisen viestinnän tiheys varmastikin myös nopeutti kategoriaan liittyvän tekniikan kehitystä verrattuna siihen, että yksilöt tai ryhmät olisivat kehittäneet tekniikkaa vain omilla tahoillaan ja vaihtaneet tietoja vain partyjulkaisujen yhteydessä.

Poikkeuksellista bytebeatin kehityskaaressa on myös runsas aiempien teosten käyttäminen uusien pohjana. Demoskenessä on perinteisesti katsottu pahalla toisten teosten hyödyntämistä, ja sama ilmiö näkyy myös sille läheisessä träkkerikulttuurissa (Lysloff 2003.) Pouet-ketjusta ei kuitenkaan löytynyt ainuttakaan viestiä, jossa joku olisi suuttunut oman materiaalinsa uusiokäytöstä. Osasyynä on varmaankin se, että etenkin alkuvaiheessa uusien teosten tuottaminen perustui suhteellisen vähän vaivaa vaativiin kokeiluihin, eikä niitä mielletty samalla tavoin omiksi hengentuotteiksi kuin pitempää rakentelua vaativat teokset mielletään.

Kilpailullisuus on demoskenessä keskeisessä osassa, ja se ilmenee yleensä muodollisten kilpailujen ja paremmuusäänestysten kautta (Reunanen 2010). Kilpailullisuus näkyy selvänä motivaattorina myös bytebeat-keskusteluketjussa, mutta se ilmenee vapaamuotoisemmin: ketjuun osallistuneet pyrkivät ”voittamaan” aiemmin julkaistut lausekkeet joko varioimalla niistä parempia, optimoimalla niitä lyhyemmiksi tai keksimällä lähestymistavoiltaan täysin uudenlaisia lausekkeita. Hyvät saavutukset saavat tunnustusta positiivisten kommenttien muodossa.

Bytebeatia käsittelevissä Hacker News- ja Reddit-keskusteluketjuissa ei tämäntyyppistä kilpailullista tekniikan eteenpäinvientiä juuri näy, vaan suurin osa niissä näkyvistä kaavoista on pelkkiä ensimmäisiä kokeiluja: ”Katsokaa, minäkin sain aikaan edes jotain!” Pouet-ketjussa nähty kilpailullinen asenne näyttäisi siis erottavan sen havaittavasti laajemmasta tietokonekulttuurista.

Bytebeat-kisailua voisi luonnehtia abstraktimmin pyrkimyksenä maksimoida *hack valueta*. Etenkin *Jargon Filen display hack* -määritelmän kolme muuttujaa vaikuttavat tässä erityisen toimivilta (Raymond 1996, 154). Algoritminen nerokkuus näkyy uudentyyppisten lausekkeiden esiin tuomisena mutta myös tavoissa yhdistellä ja optimoida lausekkeita. Esteettistä kehitystä saadaan aikaan sekä kokeellisen varioinnin että algoritmisen innovaation kautta, ja etenkin tietoisemmin rakennetut lausekkeet pyrkivät ylittämään aiemmat lausekkeet esteettisessä mielessä. Koodin lyhyyden ihanne puolestaan näkyy sekä aiempien lausekkeiden optimointikilvoittelussa että poikkeuksellisten lyhyiden lausekkeiden osakseen saamassa ihailussa.

Oma kilpailullinen alueensa liittyy selkeästi mitattaviin ennätyksiin ja niiden rikkomiseen, joka bytebeatissa näkyy etenkin pitkien lausekkeiden merkkimäärän optimointikilpailuna. Tämä rinnastuu suoraan esimerkiksi vanhoissa Amiga-demoissa nähtävään pyörivien 3D-kappaleiden kolmiomäärillä kilpailemiseen. Koodinpituuskilpailuista esimerkkinä mainittakoon *Hugi*-levykelehden vuosina 1998-2009 järjestämät *Size Coding Compot*, joissa tavoitteena oli useimmiten tuottaa tehtävänannon mukainen ohjelma mahdollisimman pienenä MS-DOS-ohjelmana.

