TEKNILLINEN KORKEAKOULU
Elektroniikan, tietoliikenteen ja automaation tiedekunta

Jaakko Pero

NAUHOITETUN JÄLKIKAIUNNAN LISÄÄMINEN ÄÄNISIGNAALIIN REAALIAJASSA

Kandidaatintyö

Espoo 10.12.2009

Työn ohjaaja:

Tapani Pihlajamäki

Tekijä: Jaakko Pero

Työn nimi: Nauhoitetun jälkikaiunnan lisääminen äänisignaaliin reaaliajassa

Päivämäärä: 10.12.2009 Kieli: Suomi Sivumäärä: 6+19

Tutkinto-ohjelma: Elektroniikka ja sähkötekniikka

Vastuuopettaja: Markus Turunen

Ohjaaja: Tapani Pihlajamäki

Työssä tarkastellaan reaaliaikaista konvoluutiokaikua ja sen toteuttamista rinnakkaislaskennalla. Työssä esitetään lyhytaikaiseen Fourier-muunnokseen perustuva tehokas konvoluutioalgoritmi, joka toteutetaan OpenCL-ohjelmointikielellä.

Konvoluutiokaiku mahdollistaa tarkan tilakaiunnan lisäämisen äänisignaaliin. Tilan impulssivasteen tallentaminen on huomattavasti yksinkertaisempaa kuin sen mallintaminen esimerkiksi kampasuodattimilla.

Lyhytaikainen Fourier-muunnos tarjoaa digitaaliseen äänenkäsittelyyn työkalun, jolla mielivaltaisen pitkiä impulssivasteita voidaan lisätä reaaliaikaiseen äänisignaaliin. Yleisesti saatavilla olevat digitaaliset konvoluutiokaikuefektit eivät kuitenkaan hyödynnä kuin pienen osan modernin moniytimisillä suorittimilla ja tehokkaalla näytönohjaimella varustellun työaseman kapasiteetista. Nykyinen energiaa säästävän suoritintekniikan kehitys suosii kellotaajuuden laskua ja rinnakkaisten laskentaydinten määrän kasvua ja näiden hyödyntäminen edellyttää uudenlaista ohjelmointitekniikkaa.

Tässä esitetyllä tekniikalla suurin osa laskennasta voidaan suorittaa näytönohjaimella käyttäen huomattavasti useampaa rinnakkaista prosessia kuin mitä tällä hetkellä on mahdollista yleiskäyttöisillä prosessoreilla. Esitettyä tekniikkaa voidaan hyödyntää paitsi kaikulaitteena myös muissa yhteyksissä esimerkiksi digitaalisen huonekorjauksen toteuttamisessa.

Avainsanat: lyhytaikainen Fourier-muunnos, nopea Fourier-muunnos, konvoluutiokaiku, jälkikaiunta

Esipuhe

Kiitos työn ohjaajalle.

Erityiskiitos Maija Perolle oikoluvusta, kommenteista ja kysymyksistä. Kiitos Heikki Hyytille rakentavasta kritiikistä viimeisellä mahdollisella hetkellä.

Kysymykset olivat ne arvokkaimmat. Vasta sitä ratakiskoa vääntäessään voi todella ymmärtää, mistä itse on puhumassa.

Otaniemi, 10.12.2009

Jaakko Pero

Sisältö

Symbolit, lyhenteet ja määritelmät

Symbolit

n	diskreetti aikamuuttuja
t	jatkuva aikamuuttuja
e	Neperin luku
j	imaginääriyksikkö

Operaattorit

$\sum_{i=0}^{N-1}$	Summa indeksin i yli, kun i käy nollasta N miinus yhteen

Lyhenteet

SIMD	Single Input Multiple Data, yksittäinen käsky kohdistetaan useisiin kohdeaineistoihin samanaikaisesti
SPMD	Single Program Multiple Data, yksittäisen ohjelman kohdistaminen useisiin kohdeaineistoihin samanaikaisesti
CPU	Central Processing Unit, tietokoneen pääsuoritin
GPU	Graphics Processing Unit, grafiikkasuoritin
FFT	Fast Fourier Transform, nopea Fourier-muunnos
LTI	Linear Time-Invariant, lineaarinen ja ajassa muuttumaton
OpenCL	Open Computing Language
STFT	Short-time Fourier Transform, lyhytaikainen Fourier-muunnos

Määritelmät

Suoritin
: Mikroprosessoria tai lyhyemmin prosessoria (engl. processor) kutsutaan tässä työssä yleensä suorittimeksi.

Multiprosessori
: Multiprosessori sisältää useita yhdelle mikropiirille tiheästi pakattuja suorittimia, jotka usein toimivat yhdessä esimerkiksi SIMD-käskyjä suoritettaessa. Usein yhden multiprosessorin sisältämät suoritinytimet jakavat saman muistialueen ja voivat sen kautta vaihtaa tietoja nopeasti.

Grafiikkakiihdytin
: Grafiikkakiihdytin tai grafiikkakortti on tietokoneen väylään liitettävä lisälaite, jonka tehtävä on grafiikan ja kuvamateriaalin nopea käsittely. Kiihdytinkortilla voi olla myös varsinainen näytönohjain, mutta esimerkiksi NVIDIA:n Tesla-sarjan korteissa ei sellaista ole.

1 Johdanto

Kehittyneet audiotekniikan ratkaisut kuten virtuaaliakustiikka edellyttävät, että reaalimaailman syötteitä kyetään reaaliaikaisesti yhdistämään ja muokkaamaan keinotekoisten signaalien ja tilamallien kanssa. Ongelma ei ole laskentatehon puute. Vuosikymmenen takaisen supertietokoneen laskentakapasiteetin ylittäviä laitteita on saatavilla edullisesti. Ongelma ei myöskään ole laskentatehon hyödyntäminen sinänsä – vielä nykyäänkin käyttökelpoiset digitaalisen signaalinkäsittelyn algoritmit kirjoitettiin jo 1950-luvulla. Ongelma on, että yleisesti saatavissa ei juurikaan ole sellaisia käytännön toteutuksia, joissa tehokkaaseen algoritmiin olisi yhdistetty optimoitu laskentakapasiteetin käyttö. Poikkeuksen muodostaa Christian Borßn [1] tuore seminaariesitelmä joka esittelee optimoidun moniytimiselle suorittimelle tarkoitetun kaikuefektin. Käsittely rajoittuu kuitenkin vain CPU-toteutukseen, ja rinnakkaislaskennan toteutus on erilainen kuin tässä työssä.

Alun perin tietokonepelien tarpeisiin suunnitelluille kotitietokoneiden grafiikkaprosessoreille (graphics processing unit, GPU) on viime vuosina kehitetty yleiskäyttöisiä ohjelmointirajapintoja, joilla ne saadaan tekemään esimerkiksi äänenkäsittelyä. Näitä piirejä löytyy nykyään lähes jokaisen työpöydältä – yleensä käyttämättömänä. Grafiikkasuorittimien soveltuvuus raskaaseen liukulukulaskentaan ja erityisesti tuoreempien mallien lukuisat rinnakkaiset laskentaliukuhihnat ovat toistaiseksi jääneet vähälle huomiolle. Useimmat alusta asti kotitietokoneellekin räätälöidyt audiosovellukset hyödyntävät vain yleiskäyttöistä pääsuoritinta (central processing unit, CPU) jättäen grafiikkasuorittimen valtavan potentiaalin täysin hyödyntämättä.

