UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Entwicklung eines Effekt-Plugins für Musik- und Filmproduktion mit Hilfe der GPU

Bachelorarbeit

Zur Erlangung des Grades eines Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Volkmar Kobelt

Erstgutachter: Prof. Dr. Stefan Müller
(Institut für Computervisualistik, AG Computergraphik)

Zweitgutachter: Dipl.-Inform. Dominik Grüntjens
(Institut für Computervisualistik, AG Computergraphik)

Koblenz, im September 2013

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Bachelorarbeit

Volkmar Kobelt

(Matr.-Nr. 209 110 196)

Thema: **Entwicklung eines Effekt-Plugins für Musik/Filmproduktion mit Hilfe der GPU**

Vor allem im Bereich des Homerecordings, aber auch in großen Tonstudios, gewinnt Software an Einfluss und verdrängt nach und nach klassische Hardware: Neben Kosten- und Platzersparnissen ist auch die einfachere Integration in eine DAW (Digital Audio Workstation) ein triftiger Grund. Momentan bei Hardware-Herstellern beliebt ist Konvolution, mit der teurere Vintage-Hardware oder natürlicher Raumklang sehr überzeugend simuliert werden können. Entsprechende Software ist ebenfalls am Aufblühen, doch die klassische CPU ist schnell überfordert mit dem großen Rechenaufwand dieser Effekte (Hardware ist in der Regel mit weitaus spezialisierteren DSPs ausgestattet). Doch in vielen Rechnern schlummert noch ungenutztes Potential: GPUs werden mehr und mehr für nicht-graphische Anwendungen genutzt, auch für Konvolutions-Effekte bietet die parallelisierbare Struktur gute Vorraussetzungen. Diese Arbeit beschäftigt sich mit dem Konvolutions- oder Faltungshall, der besonders rechenaufwändig ist.

Ziel dieser Arbeit ist es ein Effekt-Plugin auf Basis der VST-Schnittstelle zu konzipieren und zu entwickeln, wobei die eigentliche Berechnung auf der GPU stattfindet. Das Plugin soll auf OpenCL entwickelt werden um plattformunabhängig zu sein. Angedacht ist ein Konvolutionshall, in den sich beliebige Faltungskerne laden lassen. Zunächst wird das Plugin in C++ implementiert, um die Performanz beider Implementierungen vergleichen zu können. Optional vorgesehen ist die Erweiterung des Plugins um verschiedene Parameter und Regelmöglichkeiten, um vielfältigeres Sounddesign zu ermöglichen.

Schwerpunkte dieser Arbeit sind:

1. Einarbeitung (vor allem OpenCL)
2. Konzeption (Gestaltung des Interface, Aufbau der Software)
3. Realisierung/Implementierung
4. Bewertung (Klang, Performanz)
5. Optionale Erweiterungen
6. Dokumentation der Ergebnisse

Koblenz, den 8.4.2013

- Volkmar Kobelt - - Prof. Dr. Stefan Müller-

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

Ort, Datum Unterschrift

Abstract −**Diese Arbeit stellt den Faltungshall vor, ein Verfahren, das es ermöglicht, in neutral klingenden Räumen aufgenommenes Audiomaterial mit realistisch klingendem Nachhall zu versehen. Dabei wird vor allem auf die Möglichkeit eingegangen, den Effekt auf einer GPU mit Hilfe von OpenCL zu berechnen, um die hohe Parallelisierbarkeit des Problems zu seiner Lösung zu nutzen. Ziel ist es, ein VST-Plugin entwickeln zu können, das den GPU-beschleunigten Faltungsalgorithmus in verschiedener Audiosoftware nutzbar macht.**

Abstract −**This paper explains the convolution reverb, a method that enables users to add realistic sounding reverberation to audio material that was recorded in neutral sounding rooms. In particular, the possibility of computing the effect on the GPU using OpenCL is discussed, to make use of the high concurrency of the problem. This paper aims at the development of a VST plugin that utilizes the GPU accelerated convolution algorithm, so that it can be used for audio software solutions.**

