Äänen syntetisointi ja virtuaali-instrumenttien kehittäminen

Turun yliopisto
Tietotekniikan laitos
TkK-tutkielma
Tietotekniikka
Toukokuu 2025
Niklas Laiho

TURUN YLIOPISTO
Tietotekniikan laitos

Niklas Laiho: Äänen syntetisointi ja virtuaali-instrumenttien kehittäminen

TkK-tutkielma, 22 s.
Tietotekniikka
Toukokuu 2025

Virtuaali-instrumentit ovat instrumentteja, joissa ääni tuotetaan digitaalisesti. Jotta niitä voidaan kehittää, täytyy tietää minkälaisia äänisynteesimenetelmiä ja kehitystyökaluja on olemassa, ja kuinka ne käytännössä toimivat.
Tutkielmassa tarkastellaan useita äänisynteesimenetelmiä. Menetelmät ovat jaettu eri kategorioihin sen mukaan, miten ääni tuotetaan. Tutkielmaan valitut menetelmät ovat yleisimmin käytettyjä. Kehitystyökaluista käsitellään niiden toimintaperiaatteita ja annetaan yksinkertaisia esimerkkejä niillä toteutetuista ohjelmista.
Tutkielma korostaa erilaisten äänisynteesimenetelmien ominaisuuksia ja niiden tapoja tuottaa ääntä. Huomiota kiinnitetään menetelmien laskennalliseen tehokkuuteen ja tuotettujen äänien ominaisuuksiin. Tutkielmassa myös esitellään työkaluja, joilla näitä synteesimenetelmiä käyttäviä virtuaali-instrumentteja voidaan kehittää.

Asiasanat: virtuaali-instrumentti, äänisynteesi, VST

Sisällys

1 Johdanto

Tutkielmassa käsitellään virtuaali-instrumentteja, niissä käytettyjä synteesimenetelmiä ja niiden kehittämistä. Jotta voidaan kehittää virtuaali-instrumentteja, täytyy tietää eri synteesimenetelmien toimintaperiaatteet, ja tulee tuntea sopivat työkalut kehitystyöhön. Tutkielmassa synteesimenetelmistä tutkitaan niiden toimintaperiaatteita ja käyttökohteita. Käsittelyn kohteeksi on valittu yleisimpiä synteesimenetelmiä. Menetelmät on jaetta kolmeen osaan: sample-pohjaiset-, perinteiset- ja fyysisen mallinnuksen synteesimenetelmät. Näiden lisäksi tutkielmassa käsitellään virtuaali-instrumenttien kehittämistä ja tutkitaan erilaisten kehitystyökalujen ominaisuuksia.

1.1 Tutkimuskysymykset

TK1: Mitä äänisynteesin menetelmiä virtuaali-instrumenteissa käytetään?

Virtuaali-instrumenttia käyttäessä ei tule ajatelleeksi mitä kaikkea sen toteuttamiseksi on tehty. Mitkä sis ovat tärkeimpiä äänisynteesin menetelmiä? Minkälaisia ääniä tietyillä menetelmillä saadaan luotua?

TK2: Miten virtuaali-instrumentteja kehitetään?

Audiosovellusten, kuten virtuaali-instrumenttien kehitysprosessi on vaativa ja monia asioita tulee ottaa huomioon. Mitä erilaisia työkaluja virtuaali-instrumenttien kehittämiseen on olemassa? Mitä tulee ottaa huomioon kehitysprosessissa?

1.2 Tiedonhaku

Kirjallisuus on suurimmalta osin haettu Google Scholarista. Jotkut artikkeleista on peräisin toisten artikkeleiden lähteistä. Lähteenä on käytetty myös nettisivuja silloin, kun sopivia artikkeleita ei ole löytynyt. Nettisivut ovat dokumentaatioita ja yliopistojen nettisivuja, joten ne ovat luotettavia. Jokaista lukua ja alalukua varten on käytetty omaa hakulauseketta, koska yhtenäinen hakulauseke ei olisi voinut kattaa kaikkia tutkielmassa käsiteltyjä asioita.

1.3 Tutkielman rakenne

Luvussa kaksi käydään läpi digitaaliseen musiikkiin liittyviä käsitteitä. Käsitteistä pääpaino on VST teknologialla. Luvussa kolme käsitellään erilaisia äänisynteesin menetelmiä, joita voidaan käyttää virtuaali-instrumenteissa. Tutkielmassa on pyritty käsittelemään yleisimmät menetelmät. Jokaiselle esitellylle menetelmälle on oma alalukunsa. Menetelmistä käydään läpi niiden toimintaperiaate korkeammalla tasolla ja niiden käyttökohteita. Luvussa neljä käsitellään virtuaali-instrumenttien kehityksessä käytettäviä työkaluja. Alaluvuissa käsitellään kussakin, jotakin yksittäistä työkalua. Esiteltäviksi työkaluiksi on valittu sellaisia, joiden toimintaperiaatteet eroavat toisistaan.

2 Keskeiset käsitteet

2.1 Virtual Studio Technology (VST)

Virtual Studio Technology, eli VST, on Steinbergin vuonna 1996 julkaisema rajapinta, jota käyetään digitaaliseen musiikintuotantoon tarkoitettujen liitännäisten (plugin) kehittämiseen. VST mahdollistaa sen, että tietokoneella musiikkia tekevä muusikko, voi käyttää erilaisia instrumentteja ja ääntä muokkaavia efektejä sävellyksissään, joita tavallisesti voisi käyttää vain studiolla, tai omistamalla kyseisen efektin tai instrumentin fyysisenä kappaleena. Tämä tekee musiikin itsenäisestä tuottamisesta saavutettavampaa, sillä muusikoiden ei tarvitse omistaa kalliita laitteita, vaan kaikki tarvittava saadaan asennettua tietokoneelle. [1] VST:n lisäksi on olemassa muitakin audioliitännäisten kehittämiseen tarkoitettuja teknologioita, kuten Applen Audio Units (AU), ProTools-ohjelmistossa käytetty Avid Audio eXtension (AAX) ja linuxille kehitetty LADSPA [2][3]. VST on näistä kuitenkin yleisin, jonka takia se on valittu tutkielmassa tarkastelun kohteeksi.

Ensimmäinen VST-versio (1996) tuki ainoastaan liitännäisiä, jotka prosessoivat jo olemassa olevaa äänisignaalia. [1] Tällaisia liitännäisiä kutsutaan nimellä VSTfx, eli VST efekti. Ääntä tuottavia liitännäisiä pystyttiin kehittämään vasta VST:n toisen version VST2:n (1999) myötä. Muutoksena aiempaan versioon oli se, että nyt liitännäiset pystyivät ottamaan vastaan MIDI (Musical Instrument Digital Interface) dataa. VST:n uusin pääjulkaisu on VST3, vuodelta 2008. Versiossa uutta oli

äänen sisääntulot virtuaali-instrumentteja varten ja useamman MIDI-sisääntulon ja -ulostulon tuki.