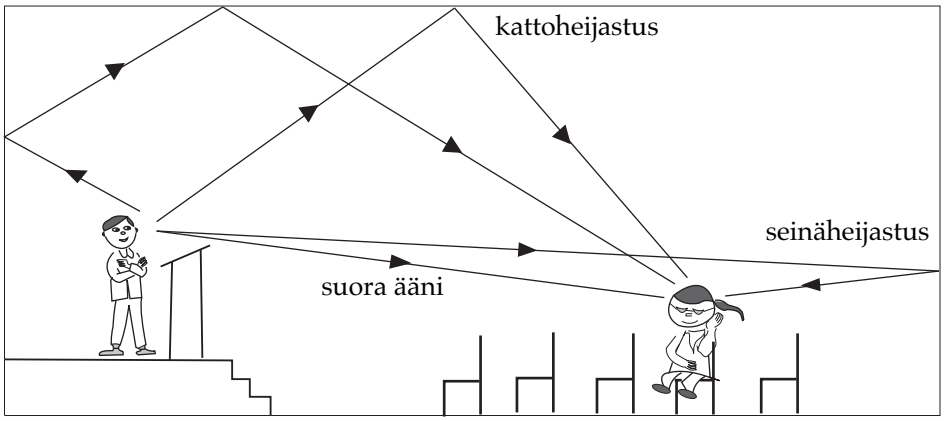
Tässä työssä etsitään reaaliaikaiseen äänisignaalin käsittelyyn uutta rinnakkaislaskentaan ja erityisesti grafiikkakiihdyttimen multiprosessorin ominaisuuksiin perustuvaa tekniikkaa. Työssä esitetään moniytimisille prosessoreille tarkoitettu reaaliaikainen konvoluutiokaikualgoritmi. Algoritmi hyödyntää grafiikkakiihdyttimen massiivista rinnakkaisuutta ja laskee äänisignaalin konvoluution halutun impulssivasteen kanssa tehokkaasti ja reaaliaikaisesti.

Työssä käsitellään vain yksiulotteista monoäänisignaalia. Algoritmia voidaan laajentaa myös monikanavaisiin järjestelmiin, vaikka niitä ei tässä työssä käsitellä. Työ on jaoteltu seuraavasti: toinen luku käsittelee huonekaiuntaa akustiikan teorian näkökulmasta, kolmas luku esittelee keinotekoiseen huonekaiuntaan liittyvän signaalinkäsittelyn teorian, neljännessä luvussa on valittujen tekniikoiden yksityiskohtainen selvitys matemaattisine pohjustuksineen, viides luku käsittelee käytännön toteutusta ja kuudes luku sisältää yhteenvedon.

2 Kaiunnan akustiikka

2.1 Äänen heijastuminen

Sanalla ääni tarkoitetaan joko kuuloaistimusta tai väliaineen muutosta, joka aikaansaa tämän aistimuksen. [2, s. 3] Ääni on myös aaltoliikettä, jonka voidaan ajatella etenevän (valon-)sädemäisesti kuten kuvassa 1. Kohdatessaan rajapinnan osa äänestä heijastuu ja osa läpäisee rajapinnan ja taittuu. Suljetuissa tiloissa seinät, katto, lattia ja muut rajapinnat aiheuttavat heijastuksia. Heijastuksen kautta kuulijalle saapunut ääni summautuu korvissa alkuperäisestä lähteestä suoraan saapuvaan ääneen. Erilaisia äänen kulkureittejä huonetilassa on havainnollistettu kuvassa 1.



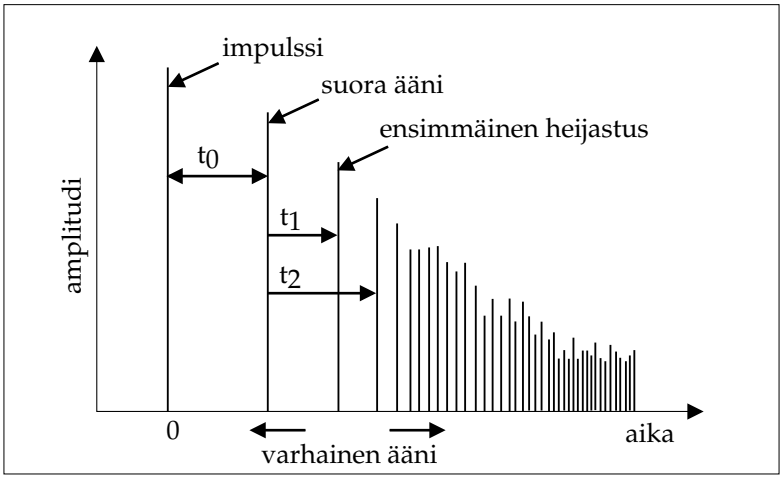
Kuva 1: Erilaisia äänen kulkureittejä äänilähteestä kuulijalle. [3, s. 21]

Saapuvien äänten aikaero vaikuttaa siihen kuullaanko niiden yhdistelmä erilaisena. Kuvassa 2 on esitetty impulssimaisen äänen (esimerkiksi laukaus, koputus tai taputus) ja sen heijastusten saapuminen kuulijalle. Kuvaaja voidaan ajatella huonetilan, äänilähteen ja kuuntelupisteen muodostaman järjestelmän vasteeksi impulssille eli impulssivasteeksi.

2.2 Jälkikaiunta

Kun kaksi samanlaista ääntä saapuu korviin siten, että niiden aikaero on yli 30 – 40 millisekuntia, kuullaan jälkimmäinen ääni kaikuna. Mikäli äänet ovat aikatasossa lähempänä toisiaan, kuullaan ne yhtenä äänenä. Jos huoneen rajapinnoista heijastuneita ääniä saapuu useita, kuullaan nämä viivästyneet äänet kaiuntana. [3, s. 169]

Huonetilassa tapahtuva äänen kuuleminen voidaan jakaa kolmeen osaan. Ensimmäiseksi kuullaan suora ääni. Viiveen jälkeen saapuvat ensiheijasteet eli huoneen rajapinnoista heijastuvat ääniaallot. Kolmannessa vaiheessa kuullaan jo useita kertoja heijastuneita aaltoja. Nämä kolme yhdessä muodostavat niin sanotun jälkikaiunnan, joka on kullekin tilalle ominainen. [4]



Kuva 2: Esimerkki impulssimaisen äänen saapumisesta kuulijalle huonetilassa. [3, s. 21]

Jälkikaiunta voidaan määritellä eri tavoin. Lahden mukaan jälkikaiunta on vaimenemistapahtuma huoneessa, kun äänilähde sammutetaan. Toisaalta jälkikaiunta voidaan määritellä äänilähteen, huonetilan ja tarkastelupisteen sijainnin yhdessä muodostaman järjestelmän impulssivasteen verhokäyränä. [5, s. 46]

2.3 Jälkikaiunta-aika

Tärkeä mitta huonekaiulle ja siten tilan akustisille ominaisuuksille on jälkikaiunta-aika. Jälkikaiunta-aika on lukuarvo, joka kertoo, kuinka kauan tilaan päästetty ääni kaikuu. Täsmällisemmin jälkikaiunta-aika määritellään aikaväliksi, joka kuluu äänilähteen sammuttamisesta siihen kun äänienergia on vaimentunut miljoonasosaan alkuperäisestä eli 60 dB. [5, s. 46]

Seuraavassa luvussa käsitellään yleisesti keinotekoista kaiuntaa ja erityisesti sen tuottamista suodattimilla.