Inhaltsverzeichnis

1. Einleitung

Einer der wichtigsten und gleichzeitig komplexesten Effekte, der aus Musikproduktion und Filmton nicht wegzudenken ist, ist der Nachhall. Im Gegensatz zu anderen Effekten, die die Dynamik des Signals zugunsten von Sprachverständlichkeit oder Ähnlichem verändern, oder Effekten, die die klangliche Qualität in unnatürlicher Weise verändern oder erweitern, arbeiten Halleffekte in eine gänzlich andere Richtung: Hallarme Signale wirken auf das menschliche Gehör unnatürlich und aus den Informationen, die im Hall enthalten sind, leitet das Gehör teilweise auch die Beschaffenheit des Raums ab. Werden verschiedene Signale zusammengemischt, die in unterschiedlich klingenden Räumen aufgenommen wurden, kann das Resultat als unnatürlich wahrgenommen werden. Aus diesem Grund wird heutzutage meist in neutral klingenden Räumen aufgenommen, um dann mit Halleffekten die gewünschte räumliche Gegebenheit zu simulieren.

Während algorithmische Halleffekte durch viele Einstellmöglichkeiten erlauben, das passende Hallverhalten "maßzuschneidern", ist ihr Klang oft unnatürlich, oder kann spezielle Räume nicht zufriedenstellend nachahmen. Der Faltungshall sorgt hier für Abhilfe: Dieser Effekt erlaubt es, das Hallverhalten eines spezifischen Raumes mit Hilfe seiner Impulsantwort zu reproduzieren und so absolut natürlichen Klang der Hallfahnen zu erzielen. Bis heute ist Faltung jedoch eine sehr rechenaufwändige Technik, sodass neben Optimierung der Software auch Ausweichen auf besser geeignete Hardware-Plattformen sinnvoll erscheint. Die GPUs, die heutzutage sogar in Handhelds und Tablets arbeiten, bieten sich dank ihrer hohen Performanz für parallelisierbare Probleme an, um einen Effekt wie den Faltungshall auf ihnen auszuführen. Dabei bietet sich des Weiteren OpenCL für eine Umsetzung an, da der Code auf verschiedenen Plattformen ausführbar ist und so besonders vielen Anwendern zugänglich ist.

Diese Arbeit erklärt zunächst die physikalischen Grundlagen des Raumhalls, um dann auf mathematische Hilfsmittel einzugehen, die es ermöglichen, einen Faltungshall zu implementieren. Im Folgenden werden verschiedene APIs und Software-Komponenten vorgestellt, die für eine Implementation in Frage kommen, um dann eine solche auf der Basis des VST-Standards zu entwerfen. Abschließend wird die Leistung verschiedener Algorithmen bewertet und daraus gefolgert, inwiefern die hier vorgestellte Technik ausgereift ist und welches Verbesserungspotential vorhanden ist.

2. Physik des Raumhalls

Die Ausbreitung des Schalls im Raum, seine Absorption und Reflexion, sind vergleichbar mit der des Lichts: Wenn der Betrachter direkt in eine Lichtquelle schaut, die alleine einen geschlossenen Raum beleuchtet, sieht er das komplette Spektrum dieser, außerdem ist sie in der Regel heller als ihre Umgebung. Der Raum den sie beleuchtet ist an sich nicht sichtbar, sondern nur die reflektierten Lichtstrahlen, die von der Lichtquelle ausgehen. Die Hallfahne die ein Halleffekt erzeugt ist also