VST-liitännäisten käyttäminen

VST-liitännäisiä käytetään usein jossakin host-ympäristössä [4]. Yleisimpiä host-ympäristöjä ovat digitaaliset äänityöasemat (Digital Audio Workstation, DAW) [1]. VST-liitännäinen voidaan ladata DAW-ohjelmaan, jolloin sitä voidaan käyttää soittamiseen ja soiton äänittämiseen esimerkiksi MIDI-ohjaimen avulla. Soitto voidaan äänittää suoraan äänisignaaliksi, tai MIDI-dataksi. Jos soitto on äänitetty MIDI-dataksi, on mahdollista vaihtaa jälkeenpäin VST-instrumenttia, joka soittaa nuotteja MIDI-datasta.

DAW-ohjelmistojen lisäksi VST-liitännäisiä voidaan käyttää muissakin host-ympäristöissä. Esimerkiksi live-esiintymisiä varten on kehitetty kevyitä host-ympäristöjä, johon VST-liitännäinen ladataan soitettavaksi reaaliajassa. Kevyt host-ohjelma on DAW-ohjelmistoa parempi tähän tarkoitukseen, koska se vaatii vähemmän laskentatehoa, joka minimoi viiveen soitossa. Kevyemmissä host-ohjelmissa käyttöliittymä on myös mukautettu vastaamaan live-esiintymisen tarpeisiin. Tällaisia host-ympäristöjä ovat esimerkiksi Cantabile tai ilmainen VSTHost [5][6]. Joistakin VST-liitännäisistä on olemassa standalone-versio, jolloin ne eivät tarvitse hostia ollenkaan, vaan niihin on ohjelmoitu sisäinen MIDI-tuki ja audio-ulostulo. [7]

VST-efektien käyttö poikkeaa VST-instrumenteista siinä, että ne eivät itsessään tuota ääntä, vaan ne ketjutetaan jonkin äänilähteen perään. Esimerkiksi DAW-ohjelmistoissa efektit asetetaan raidalle, jossa on jo valmiina asetettuna jokin virtuaali-instrumentti. Myös aiemmin mainitut VST-liitännäisten host-ympäristöt Cantabile ja VSTHost tukevat VST-efektejä. Efektien järjestyksellä on suuri merkitys siihen, miltä niiden yhdistelmä kuulostaa. Jos säröefetkin perään asetetaan kaiku, kaiun vaikutus ulottuu jo säröön prosessoituun signaaliin, jolloin säröinen ääni hautaan-

tuu kaikuun. Jos kaiku taas sijoitetaan ennen säröä, särö prosessoi myös kaikuefetkin aiheuttamaa jälkikaikua. Musiikissa efektejä ovat esimerkiksi erilaiset kaiut, säröt ja viiveet. Efektien tarkoitus on prosessoida äänisignaalia erilaisen kuuloiseksi. Efetkejä voidaan lisätä joko ennalta äänitettyyn äänisignaaliin, tai MIDI-dataa lukevaan virtuaali-instrumenttiin.

Tekninen toiminta

VST-liitännäiset koostuvat tyypillisesti kahdesta osasta: graafisesta käyttöliittymästä (GUI) ja digitaalisesta signaalinkäsittelystä (DSP) vastaavasta osasta. [7] Usein VST-liitännäisten käyttöliittymä koostuu erilaisista nupeista, liukusäätimistä ja näytöistä, jotka esittävät graafisesti tuotetun äänisignaalin ominaisuuksia. Nupeilla ja liukusäätimillä käyttäjä voi ohjata liitännäisen parametreja, joita ovat esimerkiksi äänenvoimakkuus ja panorointi.

Se, minkälaista dataa VST-liitännäiseen syötetään, ja minkälaista dataa siitä saadaan ulostulona, riippuu siitä, minkä tyyppinen liitännäinen on kyseessä. VST-instrumentti ottaa sisääntulona MIDI-dataa ja antaa ulostulona äänisignaalin. Efekti taas ottaa sisääntulona äänisignaalin, jota se prosessoi uudestaan, jolloin ulostulona on muokattu äänisignaali. Jos liitännäinen on MIDI-efekti, se muokkaa MIDI-dataa siten, että datan sisältämät nuotit käyttäytyvät eri tavalla. Tällöin MIDI-efektissä sisääntulo ja ulostulo ovat molemmat MIDI-dataa.

Liitännäisen tyyppi	**Sisääntulo**	**Prosessi**	**Ulostulo**
Instrumentti	MIDI	Syntetisointi/Samplaus	Audio
Efekti	Audio	Audioprosessointi	Audio
MIDI efekti	MIDI	MIDI-datan prosessointi	MIDI

VST3 API on kokoelma rajapintoja, joilla voidaan kehittää äänenkäsittelykomponentteja, joita ovat esimerkiksi virtuaali-instrumentit [8]. Host-ympäristöt voivat ladata näitä komponentteja

VST3-liitännäiset ovat kaksiosaisia. Ne koostuvat rajapinnoista IAudioProces-

sor, eli äänenkäsittelystä vastaavasta osasta ja IEditController, eli käyttöliittymästä vastaava osa. Näistä ensimmäinen, eli IAudioProcessor siis tekee kaiken, mikä liittyy äänen muokkaamiseen tai generoimiseen. Tämä osa liitännäisestä pyörii host-ympäristön reaaliaikaisessa audiosäikeessä. IEditController-rajapinta puolestaan hoitaa liitännäisen parametreihin, asetuksiin ja käyttöliittymään liittyvät asiat. Nämä kaksi osaa ovat erotettu toisistaan VST3-liitännäisissä, suorituskyvyn parantamiseksi. Käyttöliittymää ei välttämättä tarvitse päivittää jatkuvasti, jolloin varsinaiselle äänen prosessoinnille jää enemmän aikaa. [8]

2.2 MIDI (Musical Instrument Digital Interface)

MIDI on tekninen standardiprotokolla musikaalisen datan esittämiseen. Protokolla määrää, miten musiikki-instrumenttien, kuten syntetisaattoreiden, tulee kommunikoida tietokoneen kanssa.