3 Keinotekoinen kaiunta

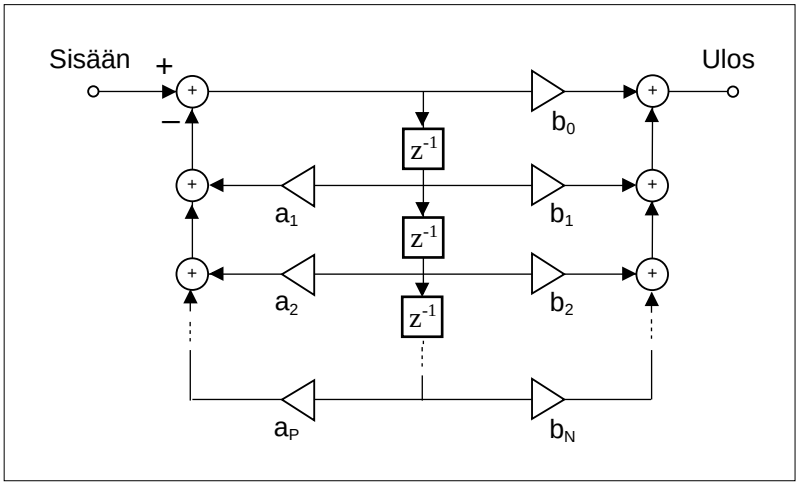
3.1 Yleistä

Keinotekoisen kaiunnan kuten muunkin virtuaaliakustiikan perusidea on, että "kuulijan korvakäytävien suulle tuotetaan samat signaalit kuin syntyisi, jos vastaava todellinen äänilähde todellisessa tilassa synnyttäisi ne." [3, s. 174] Kuuloaistimuksen kannalta on siis yhdentekevää missä ja miten äänisignaalit ovat alun perin syntyneet.

Sekä analoginen että digitaalinen äänenkäsittely on äänisignaalien muokkaamista erilaisilla suodattimilla. Suodatintekniikat voidaan jakaa kahteen päätyyppiin. Äärettömän impulssivasteen (Infinite Impulse Response, IIR) suodatin esitellään lyhyesti seuraavassa ja äärellisen impulssivasteen (Finite Impulse Response, FIR) suodatin luvussa 3.3.

3.2 Keinotekoinen kaiunta IIR-suodattimilla

IIR-suodattimet eli äärettömän impulssivasteen suodattimet perustuvat rekursioon eli takaisinkytkentään ja niiden siirtofunktiossa on napoja. Parametrisoitava keinotekoinen kaiunta on IIR-suodattimilla tyypillisesti toteutettu kampasuodattimella ja all-pass-suodattimella. Kuvassa 3 on IIR-suodattimen lohkokaavio.



Kuva 3: Lohkokaavio IIR-suodattimesta. [3, s. 42]

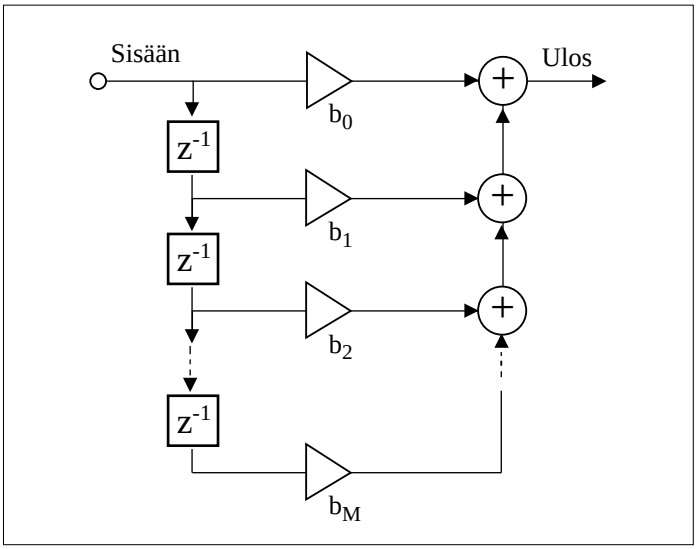
Kampasuodatin on suodatin, jonka taajuusvasteessa on tasavälein nollakohtia. Aikatasossa tarkasteltuna kampasuodattimella suodattaminen lisää signaaliin sen viivästettyjä kopioita eli kaiun. Kampasuodattimella voidaan toteuttaa esimerkiksi liukuvan keskiarvon suodatin. All-pass -suodatin on suodatin, jonka vasteen amplitudi on kaikilla taajuuksilla vakio, mutta jonka vaihevaste on mielivaltainen.[7][8]

IIR-tekniikalla on yksinkertaista luoda jyrkkävasteisia suodattimia, mutta ne ovat takaisinkytkennän takia luonteeltaan epästabiileja. Reijnen, Sonke ja de Vries ovat

käyneet läpi IIR-tekniikan käyttökelpoisuutta. Heidän mukaansa ongelma on erityisesti se, että on vain hyvin rajallinen joukko stabiileja siirtofunktioita, jotka ylipäänsä ovat toteutettavissa akustisilla tai elektronisilla takaisinkytkentäratkaisuilla. [9]

3.3 Keinotekoinen kaiunta FIR-suodattimilla

FIR-suodattimet ovat yksinkertaisia suunnitella ja stabiileja, mutta laskennallisesti raskaita. FIR-suodatuksen matemaattinen malli on konvoluutio. [10] Konvoluutio käsitellään tarkemmin luvussa 3.4.3. Kuvassa 4 on FIR-suodattimen lohkokaavio.



Kuva 4: FIR-suodattimen lohkokaavio. [3, s. 42]

3.3.1 Tilan jälkikaiunnan tallentaminen

Huoneen jälkikaiunnan tallentamiseen on kehitetty useita tekniikoita. Useat tekniikat ovat monikanavaisia, mutta yksinkertaisimmillaan vaste voidaan tallentaa yhdellä mikrofonilla.

Hyvä approksimaatio saadaan, kun huoneessa toistetaan aika-alueessa hyvin kapea piikki, likimain Diracin pulssi, ja mitataan vaste kaukokentässä. Diracin pulssin approksimaation sijaan voidaan toistaa pidempi laajakaistainen signaali ja dekonvoloida nauhoitettu vaste tällä tunnetulla signaalilla. Tulos on huoneen siirtofunktio eli impulssivaste. Yleisesti käytetty laajakaistainen signaali on “sine-sweep” eli halutun taajuusalueen yli pyyhkäistävä siniaalto. Kolmas käytetty tapa on tallentaa valkoista kohinaa huoneessa samanaikaisesti äänentoistolaitteen lähikentässä ja kaukokentässä. Impulssivaste saadaan tallenteiden ristikorrelaationa. [5, s. 42]

Kun oletetaan mitattu signaali tunnetun herätesignaalin ja siirtofunktion konvoluutioksi, voidaan siirtofunktio tietyissä tapauksissa – ja silloinkin vain likimäärin

– laskea dekonvoloimalla mittaustulos tällä tunnetulla “oikealla” signaalilla. Tässä dekonvoluutio on siis lähi- ja kaukokenttäsignaalin osamäärä taajuustasossa. [5, s. 95]

3.4 Jälkikaiunnan lisääminen signaaliin

3.4.1 Yleistä

Tallennettu jälkikaiunta lisätään signaaliin konvoloimalla. Tällöin nauhoitetulla jälkikaiunnalla konvoloitu impulssi toistuu lineaarisissa ja aikainvarianteissa olosuhteissa (linear time-invariant, LTI) samanlaisena kaiuttomassa huoneessa kuin impulssi mitatussa huoneessa. Konvoluution käytännön toteutuksia on useita. Diskreetissä aikatasossa tapahtuva suora konvoluutio tulee suodatinpituuden kasvaessa laskennallisesti raskaaksi hyvin nopeasti. Sen asymptoottinen kompleksisuus on $O(N^2)$. Suorittimien laskentatehon alati kasvaessa raja sille, miten pitkä suodatin voidaan vielä laskea reaaliajassa, tietysti siirtyy eteenpäin. Kuitenkin verrattuna kehittyneempiin algoritmeihin suora konvoluutio tuhlaa suorittimen laskenta-aikaa.