Esteettisenä ihanteena bytebeateille näyttäisivät demoskenen piirissä olevan ”tavalliset” musiikkikappaleet, jotka noudattavat yleisempää demoskene-estetiikkaa tasaisine rytmeineen, harmoniakulkuineen ja synteesiteknisine piirteineen. Tämä ihanne näkyy etenkin *mu6k*:n tuottamissa pitemmissä lausekkeissa ja samaistuu esimerkiksi 64k- ja 4k-introjen musiikilliseen kehitykseen: mitä ”isommalta” kuulostavan kappaleen saa rajalliseen tilaan, sitä parempi. Demoskenen ulkopuolisessa bytebeat-kehityksessä näyttäisi keskeisempi rooli sen sijaan olevan satunnaiskokeiluille suosiollisemmalla glitch-estetiikalla, mikä näkyy esimerkiksi bytebeat-ohjelmistojen nimissä kuten *Glitch Machine* tai *libglitch*.

Sekä demoskenellä bytebeatiin sovellettu maksimalismi että sen ulkopuolella sovellettu glitch-estetiikka edustavat hakkeriestetiikkaa, jossa tietoteknisen alustan piirteet vaikuttavat teosten syntyyn hyvin voimakkaasti. Vaikka bytebeat häivyttääkin konkreettisen laitealustan piirteet, voidaan sitä pitää omana alustanaan alustatutkimuksen mielessä (Bogost & Montfort 2007). Chip-muusikoiden menetelmiä ja motivaatioita tutkinut Carlsson (2010) kutsuu kahta edellämainittua

hakkeriestetiikan lähestymistapaa transgressioksi ja immersioksi. Transgressiossa pyritään venyttämään alustan rajoja ja aikaansaamaan sille epätyypillisiä asioita, kun taas immersiossa keskitytään siihen, mitä pidetään alustalle luonteenomaisena.

Teknisten ja esteettisten ihanteiden muuttumista voidaan havaita myös muissa ilmiöissä, jotka ovat levittäytyneet demoskenestä sen ulkopuolelle. Träkkerikulttuurin edustajille ”demo music” saattaa tarkoittaa lähinnä tietynlaista träkkerimusiikin tyyliä (Lysloff 2003). Yabsley (2007) puolestaan puhuu ”sukupolvikuilusta” modernin chip-musiikin ja demo- ja träkkeriskenejen välillä: demoskene on tekniikan kehittämiseen keskittyessään luonut sen infrastruktuurin, jonka päällä nykyinen chiptune-kulttuuri toimii (vrt. Polymeropoulou tässä numerossa).

Ihanteiden eroavuuksista huolimatta bytebeatin estetiikka on kuitenkin selkeästi minimalistista tavalla, josta Carlsson (2010) käyttää nimitystä ”Digital Economics”. Itseilmaisu vähäisellä resurssimäärällä tiukkojen teknisten rajoitusten puitteissa toimii niin chip- kuin bytebeat-muusikoillekin motivaattorina ja inspiraationlähteenä.

Bytebeatin kehityskaaressa näkyy myös ”tekniikan kesyttäminen”, joka tapahtuu apuvälineiden rakenteluna sekä teorian ja käytännön osaamisen kehittymisenä. Alkuvaiheessa vain pieni määrä tekijöitä askarteli bytebeatin parissa C-kääntäjää käyttäen, mutta www-pohjainen työkalu toi satunnaiset kokeilut jopa ohjelmointitaidottomien ulottuville. Kokeilut olivat alkuvaiheessa kaikilla osallistujilla melko umpimähkäisiä, sillä kokeneetkaan ääniohjelmoijat eivät voineet suoraan soveltaa ymmärrystään näin äärimmäisen lyhyissä lausekkeissa. Myöhemmin rakentelun tueksi syntyi teoriaa ja käytännön niksejä, ja osaamisella päteminen tuli mahdolliseksi.