MIDI-data koostuu tapahtumista. Tapahtumalla tarkoitetaan paria, johon kuuluu kaksi komponenttia: aika ja viesti. Aika sisältää tiedon siitä, kuinka kauan odotetaan ennenkuin toistetaan seuraava viesti datavirrassa. Viesti taas sisältää komennon ja dataa. Komentoja ovat esimerkiksi Note-on (0×90) ja Note-off (0×80), jotka kertovat milloin nuotin tulee soida ja milloin soimisen loppua. Data taas sisältää tiedon esimerkiksi nuotin korkeudesta ja voimakkuudesta. MIDI-tapahtuma voisi siis näyttää seuraavalta: 0×90 0×3c 0×28. Tässä esimerkissä 0x90 tarkoittaa Note-on kanavalla 0. 0x3c kertoo, että tulee soittaa nuotti C4. 0×28 taas tarkoittaa, että nuotti soitetaan voimakkuudella 40. Eli kokonaisuudessaan MIDI-tapahtuma tarkoittaa, että soitetaan C4 kanavalla 0 voimakkuudella 40. Heksadesimaali kääntyy binääriksi: 1001 0000 0011 1011 0010 1000. [9]

MIDIä käyttäviä ohjaimia on kehitetty digitaalisen musiikintuotannon helpottamiseksi. Ohjaimissa on usein koskettimet tai painikkeet samplejen toistamista varten. Ohjaimet toimivat siten, että kun käyttäjä painaa kosketinta, ohjaimesta läh-

tee tietokoneelle tieto siitä, mikä nuotti tulee soittaa, jolloin tietokone soittaa nuotin virtuaali-instrumentilla, joka on määritetty ohjaimelle.

Kuva 2.1: Akai MPK mini MK3 -MIDI ohjain
`https://www.akaipro.com/mpk-mini-mk3`

MIDIä voidaan käyttää myös ilman ulkoista MIDI-ohjainta. Nuotteja voidaan ohjelmoida MIDI protokollan mukaisesti esimerkiksi sekvenseriohjelmaan, jossa voidaan määrittää jokin virtuaali-instrumentti soittamaan näitä nuotteja. MIDI data ei siis sisällä ääntä, vaan tiedon siitä, miten sille määritettyä ääntä tulisi ohjata.

3 Synteesimenetelmät

3.1 Sample-pohjainen synteesi

Jotkut synteesimenetelmät perustuvat äänisignaalin näytteistykseen (sampling), jossa analoginen äänisignaali muutetaan digitaaliseen muotoon, jotta sitä voidaan käyttää virtuaali-instrumenteissa. Tämä tarkoittaa sitä, että kun soittaa kitaralla nuotin, tai äänittää mikrofonilla rummun lyönnin, voidaan tämä tallentaa ääninäytteenä, jolloin sitä voidaan toistaa, muokata ja yhdistää eri ääniin virtuaali-instrumenteissa. Näytteitä voidaan soittaa sellaisenaan samplereilla, joissa niihin voidaan myös yhdistää erilaisia efektejä. Näytteiden avulla on myös mahdollista mallintaa kokonainen soitin virtuaali-instrumentiksi ottamalla näyte kaikista mahdollisista äänistä, joita soitin tuottaa. [10]

Jotta ääninäyte saadaan tallennettua tietokoneelle, se täytyy muuntaa digitaaliseen muotoon. Tätä varten on kehitetty analogi-digitaalimuuntimia (A/D). A/D-muunnin näytteistää signaalin näytteenottotaajuuden mukaisin aikavälein ja kvantisoi jokaisen näytteen numeeriseksi arvoksi. [11] Tietokoneessa A/D-muunnin on äänikortissa.

Näytteistyksessä noudatetaan Nyquistin teoreemaa. Sen mukaan analogisesta signaalista tulee ottaa näytteitä vähintään kaksinkertaisella taajuudella alkuperäisen signaalin korkeimpaan taajuuten nähden, jotta signaali voidaan rekonstruoida tarkasti. Yleisimmin käytetty näytteenottotaajuus on 44,1 kHz, joka perustuu sii-

hen, että ihmisen kuuloaisti ei tunnista ääniä yli 20 kHz, jolloin on turhaa ottaa korkeampia taajuuksia mukaan näytteistykseen, sillä siitä ei olisi hyötyä, ja se kasvattaisi vain datan määrää. [12]

3.1.1 Granulaarinen synteesi

Granulaarinen synteesi voi perustua myös ääninäytteisiin, vaikka siinäkin usein käytetään syntetisoitua signaalia. Synteesimenetelmän toimintaperiaate on se, että näyte pilkotaan pieniksi, usein 1-50 millisekunnin pituisiksi "rakeiksi". Näitä rakeita voidaan toistaa peräkkäin, tai samanaikaisesti, ja niiden ominaisuuksia, kuten sävelkorkeutta, voidaan muokata, jolloin saadaan luotua alkuperäisestä ääninäytteestä poikkeavia ääniä.

Granulaarisen synteesin avulla on mahdollista toistaa ääninäyte eri sävelkorkeudella ilman, että sen pituus muuttuu. Tätä tekniikkaa kutsutaan nimellä pitch-shifting. Tavallisesti, jos näytteen sävelkorkeutta halutaan muuttaa, täytyy se toistaa nopeammin, jolloin näytteen pituus lyhenee, mutta sävelkorkeus nousee. Granulaarinen synteesi siis mahdollistaa myös näytteen pituuden muokkaamisen ilman, että sävelkorkeus muuttuu. Tämän tekniikan nimi on time-stretching. [13]

3.1.2 Wavetable-synteesi

Wavetable-synteesissä käytetään ennalta tallennettuja aaltomuotoja, joka tekee siitä periaatteessa sample-pohjaisen, vaikka menetelmä ei perustukaan ääninäytteiden toistamiseen. Aaltomuodot ovat yhden syklin mittaisia palasia ääninäytteestä. Nämä aaltomuodot järjestetään taulukoksi, josta aaltomuotoja toistetaan. Kun toistetaan ominaisuuksiltaan eriäviä aaltomuotova, saadaan aikaan monipuolisia ääniä. [14]

3.2 Perinteiset synteesimenetelmät

3.2.1 FM-synteesi

FM-synteesi, eli taajuusmodulaatiosynteesi, perustuu alunperin radiotekniikassa käytettyyn taajuusmodulaatioon, josta John Chowning kehitti musiikillsen sovelluksen. Synteesimenetelmän perusidea on se, että aaltoa moduloidaan toisella aallolla, jolloin alkuperäisen aallon taajuus muuttuu toisen aallon taajuuden mukaan. FM-synteesissä tätä ensimmäistä aaltoa, jota halutaan moduloida, kutsutaan kantaja-aalloksi, ja toista aaltoa, jolla ensimmäistä muokataan, moduloivaksi aalloksi. Näiden lisäksi tarvitaan vielä modulaatioindeksi, joka määrittää sen, kuinka paljon moduloiva aalto vaikuttaa kantaja-aaltoon [15]. Yksi kaikkien aikojen tunnetuimmista syntetisaattoreista Yamaha DX7 perustuu FM-synteesiin [16]. DX7-syntetisaattori on mallinnettu virtuaali-instrumentiksi nimeltä dexed, joka on ilmainen. [17]