nur der indirekte Schall, der durch die Reflexion und Absorption des Raumes, sowie Beugung und Brechung entsteht. Die Licht-Analogie lässt sich bei genauerer Betrachtung nicht mehr so einfach anwenden: Während Oberflächen im Raum verschiedene Frequenzanteile des Lichts herausfiltern (sprich: eine Farbe haben), ist dieser Effekt im Bereich der Akustik weniger prägnant, da die Wellenlängen sehr viel höher sind. Vor allem hohe Frequenzen werden unterschiedlich stark von verschiedenen Materialien abgeschwächt. Beugung tritt hier auch bei weitaus größer dimensionierten Hindernissen auf und kann gerichteten Schall umlenken und zu einer kugelförmigen Ausbreitung abwandeln. Die Form des Raums spielt also eine weitaus wichtigere Rolle: Die Überlagerungen vieler "Schallstrahlen" bilden sogenannte Raummoden, Resonanzen die als besonders starke Frequenzanteile im Raum wahrgenommen werden. Je komplexer die Geometrie des Raumes ist, desto mehr unterschiedliche Raummoden treten auf. Für einen quaderförmigen Raum ist die Berechnung seines Hallverhaltens noch relativ simpel, da in der Berechnung der Moden Redundanzen genutzt werden können und es folglich schnell zu einem akkuraten Ergebnis kommt. Soll jedoch ein physikalisch korrektes Modell eines komplexen Raums wie zum Beispiel einer Kathedrale errechnet werden, wird die Komplexität extrem hoch - ein solcher Raum ist durch seine höhere Zahl an Raummoden und Resonanzen klanglich attraktiver.

Im Gegensatz zum Licht sind beim Schall auch die Laufzeiten geringer: Lichtgeschwindigkeit beträgt etwa 2,998*10^8 m/s, Schallgeschwindigkeit lediglich 342 m/s. Wird das Licht an- und ausgeschaltet, ist der Raum auf der Stelle hell oder dunkel. Wenn in einer großen Halle wiederum ein Schallimpuls auftritt, wird seine Hallfahne mit kurzer Verzögerung wahrgenommen und das Abklingen kann noch mehrere Sekunden anhalten. Bei Halleffekten wird bei der Verzögerung am Anfang vom Pre-Delay gesprochen. Tritt der Hall mit weniger als 50 ms Verzögerung nach dem eigentlichen Signal ein, lässt er sich vom Menschen nicht als gesondertes Signal erkennen und lässt das Quellsignal eher dichter oder lauter wirken, was Architekten beim Bau von Theatern oder Konzertsälen ausnutzen.

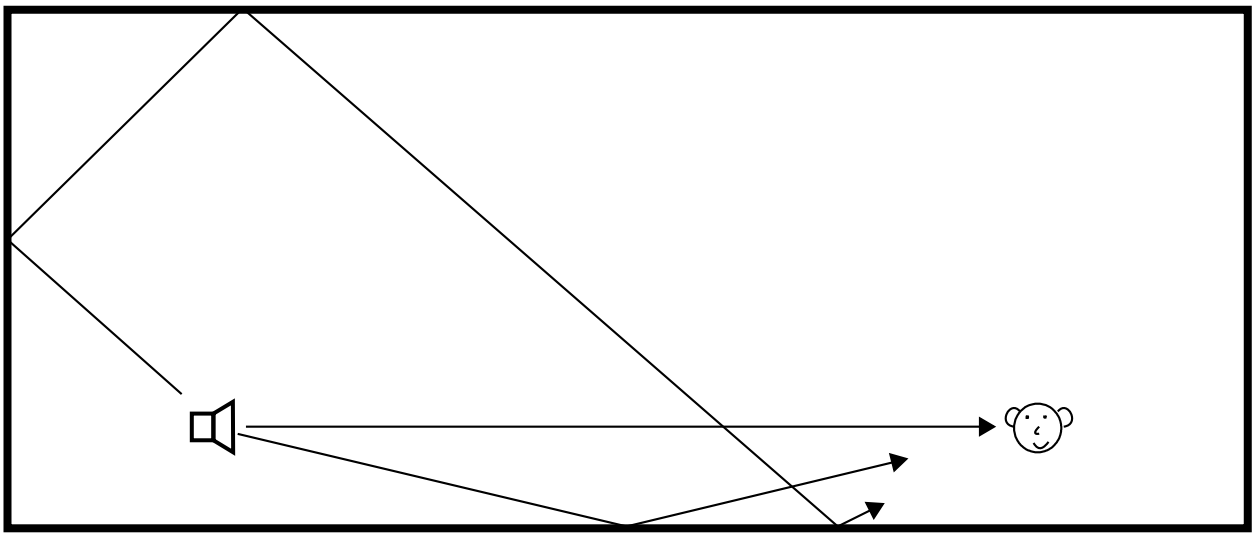

Fig. 1: Reflexionen im Raum. Nach dem direkten Schall tritt der kürzest mögliche Reflexionsvektor ein, welcher nur einmal reflektiert wird und so an lautesten ist. Der zweite Gezeigte wird mehrmals reflektiert und legt eine größere Strecke zurück. Der erste Strahl lässt sich den frühen Reflexionen zuordnen, während der Zweite Teil des Nachhalls ist.