Kuva 3. BitWiz Audio Synth, bytebeat-sovellus Applen mobiililaitteille. Lähde: http://kymatica.com/Software/BitWiz

Vertailu VIC-20-demojen kehityskaareen

Bytebeatin vaiheita vastaava kehityskaari on löydettävissä myös muutamista muista demoskenen teoskategorioista ja alustoista, esimerkiksi Commodore VIC-20 -demoista, joiden kehitykseen osallistuin aktiivisesti 2000-luvun alkupuolella.

Commodore VIC-20 ei kuulu ns. perinteisiin demoalustoihin, sillä VIC-20-käyttäjät ehtivät suuressa määrin siirtyä C-64:ään jo ennen alakulttuurin syntyä. VIC-20-demot olivat pitkään lähinnä kuriositeetteja: Pouet.netin tietokannassa on vain kahdeksan ennen vuotta 2002 julkaistua VIC-20-teosta. Laite nostettiin kuitenkin tietoisella työllä aktiiviseksi demoalustaksi vuonna 2002, ja kyseiseltä vuodelta onkin kannassa kaikkiaan 29 teosta. Vuonna 2003 julkaistuja teoksia on enää 13 ja vuonna 2004 julkaistuja 6. VIC-20:n kaari demoalustana voidaan siis jakaa bytebeatin tavoin pioneerikauteen, aktiiviseen innostuspiikkiin ja piikinjälkeiseen aikaan.

Suurin osa vuonna 2002 julkaistuista VIC-20-demoista on luonteeltaan yksittäisten uusien temppujen esittelyjä. Laitteen video- ja äänipiirin mahdollisuuksia tutkittiin ahkerasti, ja etenkin muiden alustojen demoista tuttuja efektejä uudelleentoteutettiin VIC-20:llä. Laitteiston tuntemattomia piirteitä kartoitettiin vielä myöhemminkin, ja niiden toiminnasta kehitettiin teoriaa. Tämä on verrattavissa siihen, kuinka suurimman bytebeat-innostuksen aikana etsittiin etenkin lausekkeita, jotka tuottaisivat ennestään tutussa musiikissa esiintyviä piirteitä, ja myöhemmässä vaiheessa alettiin rakentaa myös teoriaa etsinnän tueksi.

VIC-20:n ”kesyttämisvaiheessa” syntyi myös useita demontekoon tarkoitettuja kehitystyökaluja kuten musiikkieditori *Fisichella* ja grafiikkaeditorit *Picasso* ja *Brickshop*. Bytebeatin tapauksessa keskeinen käännekohta oli lausekkeiden testaamiseen sopivan www-sivun ilmestyminen, ja myös suoraviivaisempien musiikkieditorien kehittämistä sivuttiin Pouet.netin keskusteluketjussa monta kertaa. Erilaisten työkalujen ja valmisratkaisujen ilmestyminen madaltaa etenkin uusien tulokkaiden kynnystä teosten tuottamiseen.

Bytebeatin pääalustaksi konkretisoitui www-testaussivu, ja alkuvaiheessa käytetty C jäi taka-alalle. Vastaavasti VIC-20 -demoja oli aiemmin julkaistu eri tavoin laajennetuille laitteistoille, mutta vuonna 2002 pääkokoonpanoksi vakiintui muistilaajentamaton laite levyasemalla varustettuna. Alustan riittävän tarkka määrittely on tärkeää sekä tasavertaisen kilpailun mahdollistamiseksi että teosten yhteensopivuuden varmistamiseksi.

Vuonna 2003 julkaistuihin VIC-20-demoihin nähtiin jo enemmän aikaa ja vaivaa. Julkaisutahti oli hitaampi, ja suurin osa demoista julkaistiin demopartyjen kilpailuissa. Tämän jälkeen tahti laantui vuosi vuodelta, kunnes vuonna 2007 julkaisujen määrä lähti uudelleen nousuun. Vastaavaa aaltoliikettä on näkynyt myös bytebeatin aktiivisuudessa: satunnaiset uudet löydökset ovat innostaneet muitakin tekemään jotain uutta.