Perinteisessä FM-synteesissä syntyvä ääni sisältää sille ominaisen sävyn, josta huomaa sen olevan synteettinen. Jos halutaan saavuttaa luonnollinen ääni, tästä ominaissävystä täytyy päästä eroon. Tätä varten on kehitetty ModFM (modified frequency modulation), joka on laajennus FM-synteesiin. ModFM mahdollistaa luonnollisempien äänien syntetisoinnin. [16]

3.2.2 Subtraktiivinen synteesi

Subtraktiivinen synteesi on menetelmä, joka oli yleinen 1960- ja 1970-luvun syntetisaattoreissa. Yksinkertaisesti subtraktiivinen synteesi toimii siten, että ensin generoidaan signaali, jossa on rikas spektri. Sen jälkeen tätä signaalia muokataan suodattimilla, jolloin sen spektristä vähenee taajuuksia suodattimien mukaan [18] [19]. Rikasspektrinen signaali saadaan aikaan oskillaattorilla, joka tuottaa jonkin muun muotoisen aallon, kuin siniaallon, kuten saha-, kantti tai kolmioaallon. Myös valkoista tai pinkkiä kohinaa voidaan käyttää. Näiden aaltojen harmoniset ominaisuu-

den ovat rikkaammat, kuin siniaallolla, joka antaa paremmat mahdollisuudet äänen muokkaamiseen ja uusien äänien luomiseen. [20]

Digitaalisessa subtraktiivisessa synteesissä, jota kutsutaan myös virtuaaliseksi analogisynteesiksi, alkuperäistä analogisen synteesin menetelmää imitoidaan digitaalisesti. Jotta saadaan toteutettua digitaalinen syntetisaattori tällä menetelmällä, täytyy mallintaa alkuperäisten fyysisten syntetisaattoreiden analogisia moduuleja. Mallinnuksen kohteena siis ovat oskillaattorit, suodattimet ja vahvistimet. [18]

3.2.3 Additiivinen synteesi

Additiivinen synteesi perustuu täysin Fourier teoriaan, jonka mukaan mikä tahansa signaali voidaan esittää siniaaltojen summana. Tämä siis tarkoittaa sitä, eetä mikä tahansa ääni saadaan aikaan, kun yhdistellään useita siniaaltoja toisiinsa. Siniaalloilla voi olla eri taajuus, amplitudi ja vaihe, joita muokkaamalla saadaan aikaan monenlaisia ääniä. Menetelmän nimi additiivinen synteesi (lisäävä synteesi) tuleekin siitä, että näitä siniaaltoja lisätään yhteen. Teoriassa additiivisen synteesin avulla olisi mahdollista tuottaa mikä tahansa ääni, mutta mitä monimutkaisemmaksi ääni menee, sitä enemmän tarvitaan laskentatehoa, joten jossakin vaiheessa tietokoneen laskentatehon rajat tulevat vastaan, jolloin reaaliaikainen äänen syntetisointi ei onnistu. [21]

3.3 Fyysinen mallinnus

Fyysinen mallinnus (physical modeling, eli PM) on nimitys eri äänisynteesimetodeista, joissa mallinnetaan oikeiden soittimien akustisia ominaisuuksia numeerisesti. Fyysisen mallinnuksen avulla voidaan kehittää realistisen kuuloisia virtuaali-instrumentteja, jolloin muusikon ei tarvitse omistaa oikeita soittimia voidakseen käyttää niitä kappaleissaan. Fyysinen mallinnus mahdollistaa sen, että virtuaali-

instrumentissa voi olla samat ohjausparametrit, kuin oikeassa soittimessa [22]. Tällöin syntetisoidun äänen dynamiikka vastaa oikeaa soitinta, sillä kaikki ohjausparametrit, kuten esimerkiksi kitaran näppäilyn voimakkuus, kaulan pituus ja rungon materiaalin ominaisuudet voidaan ottaa huomioon. Fyysisen mallinnuksen avulla voidaan jopa mallintaa soittajan sormien liikettä kitaran kielillä [23]. Fyysisen mallinnuksen äänisynteesi ei rajoitu kuitenkaan pelkkään oikeiden soittimien jäljittelyyn, vaan samoilla tekniikoilla voidaan luoda luonnollisia ääniä täysin kuvitelluista soittimista, sillä mallinnetut akustiset ominaisuudet vastaavat äänen tuottamista fyysisessä ympäristössä. [24]

Fyysinen mallinnus eroaa perinteisestä sample-pohjaisesta synteesistä juuri äänen dynamiikassa. Sample-pohjainen synteesi perustuu ennalta tallennettuihin ääninäytteisiin (sampleihin), kun taas fyysisessä mallinnuksessa erilaiset variaatiot ääneen voidaan laskea reaaliajassa numeeristen mallien avulla ilman ääninäytteitä. Tämä mahdollistaa äänen dynamiikan ja soittimen käyttäytymisen realistisemman jäljittelyn.

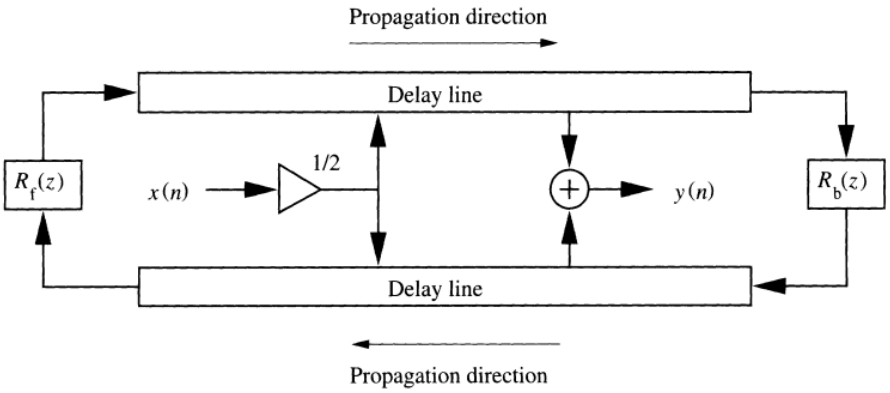
3.3.1 Digitaalinen waveguide-synteesi

Digitaalinen waveguide-synteesi on fyysisen mallinnuksen metodi, jossa äänisignaali kulkee digitaalisen waveguiden, eli aaltoputken läpi. Waveguide koostuu erilaisista suodattimista ja viive-elementeistä, jotka vaikuttavat äänisignaaliin.