3.4.2 Reaaliaikaisuus

Naiivi määritelmä reaaliaikaisuudelle on, että jokin asia tapahtuu tässä ja nyt, välittömästi. Todellisissa järjestelmissä on kuitenkin aina hitautta. Digitaalisessa järjestelmässä tämä on ilmeistä, koska suorittimen rekisterit tallettavat signaalinäytteen äärellisen mittaiseksi ajaksi, joka riippuu suorittimen kellotaajuudesta. Realistisen digitaalisen signaalinkäsittelyn näkökulmasta reaaliaikaisuus täytyykin määritellä toisin. Käytännön tarpeisiin riittävän reaaliaikaisessa järjestelmässä voi olla useiden kymmenien näytteiden viive. Kirjallisuudessa reaaliaikaisuus-termiä käytetään paljon sitä kuitenkaan määrittelemättä.

Rossing [2, s. 654] määrittelee reaaliaikaisuuden tietokonemusiikin yhteydessä laitteiston kykynä toimia siten, että se soittimena vastaa analogista vastinettaan. Esimerkiksi sähköpianon toimintaa ei voida pitää riittävän reaaliaikaisena, mikäli koskettimen painalluksesta äänen kuulemiseen kuluva aika on niin suuri, että se häiritsee pianistia.

Toinen hyvin yksinkertainen mutta ehdoton vaatimus saadaan signaalin jatkuvuudesta. Reaaliaikaisessa signaalinkäsittelyssä sekä tuleva että lähtevä signaali on tasainen datavirta. Puskurit ovat lyhyet sillä niiden aiheuttama viive halutaan yleensä minimoida. Tällöin laitteen ulostulon puskurissa tulee aina olla dataa, sillä lopulta kaiuttimelle ja kuulijalle tulevan signaalin tulee olla keskeytymätön. Hetkellinenkin puskurin tyhjeneminen aiheuttaa ei-toivotun ja häiritsevän naksahduksen ääneen. [6]

3.4.3 Konvoluutio diskreetissä aikatasossa

Lineaarisille ja aikainvarianteille systeemeille diskreetti aikatason konvoluutio määritellään [3, s. 33] seuraavasti.

$$y[n] = x[n] * h[n] = \sum_{k=0}^{N-1} h[k] \cdot x[n-k], \tag{1}$$

jossa

y	on konvoloitu signaali diskreetissä aikatasossa
x	on konvoloitava signaali diskreetissä aikatasossa
h	on suodatinsignaali jolla konvoloidaan
N	on signaalin ja suodattimen pituuksien summa
k	on summauksen indeksi

Edellä esitetyllä algoritmillä voidaan laskea konvoluutiokaikua tarvittaessa yksittäinen näyte kerrallaan. Tämä on tavoiteltavaa kun halutaan mahdollisimman reaaliaikainen toteutus. Tällöin kuitenkin tuhlataan laskenta-aikaa. Pitkille signaaleille (pituuden merkitykseen palataan luvussa 7) on tehokkaampaa laskea konvoluutio taajuustasossa.

3.4.4 Konvoluutio taajuustasossa

Konvoluution laskemiseksi taajuustasossa täytyy sekä signaali että suodatin ensin muuntaa. Muunnosta kutsutaan Fourier-analyysiksi. Diskreetti Fourier-analyysi määritellään kirjallisuudessa [2, s. 640] seuraavasti:

$$X(k) = \mathcal{F}_d\{x(n)\} = \sum_{n=0}^{N-1} x(n) e^{-jk(\frac{2\pi}{N})n} \tag{2}$$

Vastaavasti lopputulos on yleensä muunnettava takaisin aikatasoon. Tätä operaatiota kutsutaan Fourier-synteesiksi. Diskreetti Fourier-synteesi määritellään [2, s. 641] seuraavasti:

$$x(n) = \mathcal{F}_d^{-1}\{X(k)\} = \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{jk(\frac{2\pi}{N})n} \tag{3}$$

Aikatason konvoluutio muuttuu Fourier-synteesissä taajuustason kertolaskuksi seuraavasti [3, s. 35]:

$$\mathcal{F}_d\{x(n) * y(n)\} = X(k) \cdot Y(k) \tag{4}$$

Nyt aikatason konvoluutio voidaan suorittaa [3, s. 35] taajuustasossa kokonaisuudessaan seuraavasti:

$$x(n) * y(n) = \mathcal{F}_d^{-1}\{X(k) * Y(k)\} = \mathcal{F}_d^{-1}\{\mathcal{F}_d\{x(n)\} \cdot \mathcal{F}_d\{y(n)\}\} \tag{5}$$

Taajuustasossa suoritettava kertolasku tehdään siis termeittäin.

4 Jälkikaiunnan lisääminen reaaliaikaisesti

4.1 Lyhytaikainen Fourier-muunnos

Edellä esitetty konvoluutioalgoritmi on ongelmallinen, koska N-mittaisen impulssivasteen ja reaaliaikasignaalin konvoluution laskemiseksi tulee reaaliaikasignaalia tallentaa puskurimuistiin käytännössä N näytettä ennen kuin laskutoimitusta voidaan suorittaa. Kahden sekunnin jälkikaiunnan lisääminen aiheuttaa siis vähintään kahden sekunnin viiveen suodattimen ulostulossa.

Smith [11] määrittelee lyhytaikaisen Fourier-muunnoksen, (short-time Fourier transform, STFT) diskreetissä aika-alueessa seuraavasti:

$$X_m(\omega) = \sum_{n=-\infty}^{\infty} x(n)w(n-mR)e^{-j\omega n} = \mathrm{DTFT}_\omega(x \cdot \textsc{Shift}_{mR}(w)) \tag{6}$$

jossa

$X_{(m)}$	on Fourier-muunnettu taajuuskomponentti taajuudella ω.
m	on indeksi joka kertoo monennettako ikkunaa käsitellään
ω	on kulmataajuus
$x(n)$	on muunnettava signaali diskreetissä aikatasossa
$w(n)$	on ikkunafunktio diskreetissä aikatasossa
R	on ikkunafunktion pituus, tässä vakio
e	on Neperin luku
j	on imaginääriyksikkö
$\textsc{Shift}_{mR}$	kuvaa muunnettavan ikkunan siirtymistä aika-alueessa m·R verran

Ikkunointifunktio w tulee valita siten, että se täyttää kriteerin

$$\sum_{m=-\infty}^{\infty} w(n-mR) = 1, \;\; \forall n \in \mathbb{Z}. \tag{7}$$

Ikkunointi ei tietenkään saa vaikuttaa lopputulokseen, eli ikkunoidun signaalin summan tulee olla ikkunoimaton signaali. Yksinkertaisimmillaan ikkunafunktio voi olla muodoltaan suorakaideaalto.