Lopuksi

Bytebeat ei muodostunut demoskenen piirissä kovinkaan vakiintuneeksi teoskategoriaksi lupaavasta alusta huolimatta. Tekniikasta kiinnostuneita harrastajia on kuitenkin edelleen, ja sillä saattoi hyvinkin olla vaikutusta pikkuintrojen musiikkiteknologiseen kehitykseen. Vaikutusta on kuitenkin tässä vaiheessa turhan aikaista arvioida. Bytebeat-tyyliset, perinteisessä hakkerikulttuurissa ”maagisiksi” kutsutut lähestymistavat tulevat kuitenkin joka tapauksessa demoskenessä aina vain relevantimmiksi, mikäli demoskenen keskittyminen aina vain pienempiin kokokategorioihin jatkuu.

Bytebeat sai aikaan myös monia mielenkiintoisia demoskenen ulkopuolisia kehityskulkuja, joita tämä artikkeli pääsi aiherajauksensakin vuoksi vain ohimennen sivuamaan. Erityisen mielenkiintoisena pidän mahdollista ideoiden ja tekniikoiden jatkokehittelyä kokeellisen musiikin ja musiikkiteknologian piirissä.

Aktiivinen bytebeatin suosiokausi oli monin tavoin poikkeuksellinen ilmiö demoskenellä avoimen kehityskulkunsa, matalan osallistumiskynnyksensä ja hyvin nopean ajallisen etenemisensä vuoksi. Se haastaa siksi monet alakulttuuriin liittyvät stereotyyppiset käsitykset: demokulttuurin teknisesti äärimmäisetkään osa-alueet eivät ole pelkkää vuosikausien ajan tekniikalle omistautuneiden huippuosaajien pitkäjänteistä puurtamista, vaan niissäkin on tilaa kevyemmille sivujuonteille.

Koska bytebeat-lausekkeiden maailma on teknisesti melkoisen yksinkertainen ja rajoittunut, voidaan sitä pitää monin tavoin lähestyttävämpänä kuin esimerkiksi alustatutkimuksen tähän asti suosimia konkreettisia laitealustoja. Bytebeat tuo selvästi esiin sen, kuinka alustastason alkeiselementit ohjaavat niistä rakennettujen ohjelmien muodostumista niin tekniikan kuin estetiikankin kannalta. Tämän sisäistäminen on hyväksi kenelle tahansa, joka haluaa ymmärtää tietoteknisten alustojen olemusta ja koko digitaalisen kulttuurin perustaa.

Lähteet

Aineisto

Beeler, Michael, William R. Gosper ja Rich Schroeppel. 1972. ”HAKMEM.” Memo 239. Massachusetts: Artificial Intelligence Laboratory, Massachusetts Institute of Technology. Tarkistettu 8.4.2014. http://www.inwap.com/pdp10/hbaker/hakmem/hakmem.html.

Hackaday. 2011. ”AVR chiptune project turns this simple code into music.” Tarkistettu 8.4.2014. http://hackaday.com/2011/10/03/avr-chiptune-project-turns-this-simple-code-into-music/

Hacker News. 2011. ”Algorithmic symphonies from one line of code.” Tarkistettu 8.4.2014. https://news.ycombinator.com/item?id=3063359

Pouet.net. 2011. ”Experimental music from very short C programs.” Tarkistettu 8.4.2014. http://www.pouet.net/topic.php?which=8357.

Reddit. 2011a. ”Algorithmic symphonies from one line of code – how and why?” Tarkistettu 8.4.2014. http://www.reddit.com/r/programming/comments/kyj77/algorithmic_symphonies_from_one_line_of_code_how/

Reddit. 2011b. ”Experimental one-line algorithmic music.” Tarkistettu 8.4.2014. http://www.reddit.com/r/programming/comments/mbakl/experimental_oneline_algorithmic_music/

Sitaker, Kragen Javier. 2011. ”Bytebeat.” Tarkistettu 8.4.2014. http://canonical.org/~kragen/bytebeat/

SuperCollider. 2012. ”News in 3.5.” Tarkistettu 8.4.2014. http://doc.sccode.org/Guides/News-3_5.html