Suodattimia käytetään mallintamaan signaalin häviötä, joka tapahtuu esimerkiksi tilanteessa, jossa aalto kulkee kitaran kieltä pitkin tallaan ja heijastuu takaisin. Tässä heijastuksessa aallossa tapahtuu vaiheenkääntö ja se vaimenee, jonka takia waveguideen tarvitaan suodatin, joka muokkaa signaalia samalla tavalla, kuin oikeassa soittimessa. [25] Joissakin tilanteissa suodattimia voidaan käyttää mallintamaan esimerkiksi sormen kosketuksen aiheuttamaa vaimennusta kitaran kielellä. [23]

Viive-elementeillä mallinnetaan aallon etenemistä esimerkiksi kitaran kielessä. Yksinkertaisessa waveguide-mallissa on kaksi viivelinjaa, jotka kulkevat vastakkaisiin suuntiin. Kahden viivelinjan mallista käytetään nimitystä bi-directional delay line, eli kaksisuuntainen viivelinja. [22]

Waveguiden suodattimet ja viive-elementit järjestellään siten, että ne vaikuttavat digitaaliseen signaaliin samalla tavalla, kuin fyysisen soittimen eri osat vaikuttavat ääniaaltoon.



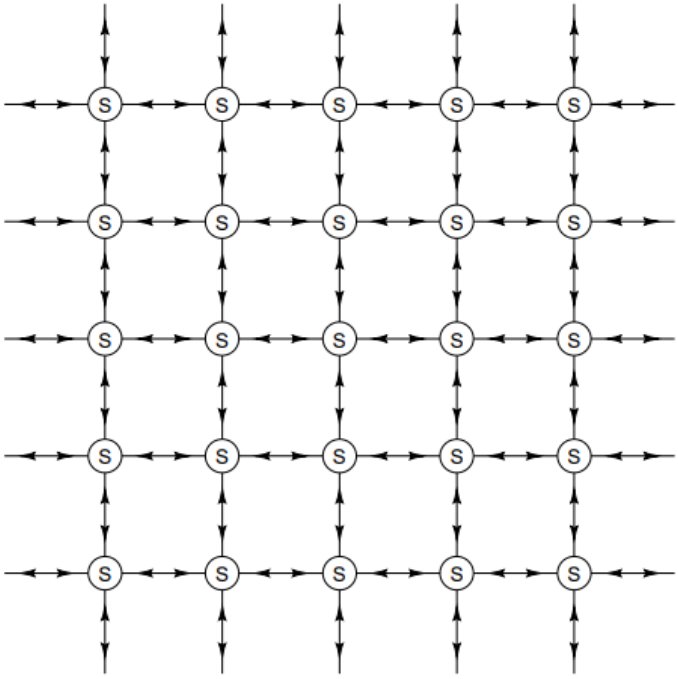
Kuva 3.1: Yksinkertainen kielisoittimen waveguide -malli, jossa kaksi erisuuntaista viive-elementtiä ja kaksi heijastussuodatinta.[25]

3.3.2 Digitaalinen waveguide-verkko

Kuvan waveguide-mallissa aalto etenee kahteen suuntaan yhdessä ulottuvuudessa, jolloin sitä voidaan kutsua yksiulotteiseksi waveguideksi. Yksiulotteisilla waveguideilla saadaan mallinnettua esimerkiksi kieli- ja puhallinsoittimia. Jos halutaan mallintaa ääniaallon etenemistä jollakin pinnalla, kuten rumpukalvolla, tarvitaan waveguideen toinen ulottuvuus, jolloin kyseessä on digitaalinen waveguide-verkko. [26]

Waveguide-verkko voidaan muodostaa yhdistämällä useita kaksisuuntaisia viivelinjoja verkoksi. Viivelinjojen yhtymäkohtiin tulee sirontaliitos (scattering junction), josta signaali jakautuu neljään suuntaan tapauskohtaisesti määritettyjen sääntöjen mukaan. Tällaista verkkoa kutsutaan kaksiulotteiseksi digitaaliseksi waveguide-

verkoksi. [26]



Kuva 3.2: Kaksiulotteinen digitaalinen waveguide-verkko [26]

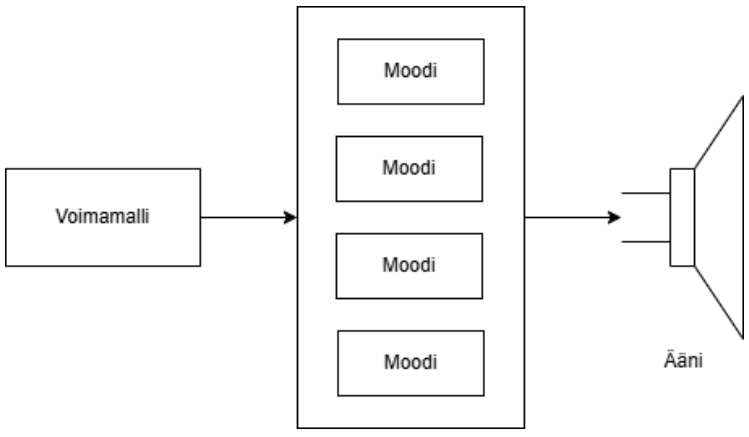
Waveguideista voidaan myös muodostaa kolmiulotteinen verkko. Tällöin signaalin täytyy jakauta kuuteen suuntaan sirontaliitoksissa, jotta x-, y- ja z-akseleilla olisi kaikilla kahteen suuntaan kulkevia signaaleja. Kolmiulotteisilla aaltoputkiverkoilla voidaan mallintaa esimerkiksi soittimen rungon sisäistä akustiikkaa tai vaikka kokonaisen konserttisalin. Kolmiulotteisessa verkossa haasteena on se, kuinka paljon laskentatehoa tarvitaan simuloimaan äänen kulkemista suuressa tilassa. Kun verrataan kaksiulotteiseen verkkoon, kolmiulotteisessa verkossa olevien liitoskohtien määrä on moninkertainen, jolloin myös laskentaoperaatioiden määrä moninkertaistuu. Tämän seurauksena laajoja kolmiulotteisia verkkoja ei välttämättä voida käyttää reaaliaikaisessa äänisynteesissä, sillä viive olisi liian suuri. [27]

3.3.3 Modaalisynteesi

Modaalisynteesi, eli modal synthesis, on laajalti käytetty fyysisen mallinnukseen perustuva synteesimenetelmä. Se siis perustuu fyysisten kappaleiden ominaisuuksien numeeriseen jäljittelyyn digital waveguide -synteesin tavoin. Erona digital waveguide -synteesiin on kuitenkin se, että modaalisynteesissä ei mallinneta ääniaallon etenemistä soittimessa. Sen sijaan modaalisynteesin lähestymistavassa mallinnetaan soittimen rakenteen modaalisia ominaisuuksia, eli miten soitin resonoi.

Perusajatuksena modaalisynteesissä on siis se, että jokaisella kappaleella on sille ominaisia taajuuksia, joilla se värähtelee. Näitä ominaistaajuuksia kutsutaan modaaleiksi. Yhdellä kappaleella on useita modaaleja, joiden yhdistelmästä syntyy sen uniikki sointi. Modaaleihin vaikuttaa kappaleen muoto, materiaali ja rajaehdot, kuten esimerkisi se, onko kappale kiinnitetty johonkin.