Tällöin yksittäisten muunnosten summaksi tulee alkuperäinen DTFT täydellisenä:

$$\sum_{m=-\infty}^{\infty} X_m(\omega) = \mathrm{DTFT}_\omega(x) = X(\omega) \tag{8}$$

Ikkunoinnin avulla konvoluutiokin voidaan laskea paloittain. Kun kumpikin konvoloitavista signaaleista on viipaloitu ikkunafunktiolla ja viipale muunnettu taajuustasoon, voidaan konvoluutio laskea tälle viipaleelle erikseen. Käänteismuunnos tulee nyt lisätä oikealle paikalleen huomioiden, että konvoluutiotulos on pituudeltaan kaksinkertainen alkuperäiseen ikkunaan nähden. Lopullista tulosta laskettaes-

sa eri viipaleiden konvoluutiotulokset tulee asetella aikatasossa oikeille paikoilleen, jolloin ne tulevat väistämättä osittain päällekkäin. Päällekkäiset osat voidaan kuitenkin yksinkertaisesti laskea termeittäin yhteen. Seuraavassa alaluvussa esitellään tätä tekniikkaa tarkemmin.

4.2 Viiveetön konvoluutioalgoritmi

Gardner on tutkinut STFT-algoritmien soveltuvuutta reaaliaikasovelluksiin ja esittää ratkaisuksi aikatason ja taajuustason konvoluution hybridiä. Gardnerin moniajomallissa aikatason konvoluution viiveettömyyttä käytetään hyväksi ostamaan aikaa radix-2 FFT:llä suoritettaville lohkoille. [12]

Konvoloitavat signaalit jaetaan STFT:tä varten N näytteen mittaisiin palasiin. N tulisi olla niin suuri, että konvoluution käytännön toteutus on tehokkaampi kuin suora aikatason konvoluutio. Gardner on esittänyt perinteisille DSP-prosessoritoteutuksille arvoa 32 tai 64. Valinta määrää suoraan myös minimiviiveen, mikäli käytetään vain FFT:tä. Ongelma voidaan kuitenkin kiertää hybriditoteutuksella, jossa ensimmäiset alkiot lasketaankin aikatasossa. Valittaessa N:lle arvoa tulee huomata, että se suodattimen pituuden kanssa määrää myös sen, kuinka monelle laskentayksikölle suoritus voidaan rinnakkaistaa.

Käytännön tilanteessa varsinainen signaali on päättymätön ja satunnainen datavirta, jota luetaan äänikortin näytteenottopuskureista. Puskurointi aiheuttaa välittömästi myös viivettä, joka tulee huomioida myös algoritmien valinnoissa. Teoreettisen viiveen pienentäminen ei aina ole järkevää, koska se yleensä tehdään jonkin muun parametrin kustannuksella. Impulssivaste sen sijaan on tunnettu, ja se voidaan muuntaa valmiiksi taajuustasoon ennen algoritmin käynnistämistä.

Edellä kuvattu algoritmi on käytännöllinen vain, jos Fourier-analyysi ja -synteesi lasketaan nopeasti. Seuraavassa alaluvussa on kuvattu eräs tapa laskea diskreettiaikainen Fourier-muunnos hyvin nopeasti.

4.3 Fourier-muunnoksen laskeminen nopeasti

Edellä esitetty diskreettiaikainen Fourier-muunnos on teoreettiselta tehokkuudeltaan $O(N^2)$. Tämä tekee algoritmista epäkäytännöllisen ja raskaan. [13] Yleisesti Fourier-muunnos lasketaan tekniikalla, joka parhaiten tunnetaan nimellä “Radix-2 Decimation In Time Fast Fourier Transform” (radix-2 DIT FFT). Amerikkalaiset matemaatikot James Cooley ja John Tukey julkaisivat algoritmin vuonna 1964 [12]. Sen tärkeä seuraus oli, että useat signaalinkäsittelyn ongelmat olivat nyt ratkaistavissa tehokkaimmin taajuustasoon siirrettynä.

FFT-algoritmejä on olemassa toki muitakin. Itse asiassa käsite FFT sisältää kaikki nopeat Fourier-muunnokset, ja nopea taas tarkoittaa mitä tahansa muutosta edellisessä luvussa esitettyyn naiiviin DTFT:hen, jolla sen kompleksisuutta saadaan vähennettyä. Cooleyn ja Tukeyn versio on kuitenkin yksinkertaisin.

Radix-2 algoritmin eräs olennainen rajoite on, että muunnettavan aineiston pituuden tulee olla $2^x,\ \forall x \in \mathbb{N}^*$. Tämä on yleensä helposti saavutettavissa yksinkertaisesti lisäämällä syötteeseen seuraavasta kahden monikerrasta puuttuva määrä tyhjiä (arvoltaan nolla) alkioita. Kuitenkin tästä seuraa että mitä pidemmälle signaalille muunnos lasketaan sitä suurempi määrä nolla-alkioita joudutaan keskimäärin lisäämään. Algoritmin on kuitenkin käytävä läpi myös nämä lisätyt alkiot.

Radix-2 DIT FFT on asymptoottiselta suoritusajaltaan $O(Nlog_2(N))$. Tämä perustuu alkuperäisen muunnoksen puolittamiseen. Mikäli aineiston pituus on $2^x,\ \forall x \in \mathbb{N}^*$, voidaan jakaminen suorittaa loppuun asti.

Seuraavassa esitelty radix-2 DIT FFT -algoritmi on Arndtin [13, s.407] kirjan mukainen.

Parillisille n, Fourier-muunnettu alkio k on:

$$\mathcal{F}[a]_k = \sum_{x=0}^{n-1} a_x z^{xk} = \sum_{x=0}^{n/2-1} a_{2x} z^{2xk} + \sum_{x=0}^{n/2-1} a_{2x+1} z^{(2x+1)k} \tag{9}$$

$$= \sum_{x=0}^{n/2-1} a_{2x} z^{2xk} + z^k \sum_{x=0}^{n/2-1} a_{2x+1} z^{2xk} \tag{10}$$

jossa

- z on $e^{\sigma i 2\pi/n}$
- σ on ± 1, muunnoksen suunta
- k on muunnetun alkion indeksi, $k \in \{0, 1, 2, ..., n-1\}$

$$\sum_{x=0}^{n-1} a_x z^{j+\delta n/2} = \sum_{x=0}^{n/2-1} a_x^{(even)} z^{2x(j+\delta n/2)} + z^{j+\delta n/2} \sum_{x=0}^{n/2-1} a_x^{(odd)} z^{2x(j+\delta n/2)} \tag{11}$$

Kun $\delta = 0$, saadaan:

$$= \sum_{x=0}^{n/2-1} a_x^{(even)} z^{2xj} + z^j \sum_{x=0}^{n/2-1} a_x^{(odd)} z^{2xj} \tag{12}$$

Kun $\delta = 1$, saadaan:

$$= \sum_{x=0}^{n/2-1} a_x^{(even)} z^{2xj} - z^j \sum_{x=0}^{n/2-1} a_x^{(odd)} z^{2xj} \tag{13}$$