The Wire. 2009. ”SuperCollider 140.” Tarkistettu 8.4.2014. http://thewire.co.uk/audio/tracks/supercollider-140.1

Youtube. 2011a. ”Experimental music from very short C programs.” Tarkistettu 8.4.2014. http://www.youtube.com/watch?v=GtQdIYUtAHg

Youtube. 2011b. ”Experimental one-line algorithmic music – the 2nd interation.” Tarkistettu 8.4.2014. http://www.youtube.com/watch?v=qlrs2Vorw2Y

Youtube. 2011c. ”Music from very short programs – the 3rd iteration.” Tarkistettu 8.4.2014. http://www.youtube.com/watch?v=tCRPUv8V22o

Kirjallisuus

Bogost, Ian ja Nick Montfort. 2007. ”New Media as Material Constraint: An Introduction to Platform Studies.” Durham NC: Duke University, 1st International HASTAC Conference.

Carlsson, Anders. 2010. *Power Users and Retro Puppets: A Critical Study of the Methods and Motivations in Chipmusic.* Maisterintutkielma. Lund: Lund University, Department of Media and Communications Studies.

Clarke, Arthur C. 1973. *Profiles of the Future: An Inquiry into the Limits of the Possible*. New York: Harper & Row.

Heikkilä, Ville-Matias. 2011a. ”Algorithmic symphonies from one line of code – how and why?” Tarkistettu 8.4.2014. http://countercomplex.blogspot.fi/2011/10/algorithmic-symphonies-from-one-line-of.html

Heikkilä, Ville-Matias. 2011b. ”Some deep analysis of one-line music programs.” Tarkistettu 8.4.2014. http://countercomplex.blogspot.fi/2011/10/some-deep-analysis-of-one-line-music.html

Heikkilä, Ville-Matias. 2011c. ”Discovering Novel Computer Music Techniques by Exploring the Space of Short Computer Programs.” Tarkistettu 8.4.2014. http://arxiv.org/abs/1112.1368

Kaliakatsos-Papakostas, Maximos, Michael G. Epitropakis, Andreas Floros ja Michael N. Vrahatis. 2012. ”Interactive Evolution of 8–bit melodies with Genetic Programming towards Finding Aesthetic Measures for Sound.” *Evolutionary and Biologically Inspired Music, Sound, Art and Design* 7247:141-52.

Lysloff, Rene. 2003. ”Musical Life in Softcity: An Internet Ethnography.” Teoksessa *Music and Technoculture,* toimittaneet Rene Lysloff ja Leslie Gay, 23-63. Middletown: Wesleyan University Press.

Montfort, Nick, Patsy Baudoin, John Bell, Ian Bogost, Jeremy Douglass, Mark C. Marino, Michael Mateas, Casey Reas, Mark Sample ja Noah Vawter. 2012. *10 PRINT CHR$(205.5 + RND(1)); : GOTO 10*. Massachusetts: MIT Press.

Raymond, Eric S. 1996. *The New Hacker's Dictionary.* Massachusetts: MIT Press, 3. painos.

Raymond, Eric S. 2004. ”Jargon File.” Tarkistettu 8.4.2014. http://www.catb.org/jargon/

Reunanen, Markku. 2010. *Computer Demos – What Makes Them Tick?* Lisensiaatintutkimus. Espoo: Aalto-yliopiston teknillinen korkeakoulu, Mediatekniikan laitos.

Reunanen, Markku. 2013. ”Neljän kilotavun taide.” *Wider Screen* 2-3/2013. http://widerscreen.fi/numerot/2013-2-3/neljan-kilotavun-taide/

Tolonen, Tero, Vesa Välimäki ja Matti Karjalainen, 1998. Evaluation of Modern Sound Synthesis Methods. Raportti 8. Espoo: Teknillinen korkeakoulu, Akustiikan ja signaalinkäsittelyn laboratorio.

Yabsley, Alex. 2007. *The Sound of Playing: A Study into the Music and Culture of Chiptunes.* Musiikkiteknologian kandidaatintutkielma, Queensland Conservatorium, Griffith University.