Kappale mallinnetaan joukkona yksittäisiä moodeja, jotka ovat toisistaan riippumattomia. Moodi tarkoittaa yksittäistä värähtelymuotoa, ja kappaleen modaaliset ominaisuudet muodostuvat näiden moodien kokonaisuudesta. Jokaisella moodilla on oma taajuus, vaimennus ja voimakkuus. [28]



Kuva 3.3: Modaalisynteesin toiminta yksinkertaisesti

4 Virtuaali-instrumenttien kehittäminen

4.1 FAUST

FAUST (Functional AUdio STream) on korkean tason funktionaalinen ohjelmointikieli. Se on kehitetty reaaliaikaista digitaalista signaalinprosessointia varten. FAUST:ssa on C++-kääntäjä, joka muuntaa korkean tason koodin C++-tiedostoiksi. FAUST:ssa on sisäänrakennettuna erilaisia arkkitehtuuritiedostoja ja scriptejä, joilla käännetty C++-tiedosto saadaan muunnettua eri formaatteihin. Näitä formaatteja ovat esimerkiksi kaikki yleisimmät audioliitännäisformaatit, kuten VST ja AU. FAUST-lähdekoodilla voidaan myös luoda ohjelmia muille sovelluksille, kuten PureDatalle tai Max/MSP:lle [29]. FAUST-ohjelmointikieltä voidaan käyttää lokaalisti asentamalla se tietokoneelle, mutta sille on olemassa myös selainpohjainen ohjelmointiympäristö [30].

FAUST on korkean tason kieli, joten siinä on sisäänrakennettuja funktioita ja komponentteja, joita voidaan digitaaliseen signaalinkäsittelyyn [29]. Ohjelmointikieli on helppokäyttöinen tämän funktionaalisuuden takia. Esimerkiksi siniaaltoa taajuudella 440 Hz tuottava oskillaattori saadaan määriteltyä yhdellä koodirivillä:

```
prosessi = os.osc(440);
```

Käyttöliittymän luomista varten FAUSTissa on valmiita komponentteja, kuten hs-

lider, joka tarkoittaa horisontaalista liukua. Seuraavat koodirivit muodostavat siniaaltoa tuottavan oskillaattorin, jonka taajuutta voi muuttaa liu'ulla:

```
taajuus = hslider("Taajuus", 440, 20, 2000, 1);
prosessi = os.osc(taajuus);
```

Esimerkkikoodissa hslider-komponentin parametrit ovat järjestyksessä: liu'un käyttäjälle näkyvä nimi, oletusarvo, minimiarvo, maksimiarvo ja askelkoko.

Jos FAUSTilla kirjoitettu ohjelma halutaan kääntää VST-liitännäiseksi, voidaan yksinkertaisesti ajaa komento:

```
faust2juce esimerkkitiedosto.dsp
```

Tämä komento luo automaattisesti JUCE-kehystä käyttävän projektikansion FAUST-koodin pohjalta. JUCEssa projekti saadaan muutettua helposti VST-liitännäiseksi, koska luotu projektikansio sisältää tälle tarvittavan rakenteen. [31]

4.2 JUCE

JUCE on avoimen lähdekoodin ohjelmistokehys, jolla voidaan luoda audiosovelluksia, kuten esimerkiksi VST-liitännäisiä. JUCE sisältää kääreen, joka mahdollistaa liitännäisten kääntämisen kaikille liitännäisformaateille, kuten VST, AudioUnit ja AAX. Kehys mahdollistaa sovellusten järjestelmäriippumattoman kehityksen, joka tarkoittaa, että sama koodi toimii kaikissa ympäristöissä riippumatta käyttöjärjestelmästä. [7] [32]

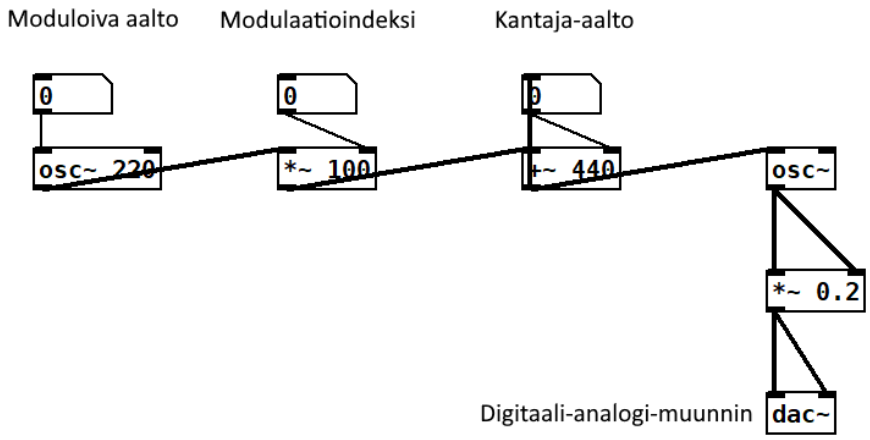
JUCE-kehyksen mukana asennetaan sen oma projektinhallintatyökalu nimeltä Projucer. Sen avulla voidaan luoda projekti helposti eri liitännäistyypeille, koska luodessa projektia voidaan valita mille formaateille projekti ollaan luomassa. Projucer asentaa tarvittavat riippuvuudet eri liitännäisformaateille. [33]

JUCE sisältää luokat AudioProcessor ja AudioProcessorEditor, jotka mahdollistavat ääntä prosessoivien sovellusten kehittämisen. Näistä ensimmäistä, eli AudioProcessor -luokkaa, käytetään äänen generoimisen tai muokkaamisen. Luokkaan sisältyy esimerkiksi funktio processBlock(), jota voidaan pitää tärkeimpänä audiosovelluksen kannalta, sillä sen sisään ohjelmoidaan audion ja MIDI-datan käsittelystä vastaavat ominaisuudet. AudioProcessorEditor -luokkaa käytetään puolestaan graafisen käyttöliittymän kehittämiseen. Luokassa ei siis prosessoida ääntä, vaan luodaan toiminnallisuus sille, että käyttäjä voi säätää äänen parametreja. [32]

4.3 Pure Data

Pure Data on visuaalinen ohjelmointikieli, jolla voidaan kehittää reaaliaikaisia signaalinprosessointiohjelmia [34]. Ohjelmointi tapahtuu yhdistämällä erilaisia objekteja toisiinsa, siten että ne ovat vuorovaikutuksessa keskenään. Objekteja on useaan tarkoitukseen, ja ne voivat olla esimerkiksi ääntä tuottavia, signaalia käsitteleviä tai vaikka matemaattisia operaatioita. Objektit esitetään Pure Datassa "tilde objekteina", eli ne kirjoitetaan muotoon osc~[35]. Tämä tuottaa jatkuvaa siniaaltoa määritellyllä taajuudella. [36]

Pure Datan graafisen ohjelmoinnin avulla on helppoa syntetisoida ääntä eri synteesimenetelmiä käyttäen. Eri menetelmien algoritmit voidaan toteuttaa liittämällä objekteja toisiinsa, eikä perinteistä ohjelmointitaitoa tarvita [35]. Esimerkiksi FM-synteesi voidaan toteuttaa yksinkertaisesti seuraavalla tavalla:



Kuva 4.1: FM-synteesi Pure Datalla. Objekteja, joiden arvoiksi on asetettu 0, voidaan käyttää muokkaamaan taajuuksia ja modulaatioindeksiä.