Kaavat voidaan kirjoittaa toiseen muotoon, jolloin tuloksena on radix-2 DIT FFT -askel:

$$\mathcal{F}[a]^{(left)} \underset{=}{}^{n/2} \mathcal{F}[a^{(even)}] + \mathcal{S}^{1/2} \mathcal{F}[a^{(odd)}] \tag{14}$$

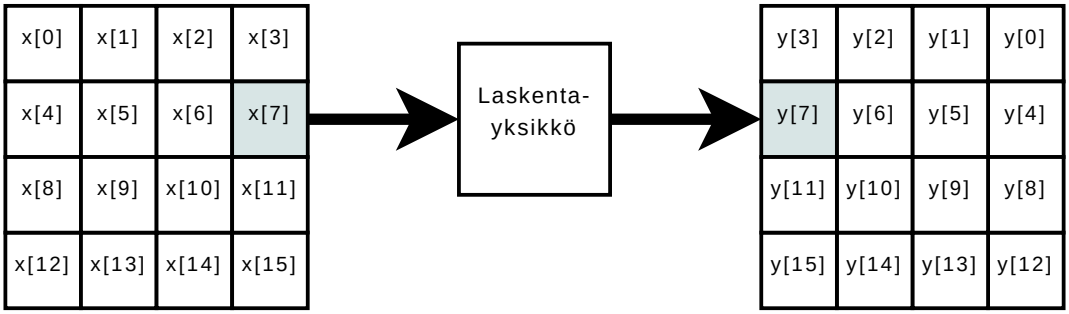
$$\mathcal{F}[a]^{(right)} \overset{n/2}{=} \mathcal{F}[a^{(even)}] - \mathcal{S}^{1/2}\mathcal{F}[a^{(odd)}] \tag{15}$$

Tämä rekursiivinen esitysmuoto on kenties elegantti, mutta käytännön toteutuksissa käytetään algoritmin iteratiivista muunnosta [13].

5 Algoritmin muokkaaminen rinnakkaislaskettavaksi

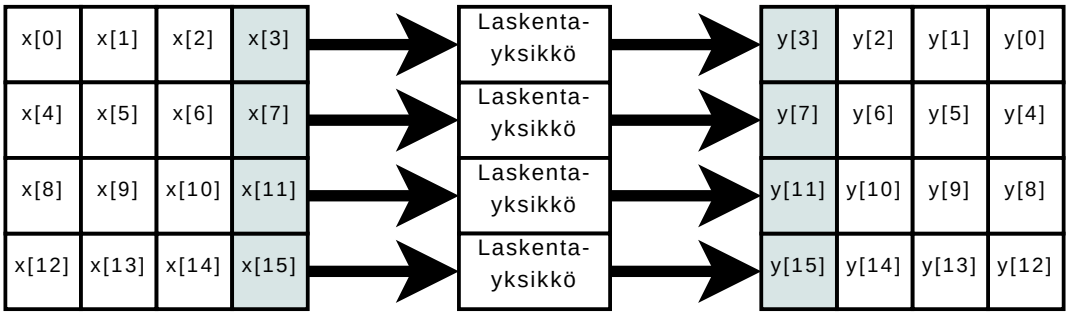
5.1 Yleistä

Perinteisesti signaalinkäsittelyä on tehty suorittimilla, joissa on ollut yksittäinen laskentayksikkö. Perusidea on ollut rakentaa mahdollisimman korkealla kellotaajuudella toimiva suoritin, joka on pystynyt käsittelemään alkioita mahdollisimman nopeasti. Tällaisen “singlecore”-suorittimen toimintaperiaate on hahmoteltuna kuvaan 5.



Kuva 5: Yksiytimisen suorittimen toiminta.

Kellotaajuutta ei voida kuitenkaan nostaa rajatta ilman seurauksia. CMOS-tekniikalla sähköenergian kulutus ja toisaalta piirin lämmöntuotto kasvaa taajuutta nostettaessa. Vaihtoehtoinen tapa on jakaa käsiteltävä aineisto osiin ja suorittaa laskenta useammalla mutta hitaammalla yksiköllä. Tällaisen suorittimen toimintaperiaate on hahmoteltuna kuvaan 6.

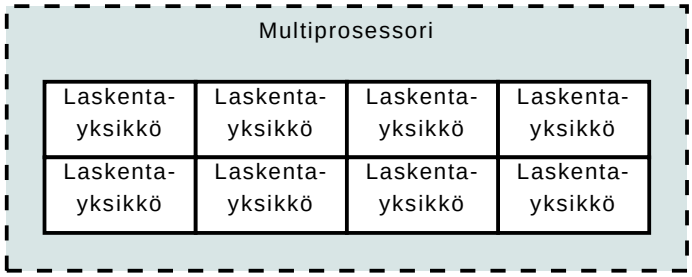


Kuva 6: Moniytimisen suorittimen toiminta.

Moniytimisen signaaliprosessorin toiminta on optimoitu SIMD-tekniikalle (single input – multiple data, SIMD), jossa nimensä mukaisesti yksittäinen käsky kohdistetaan samanaikaisesti useille, jopa tuhansille eri lähdeaineiston alkioille [14, s. 17]. Moderni grafiikkaprosessori sen sijaan käyttää perusidealtaan täysin vastaavaa, mutta monimutkaisempaa SPMD-tekniikkaa (single program – multiple data), jossa yksittäisen

käskyn sijaan suoritetaan kokonainen ohjelma [14, s. 17]. Optimaalinen teho saavutetaan, kun aineisto jaetaan tasaisesti laskentayksiköiden kesken siten, että aineiston koko on laskentayksiköiden määrän monikerta.

NVIDIA käyttää grafiikkakorttiensa suorittimista nimitystä "multiprocessor" – multiprosessori. Multiprosessori sisältää useita laskentayksiköitä ja jonkin verran nopeaa välimuistia. Multiprosessorin yksinkertainen periaatekuva on esitetty kuvassa 7. [16]



Kuva 7: Periaatekuva grafiikkakiihdyttimen multiprosessorista.

Seuraavassa alaluvussa kuvataan esimerkinomaisesti grafiikkakiihdyttimelle soveltuva STFT-algoritmi.

5.2 SPMD–STFT -algoritmi

Koska toteutus halutaan reaaliaikaiseksi, tulee ikkunanpituuden olla pieni, esimerkiksi 256 näytettä [1]. Input- eli sisäänmenopuskuri täytetään ikkunanpituuden osoittamalla määrällä konvoloitavaa signaalia [12]. Olettaen, että suodatin ei muutu kesken suorituksen, voidaan se Fourier-muuntaa valmiiksi ja tallettaa grafiikkakortin muistiin.

Käynnistetään niin monta laskentayksikköä kuin mitä vaaditaan suodatinsignaalin jakamiseksi ikkunanpituuden kokoisiin osiin [16]. Jokaisen laskentayksikön rekisterimuistiin kopioidaan näytesignaali ja osa suodattimesta. Kaikki laskentayksiköt suorittavat saman ohjelman, mutta eri suodattimen osalle. Laskentayksiköillä tulee olla tieto mitä kohtaa suodattimesta se käsittelee, jotta ne voivat kopioida tulokset oikeaan kohtaan.