In der Regel wird bei einer Hallfahne, also dem reinen Hall-Klang eines Raumes, zwischen frühen Reflexionen und dem Nachhall unterschieden: Kurz nach dem Pre-Delay und dem direkten Schall treten die ersten Reflexionen auf, die den Hörer über die kürzesten Wege (abgesehen vom direkten) erreichen. Diese frühen Reflexionen werden seltener reflektiert als die des Nachhalls und sind deshalb deutlich lauter. Die später eintreffenden Reflexionen des Nachhalls sind sehr viel leiser, sodass die gesamte Hallfahne in der Regel einen exponentiellen Abfall darstellt. Wenn besonders viele Reflexionen auf einen Zeitpunkt fallen, der mindestens 50 ms vom Ursprungssignal liegt, nimmt der Hörer diese als Echo war.

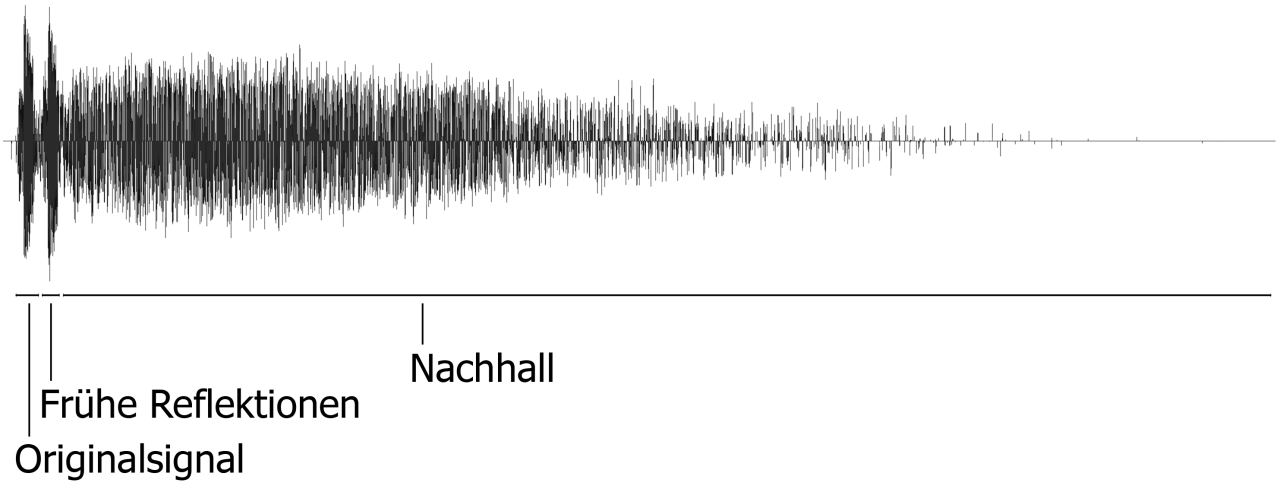

Fig. 2: Hallfahne einer Kirche. Der erste Ausschlag in der Wellenform ist der Schallimpuls, der den Raum anregt. Die zweite Spitze sind die frühen Reflexionen, die besonders deutlich sind, der eigentliche Nachhall ist sehr viel leiser, hält aber länger an. In dieser Darstellung ist die Amplitude logarithmisch skaliert, da der Nachhall sonst neben Signal und frühen Reflexionen kaum sichtbar wäre.