Pure Datassa luotua projektia on myös mahdollista käyttää VST-liitännäisenä. Tähän tarvitaan kuitenkin ulkoista liitännäistä Pure Dataan, sillä ohjelmassa itsessään ei ole tälläistä ominaisuutta. Camomile on Pure Dataan kehitetty liitännäinen, joka mahdollistaa Pure Data patchin käyttämisen VST-liitännäisenä hostympäristössä. Camomile tukee myös muita liitännäisformaatteja, kuten AU:ta ja LV2:ta. Camomile on kehitetty JUCE-kehyksellä. [37]

5 Yhteenveto

Tutkielmassa pyrittiin vastaamaan kahteen tutkimuskysymykseen, jotka määriteltiin johdantoluvussa. Tutkielma jakautui kahteen aihepiiriin: virtuaali-instrumenttien äänisynteesimenetelmiin ja näiden kehittämiseen.

TK1: Mitä äänisynteesin menetelmiä virtuaali-instrumenteissa käytetään?

Äänisynteesimenetelmiä on monia ja ne voidaan jakaa eri kategorioihin, joita ovat sample-pohjainen synteesi, perinteiset synteesimenetelmät ja fyysinen mallinnus.

Sample-pohjainen synteesi perustuu ennalta tallennettuihin ääninäytteisiin, joita toistetaan joko sellaisenaan, tai muokattuna erilaisin efektein. Jotta ääninäytteitä voidaan toistaa, ne täytyy muuntaa digitaaliseen muotoon näytteistämällä. Näytteistys tapahtuu analogi-digitaalimuuntimella ja siinä noudatetaan Nyquistin teoreemaa. Sample-pohjaisia synteesimenetelmiä ovat granulaarinen synteesi ja wavetable synteesi.

Perinteiset synteesimenetelmät ovat sellaisia, joissa ääni syntetisoidaan jonkin yksinkertaisen algoritmin avulla, joka ei yritä kuvata sitä, miten ääni syntyy akustisissa soittimissa. Nämä synteesimenetelmät ovat ominaisia vanhoille analogisyntetisaattoreille, mutta samoja äänisynteesin algoritmeja voidaan käyttää virtuaali-instrumenteissa. Joissakin tapauksissa myös mallinnetaan analogisyntetisaattoreita komponenttitasolla. Perinteisiä synteesimenetelmiä ovat FM-synteesi, subtraktiivinen synteesi ja additiivinen synteesi.

Fyysinen mallinnus on nimitys sellaisista äänen synteesimenetelmistä, joissa mal-

linnetaan oikeiden soittimien tai kappaleiden akustisia ominaisuuksia. Tästä syystä fyysisen mallinnuksen menetelmillä saadaan aikaan luonnolliselta kuulostavia virtuaali-instrumentteja. Menetelmien avulla on mahdollista myös luoda täysin kuviteltuja instrumentteja, joissa ääni tuotetaan kuitenkin samalla tavalla kuin fyysisessä maailmassa. Fyysisen mallinnuksen synteesimenetelmiä ovat esimerkiksi digitaalinen aaltoputkisynteesi, digitaalinen aaltoputkiverkko ja modaalisynteesi.

TK2: Miten virtuaali-instrumentteja kehitetään?

Virtuaali-instrumenttien kehittäminen vaatii hyviä ohjelmointitaitoja ja tietämystä digitaalisesta signaalinkäsittelystä. Sellaisen luominen tyhjästä olisi vaativa prosessi, koska täytyisi ottaa huomioon esimerkiksi se, miten instrumentti toimisi halutussa formaatissa, kuten VST-liitännäisenä. Myös MIDI-tuki olisi hankalaa ohjelmoida tyhjästä. Tämän kehitysprosessin helpottamiseksi on kehitetty erilaisia työkaluja, joita ovat esimerkiksi FAUST-ohjelmointikieli, JUCE-kehys ja visuaalinen ohjelmointikieli Pure Data.

FAUST on korkean tason ohjelmointikieli, jolla voidaan helposti luoda erilaisia audiosovelluksia, kuten virtuaali-instrumentteja. Siinä on sisäänrakennettuna funktioita ja komponentteja, joita käytetään digitaaliseen signaalinkäsittelyyn ja käyttöliittymän luomiseen. FAUSTissa on C++-kääntäjä ja useita scriptejä, joilla C++-tiedosto saadaan käännettyä haluttuun muotoon, kuten JUCE- tai Pure Data-projektiksi.

JUCE on ohjelmistokehys, jota käytetään audiosovellusten kehittämisessä. JUCE on järjestelmäriippumaton. Kehyksen lisäksi JUCEssa on oma projektinhallintatyökalu Projucer, jolla voidaan alustaa projekti helposti tietyille formaateille. Jos Projucerilla projekti alustetaan esimerkiksi VST-formaatille sopivaksi, työkalu asentaa siihen tarvittavat riippuvuudet itsestään.

Pure Data on visuaalinen ohjelmointikieli, joka erikoistuu reaaliaikaiseen signaalinkäsittelyyn. Siinä voidaan yhdistellä erilaisia objekteja toisiinsa graafisessa ympä-

ristössä. Objekteja ovat esimerkiksi oskillaattorit ja erilaiset säätimet. Pure Data on hyvä sellaisille käyttäjille, joilla ei ole ohjelmointitaitoa, koska graafinen ohjelmointi tekee kehitystyöstä selkeämpää. Pure Datalla ei saa käännettyä ohjelmaa audioliitännäiseksi, mutta tätä varten on kehitetty kolmannen osapuolen liitännäinen Pure Dataan nimeltä Camomile.

Lähdeluettelo

[1] G. Tanev ja A. Bozhinovski, ”Virtual studio technology and its application in digital music production”, 2013.

[2] V. Goudard ja R. Muller, ”Real-time audio plugin architectures”, *Comparative study. IRCAM-Centre Pompidou. France*, 2003.

[3] Avid Technology, Inc., *AAX SDK – Avid Developer*, `https://developer.avid.com/aax/`, Viitattu: 16. kesäkuuta 2025, 2025.