Laskentayksikön ohjelma eli kernel laskee ensin nopean Fourier-muunnoksen signaaliviipaleelle ja laskee sitten konvoluution eli Fourier-muunnetun signaalin ja Fourier-muunnetun suodatinviipaleen tulon termeittäin. Mikäli nopeaa rekisterimuistia on riittävästi, kannattaa data ja välitulokset säilyttää siinä. Tässä kohden on huomattava, että Fourier-tasossa alun perin reaaliset signaali ja suodatin ovat kompleksisia. Muistia niiden tallentamiseksi tarvitaan siis kaksinkertainen määrä. Konvoluutiotulos on seuraavaksi Fourier-käänteismuunnettava takaisin diskreetin aikatason signaaliksi [13]. Tässä vaiheessa äärellinen sananpituus aiheuttaa pyöristysvirheen, jolloin käänteismuunnettu signaali ei välttämättä olekaan enää reaalinen. Tapauskohtaisesti tulee selvittää, voidaanko imaginääriosat yksinkertaisesti jättää huomiotta. Lopullisen tuloksen tulee kuitenkin olla reaalinen signaali.

Seuraavassa vaiheessa kopioidaan tai täsmällisemmin lisätään kultakin laskentayksiköltä saatu tulos välimuistina toimivaan grafiikkakiihdyttimen varsinaisessa muistissa sijaitsevaan rengaspuskuriin. Tämä vaihe on hieman haastava, koska eri laskentayksiköiden tulokset menevät aika-alueessa ja siten muistissakin päällekkäin. Eräs tapa järjestää asia on tehdä kopiointi kahdessa osassa, jolloin kaikki yksiköt kopioivat ensimmäisen ikkunanpituudellisen dataa ja tämän jälkeen odottavat, kunnes kaikki yksiköt ovat saaneet kopioinnin tähän vaiheeseen. Toisessa vaiheessa kopioidaan kunkin laskentayksikön tuloksen jälkimmäinen osa ja siis summataan se puskurissa jo olevan datan kanssa. Näin vältetään tilanne, jossa eri yksiköt tulisivat käsitelleeksi samaa muistialkiota samaan aikaan [16].

Kun kaikki laskentayksiköt ovat saaneet kopioinnin suoritettua, jatketaan suoritusta vielä yhdessä yksikössä. Tämä kopioi rengaspuskurista ikkunanpituuden verran "vanhinta" dataa (johon on siis kumuloitunut konvoluution koko suodattimen yli) ja nollaa sitten tuon kohdan puskurissa. Tämän jälkeen siirretään rengaspuskurin indeksiä ikkunanpituuden verran eteenpäin ja aloitetaan seuraava sykli.

5.3 Algoritmin kuvaaminen OpenCL-kielellä

Eräs ratkaisu rinnakkaislaskennan toteutukselle on OpenCL-ohjelmointirajapinta. OpenCL on johtavista elektroniikkavalmistajista koostuvan Khronos Groupin kehittämä avoin rinnakkaislaskentaa tukeva ohjelmointikieli [14]. OpenCL perustuu C-kielen versioon C99. C on variaatioineen maailman yleisimpiä ohjelmointikieliä. OpenCL kuten C:kin on suunniteltu mahdollisimman laitteistoriippumattomaksi. OpenCL-kielinen ohjelma, "kernel", voidaan kääntää vasta pääohjelman ajon aikana, jolloin eri laitekokoonpanoille ei tarvita omia valmiita binäärikäännöksiä. [15]

Parametrisoituja konvoluutiokaikuefektejä on yleisesti saatavilla, mutta ne eivät yleensä hyödynnä mahdollisuutta siirtää osa laskennasta pois keskusprosessorilta (CPU) esimerkiksi grafiikkakortille (GPU).

Tässä työssä on laitteiston osalta rajoituttu vain NVIDIA:n ja Applen laitteistoihin. Ympäristön valintaan vaikutti oleellisesti se, että työtä kirjoitettaessa Apple Mac OS 10.6 oli juuri julkaistu ja OpenCL 1.0 -rajapinta sekä tarvittavat ohjelmointityökalut olivat siihen helposti saatavilla. Vaihtoehtoisten laitteiden hankkiminen työtä varten ei ollut taloudellisesti järkevää. Käytettävissä ollut laitteisto on siten vaikuttanut joihinkin algoritmin realisoinnissa tehtyihin valintoihin.

NVIDIAn OpenCL Best Practices on kokoelma kyseisen laitevalmistajan ilmoittamia parhaita ohjelmointikäytäntöjä. Sen mukaan GPU:n ja isäntäkoneen välinen tiedonsiirto on kaikkein hitainta ja sitä suositellaankin vältettäväksi. Kohtuullisen monimutkaisenkin operaation suorittaminen hitaammalla CPU:lla saattaa muodostua nopeammaksi kuin aineiston siirtäminen kortille suoritettavaksi ja edelleen saadun tuloksen siirtäminen isäntäkoneen muistiin. [16]

Toinen pullonkaula löytyy laskentakortin oman muistin ja varsinaisten suorittimien välisestä väylästä. Huolimatta siitä, että tämä on yleensä kertaluokkia nopeampi

siirtotie kuin mitä kortin ja isäntäkoneen välinen väylä, on sen turhaa käyttämistä kuitenkin vältettävä. Nvidian ilmoittama viive muistioperaatiolle on 400 - 600 kellojaksoa. Useiden operaatioiden laskeminen toistuvastikin on siis suositeltavampaa kuin välitulosten tallentaminen muistiin, ellei lookup taulukko ole riitävän pieni mahtuakseen laskentayksikön omaan nopeaan rekisterimuistiin.

OpenCL:n lukujen käsittely erityisesti trigonometristen funktioiden tapauksessa on IEEE 754 -standardin mukaista tietyillä funktioilla. Kuitenkin nämä funktioiden suorittaminen vaatii huomattavasti enemmän kellojaksoja kuin niinsanotut "native"-funktiot, joiden taas ei tarvitse täyttää standardin vaatimuksia. Yleisesti näidenkin tarkkuus on riittävän hyvä.

Kompleksilukujen funktiot puuttuvat OpenCL 1.0:sta, vaikka ne ovatkin C99:ssä. Tätä versiota käytettäessä kompleksilukujen käsittelyyn on kirjoitettava omat toteutukset. Toisaalta OpenCL tarjoaa alkeistyyppeihin laajennokset, joiden avulla kompleksiluvut voidaan esittää float2 -tyyppisenä kahden liukuluvun vektorina. Kielen sisäänrakennetut yhteen- ja vähennyslaskuoperaatiot ovat float2:lle samat kuin mitä kompleksiluvuille tulisikin käyttää.

Alun perin ikkunan pituudeksi kaavailtiin 1024:ää tai jopa suurempaa, mutta ongelmaksi muodostui se, että käytössä olevien grafiikkakorttien nopea rekisterimuisti ei riittänyt algoritmin suorittamiseen mikäli ikkunan koko on suuri. Lopulta toimivaksi ikkunan ylärajaksi tuli 256 näytettä. Tämä on siltä osin järkevää, että reaaliaikasovelluksessa pyritäänkin minimoimaan puskureiden koko. Nopean Fouriermuunnoksen suhteellinen tehonlisäys verrattuna "hitaaseen" DTFT-muunnokseen kuitenkin heikkenee ikkunakoon pienentyessä. Gardnerin [12] mukaan käytännöllinen alaraja ikkunanpituudelle on 32 tai 64 näytettä. Tätä pienemmillä ikkunoilla nopea muunnos onkin hitaampi.