[4] M. J. Yee-King, L. Fedden ja M. d'Inverno, ”Automatic programming of VST sound synthesizers using deep networks and other techniques”, *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 2, nro 2, s. 150–159, 2018.

[5] Hermann Seib, *VSTHost – A Program to Host VST Plugins*, `https://www.hermannseib.com/english/vsthost.htm`, Accessed: 2025-06-16, 2025.

[6] Topten Software, *Cantabile – Software for Performing Musicians*, `https://www.cantabilesoftware.com/`, Accessed: 2025-06-16, 2025.

[7] B. Andrei, M. V. Merino ja I. Malavolta, ”The Ecosystem of Open-Source Music Production Software–A Mining Study on the Development Practices of VST Plugins on GitHub”,

[8] *VST3 API Documentation - VST3 Developer Portal — steinbergmedia.github.io*, `https://steinbergmedia.github.io/vst3_dev_portal/pages/Technical+`

Documentation/API+Documentation/Index.html#the-processing-part, [Accessed 27-05-2025].

[9] H. M. de Oliveira ja R. de Oliveira, "Understanding midi: A painless tutorial on midi format", *arXiv preprint arXiv:1705.05322*, 2017.

[10] E. Getahun, A. Behailu ja M. Tekeba, "Design and implementation of virtual studio technology instrument plug-in for Kirar", *Zede Journal*, vol. 36, s. 41–53, 2018.

[11] R. H. Walden, "Analog-to-digital converter survey and analysis", *IEEE Journal on selected areas in communications*, vol. 17, nro 4, s. 539–550, 1999.

[12] D. Lavry, "Sampling theory for digital audio", *Lavry Engineering, Inc. Available online: http://www. lavryengineering. com/documents/Sampling_ Theory. pdf (checked 24.5. 2010)*, 2004.

[13] C. Roads, "Introduction to granular synthesis", *Computer Music Journal*, vol. 12, nro 2, s. 11–13, 1988.

[14] R. Bristow-Johnson, "Wavetable synthesis 101, a fundamental perspective", teoksessa *Audio engineering society convention 101*, Audio Engineering Society, 1996.

[15] J. M. Chowning, "The synthesis of complex audio spectra by means of frequency modulation", *Journal of the audio engineering society*, vol. 21, nro 7, s. 526–534, 1973.

[16] V. Lazzarini ja J. Timoney, "Theory and practice of modified frequency modulation synthesis", *Journal of the Audio Engineering Society*, vol. 58, nro 6, 2010.

[17] *Dexed — asb2m10.github.io*, `https://asb2m10.github.io/dexed/`, [Accessed 26-05-2025].

[18] A. Huovilainen ja V. Välimäki, "New approaches to digital subtractive synthesis", teoksessa *ICMC*, 2005.

[19] R. A. Moog, "Voltage controlled electronic music modules", *Journal of the Audio Engineering Society*, vol. 13, nro 3, s. 200–206, 1965.

[20] R. Corporation, *Roland - A Beginner's Guide To Subtractive Synthesis — roland.com*, https://www.roland.com/uk/blog/guide-to-subtractive-synthesis/, [Accessed 27-05-2025].

[21] J. O. Smith ja S. CCRMA, "Additive Synthesis (Early Sinusoidal Modeling)", Center for Computer Research in Music ja Acoustics (CCRMA), Stanford University, Online Technical Report, 2025, Saatavilla verkossa: https://ccrma.stanford.edu/~jos/sasp/Additive_Synthesis_Early_Sinusoidal.html.

[22] J. O. Smith, "Physical Modeling Using Digital Waveguides", *Computer Music Journal*, vol. 16, nro 4, s. 74–91, 1992, ISSN: 01489267, 15315169. url: http://www.jstor.org/stable/3680470 (viitattu 28. 03. 2025).

[23] G. Cuzzucoli ja V. Lombardo, "A Physical Model of the Classical Guitar, Including the Player's Touch", *Computer Music Journal*, vol. 23, nro 2, s. 52–69, 1999, ISSN: 01489267, 15315169. url: http://www.jstor.org/stable/3680735 (viitattu 28. 03. 2025).

[24] V. VÄLIMÄKI ja T. TAKALA, "Virtual musical instruments — natural sound using physical models", *Organised Sound*, vol. 1, nro 2, s. 75–86, 1996. DOI: 10.1017/S1355771896000039.

[25] M. Karjalainen, V. Välimäki ja T. Tolonen, "Plucked-String Models: From the Karplus-Strong Algorithm to Digital Waveguides and beyond", *Computer Music Journal*, vol. 22, nro 3, s. 17–32, 1998, ISSN: 01489267, 15315169. url: http://www.jstor.org/stable/3681155 (viitattu 28. 03. 2025).

[26] S. A. Van Duyne ja J. O. Smith, "Physical modeling with the 2-D digital waveguide mesh", s. 40–40, 1993.

[27] D. Murphy, A. Kelloniemi, J. Mullen ja S. Shelley, "Acoustic modeling using the digital waveguide mesh", *IEEE Signal Processing Magazine*, vol. 24, nro 2, s. 55–66, 2007.

[28] K. Van Den Doel ja D. K. Pai, "Modal synthesis for vibrating objects", *Audio Anectodes. AK Peter, Natick, MA*, s. 1–8, 2003.

[29] J. O. Smith III, "Audio signal processing in Faust", *online tutorial: https://ccrma. stanford. edu/jos/aspf*, 2010.

[30] S. Ren, S. Letz, Y. Orlarey et al., "FAUST online IDE: dynamically compile and publish FAUST code as WebAudio Plugins", teoksessa *WAC 2019-5th Web Audio Conference*, 2019.

[31] A. Albouy ja S. Letz, "Faust audio DSP language for JUCE", teoksessa *Linux Audio Conference*, CIEREC, 2017, s. 61–68.

[32] JUCE Developers, *JUCE Documentation: Audio*, `https://docs.juce.com/master/index.html#tag_audio`, Accessed: 2025-06-11, 2025.

[33] JUCE Developers, *Projucer Manual - Welcome*, Accessed: 2025-06-11, JUCE, 2025. url: `https://docs.juce.com/master/projucer_manual.html#projucer_manual_introduction_welcome`.

[34] M. S. Puckette et al., "Pure data", teoksessa *ICMC*, 1997.

[35] M. Puckette et al., "Pure data: another integrated computer music environment", *Proceedings of the second intercollege computer music concerts*, s. 37–41, 1996.

[36] Pure Data Community, *Start Here — Pure Data Documentation*, Accessed: 2025-06-11, n.d. url: `https://puredata.info/docs/StartHere/`.

[37] P. Guillot, "Camomile: creating audio plugins with pure data", teoksessa *Linux Audio Conference*, 2018.