Asymptoottisen suoritusajan teoreettinen pohdiskelu, ainakin kovin yksityiskohtainen sellainen, on turhaa, koska OpenCL-kielinen ohjelma käy useiden käännösvaiheiden läpi joista jokainen pyrkii optimoimaan suorituksen laitteistolle optimaaliseksi. Lopullinen laitteiston suorittama komentosarja ja sen suoritusaika saattaa poiketa oleellisesti siitä, mikä lähdekoodin perusteella näyttäisi oikealta. Esimerkiksi yksittäisen alkeisoperaation määritelmä on moniprosessorijärjestelmässä hyvin ongelmallinen, koska yksittäinen operaatio voi kohdistua samanaikaisesti useampaan data-alkioon.

6 Yhteenveto

Digitaalisen signaalin ja huoneen impulssivasteen konvoluutio voidaan laskea aikatasossa triviaalisti. Tällöin kuitenkin tuhlataan huomattavasti laskenta-aikaa, koska algoritmi on tehoton. Perinteisellä nopealla Fourier-muunnoksella, jossa konvoluutio lasketaan koko impulssivasteen mittaiselle signaalille yhdellä kertaa, muodostuu impulssivasteen mittaisen lähdesignaalin odottamisesta ja varsinaisen laskennan suorittamisesta viive, joka tekee siitä käytännön reaaliaikasovelluksiin kelpaamattoman.

Tässä työssä on esitelty tekniikka jossa jakamalla käsiteltävät signaalit pienempiin osiin voidaan raskas laskenta jakaa useiden prosessoriytimien ja OpenCL-rajapintaa hyödyntäen jopa yleiskäyttöisten ja grafiikkaprosessorien välille, jolloin työaseman potentiaalinen laskentakapasiteetti saadaan maksimaalisesti hyödynnettyä. Luvussa 5.2 esitetyllä algoritmilla on mahdollista laskea jatkuvan signaalin ja suodattimen konvoluutiota tehokkaasti tavallisella kotitietokoneella. Laskenta voidaan hajauttaa grafiikkakiihdyttimen multiprosessoreille. Haasteen tuo reaaliaikaisuuden vaatimus, koska aineiston siirto äänikortilta grafiikkakiihdyttimelle on hidasta verrattuna muihin operaatioihin. Esitetyllä tekniikalla suurin osa laskennasta voidaan suorittaa näytönohjaimella käyttäen huomattavasti useampaa rinnakkaista prosessia kuin mitä tällä hetkellä on mahdollista yleiskäyttöisillä prosessoreilla. Mikäli väylänopeuden kasvun tai muun ratkaisun ansiosta viive äänikortin ja grafiikkakiihdyttimen välisessä tiedonsiirrossa saadaan riittävän pieneksi, voidaan tällä tekniikalla laskea merkittävästi pidempiä suodattimia ja useampikanavaista monikanavaääntä tehokkaasti kuin perinteisellä CPU-toteutuksella.

Aihealue on verrattain uusi ja erityisesti massiivisesti moniytimisen prosessorin tehokas hyödyntäminen edellyttää perusteellista lisätutkimusta niin teorian kuin käytännönkin tasolla. Selvää on, että nykyiset kaupallisesti saatavilla olevat ohjelmalliset konvoluutiokaiut eivät hyödynnä modernin PC:n laskentakapasiteettia täysimittaisesti.

Viitteet

[1] Borß, Christian. *A VST Reverberation Effect Plugin Based On Synthetic Room Impulse Responses.* Institute of Communication Acoustics, Ruhr-Universität Bochum, Bochum, Saksa, 2009.

[2] Rossing, Thomas D. & Moore, F. Richard & Wheeler, Paul A. *The Science of Sound.* 3. painos. Yhdysvallat, 2002.

[3] Karjalainen, Matti. *Kommunikaatioakustiikka.* 2. painos. Teknillinen korkeakoulu, Espoo, 2008.

[4] Karisto, Hannu & Kenttämies, Jouni & Koivumäki, Ari & Korpinen, Pertti. *Äänipää, Tilavaikutelma.* Verkkodokumentti. Viitattu 6.9.2009. Saatavissa: `http://www.aanipaa.tamk.fi/tila_2.htm#mozTocId42168`

[5] Lahti, Tapio. *Akustinen mittaustekniikka.* Teknillinen korkeakoulu, Espoo, 1995.

[6] Reinhardt, David & Maher, Robert. *A Real Time DSP Kernel for Concurrent Audio Tasks.* AES Convention Paper 4825, Audio Engineering Society, New York, Yhdysvallat, 1998.

[7] Smith, Julius O. *Physical Audio Signal Processing: Feedforward Comb Filters.* Verkkodokumentti. Center for Computer Research in Music and Acoustics (CCRMA), Department of Music, Stanford University, Yhdysvallat, 2009. Viitattu 11.11.2009. Saatavissa: `https://ccrma.stanford.edu/~jos/pasp/Feedforward_Comb_Filters.html`

[8] Smith, Julius O. *Physical Audio Signal Processing: Allpass Filters.* Verkkodokumentti. Center for Computer Research in Music and Acoustics (CCRMA), Department of Music, Stanford University, Yhdysvallat, 2009. Viitattu 11.11.2009. Saatavissa: `https://ccrma.stanford.edu/~jos/pasp/Allpass_Filters.html`

[9] Reijnen, Antwan J. & Sonke, Jan-Jakob & de Vries, Diemer. *New Developments in Electroacoustic Reverberation Technology.* AES Convention Paper 3978, Audio Engineering Society, New York, Yhdysvallat, 1995.

[10] Wen-Chieh, Lee & Chung-Han, Yang & Chi-Min, Liu & Jiun-In, Guo. *Perceptual Convolution for Reverberation.* AES Convention Paper 5992, Audio Engineering Society, New York, Yhdysvallat, 2003.

[11] Smith, Julius O. *Spectral Audio Signal Processing: Mathematical Definition of the STFT.* Verkkodokumentti. Center for Computer Research in Music and Acoustics (CCRMA), Department of Music, Stanford University, Yhdysvallat, 2009. Viitattu 27.9.2009. Saatavissa: `http://ccrma.stanford.edu/~jos/sasp/Mathematical_Definition_STFT.html`

[12] Gardner, William G. *Efficient Convolution without Input-Output Delay.* Perceptual Computing Section, MIT Media Lab, Massachusetts Institute of Technology, Yhdysvallat, 1995.

[13] Arndt, Jörg *Matters Computational.* Verkkodokumentti. Saksa, 2009. Viitattu 14.10.2009. Saatavissa: `http://www.jjj.de/fxt/#fxtbook`

[14] Khronos Group. *OpenCL Overview.* Verkkodokumentti. Viitattu 6.9.2009. Saatavissa: `http://khronos.org/opencl/`

[15] AMD. *An Introduction to OpenCL.* Verkkodokumentti. Viitattu 27.9.2009. Saatavissa: `http://ati.amd.com/technology/streamcomputing/intro_opencl.html`

[16] NVIDIA. *NVIDIA OpenCL Best Practices Guide.* Verkkodokumentti. Viitattu 31.10.2009. Saatavissa: `http://developer.download.nvidia.com/compute/cuda/2_3/opencl/docs/NVIDIA_OpenCL_BestPracticesGuide.pdf`