3. Geschichte der Halleffekte

In den Anfängen der Klangaufzeichnung und -übertragung wurde, um Material mit Raumhall anzureichern, jenes in spezielle Hallräume gespielt und aufgenommen. Diese Räume wiesen teilweise komplizierte Architekturen auf und es wurden beispielsweise große gebogene Plexiglasscheiben an der Decke aufgehangen, um die Qualität des entstehenden Halls besonders reichhaltig zu gestalten. Die Einrichtung eines solchen Raumes war jedoch sehr aufwändig und teuer, weshalb im Laufe der Zeit mehr und mehr Alternativen entwickelt wurden. Die Bedeutung von solchen Halleffekten ist bis heute immer mehr gestiegen: So werden heute sowohl Musik, als auch Filmton in weitestgehend schalltoten oder neutralen Räumen aufgenommen und dann mit Halleffekten bearbeitet, um einen natürlichen Klang zu erzielen, der dem jeweiligen Kontext entspricht.

Abb. 1: Speziell präparierter Hallraum. Die aufgehangenen Plexiglasplatten erweitern die Geometrie des Raums, sodass besonders viele Moden entstehen und komplexerer Nachhall auftritt.

Quelle: https://www.akustik.rwth-aachen.de/Diplomarbeiten/dipl_iwi_dir_gum/img_hallraum_probe

Abb. 2: Federhallmodule von Tube Amp Doctor. Die Anschlussbuchsen an der Gehäuseseite sind einfache RCA-Buchsen, wie sie an Stereoanlagen Verwendung finden - Trotzdem benötigen sie Vor- und Nachverstärkung, was meist zu deutlichem Grundrauschen führt.

Quelle: http://www.tubeampdoctor.com/images/Reverb/TAD-Reverb.jpg

3.1 Federhall

Beim Federhall wird das Ausschwingen einer Spiralfeder genutzt, um das Verhalten eines Raums nachzuahmen. Eine Spule, an der das Quellsignal anliegt regt eine Spiralfeder an, die über eine längere Strecke locker gespannt ist und mehrmals gefaltet sein kann (so dass sie eine Z-Form ergibt). Am anderen Ende der Feder ist eine zweite Spule vorhanden, in die die Feder eine Wechselspannung induziert, die dann die simulierte Hallfahne darstellt. Ein erheblicher Nachteil dieses Verfahrens ist die Störanfälligkeit, sowohl durch elektromagnetische Einstrahlungen, als auch mechanische Einwirkungen. Die Halltanks, die die Federn enthalten sind in der Regel 20-60cm lang und bis heute in Gitarrenverstärkern verbaut. Früher wurden sie auch für andere Instrumente oder generell für Verhallung von Tonspuren verwendet. Ihr eigentümlicher Klang ist nicht als sehr realistisch zu bewerten, ist aber trotzdem, oder gerade deswegen sehr beliebt: So ist er charakteristisches Merkmal für Musikstile wie Dub Raggae oder Surf Rock, aber auch in vielen Filmen der 60er und 70er Jahre eingesetzt, um psychedelische oder surreale Atmosphäre zu schaffen.

3.2 Plattenhall

Nach ähnlichem Prinzip arbeitend, aber mit weitaus natürlicher klingendem Resultat, ist der Plattenhall. Hierbei wird eine große (gewöhnlich 2x1m) Stahlplatte, die in einen Rahmen eingespannt ist, mit einem Treiber in Schwingung versetzt und diese Schwingungen werden dann von einem oder (für Stereohall) zwei Wandlern wieder in elektrische Signale umgewandelt. Später wurde das System verkleinert, indem statt der Platte eine Goldfolie eingesetzt wurde, die dann auch mit kleineren Wandlern und Treibern betrieben wurde. Durch mechanische Dämpfung der Platte lässt sich zusätzlich Klangfarbe und Nachhallzeit beeinflussen. Der Klang dieser Geräte ist weitaus näher am natürlichen Hall als der des Federhalls, aber der Klang entspricht keinem Raum, in dem normalerweise eine musikalische Darbietung zu erwarten wäre. Ebenso wie beim Federhall wird jedoch jener eigentümliche Klang bis heute geschätzt und ist deswegen in vielen modernen Software- und Hardware-Lösungen als Emulation vorhanden. Moderne Tonstudios verwenden teilweise noch große Hallplatten für ihre Produktionen.

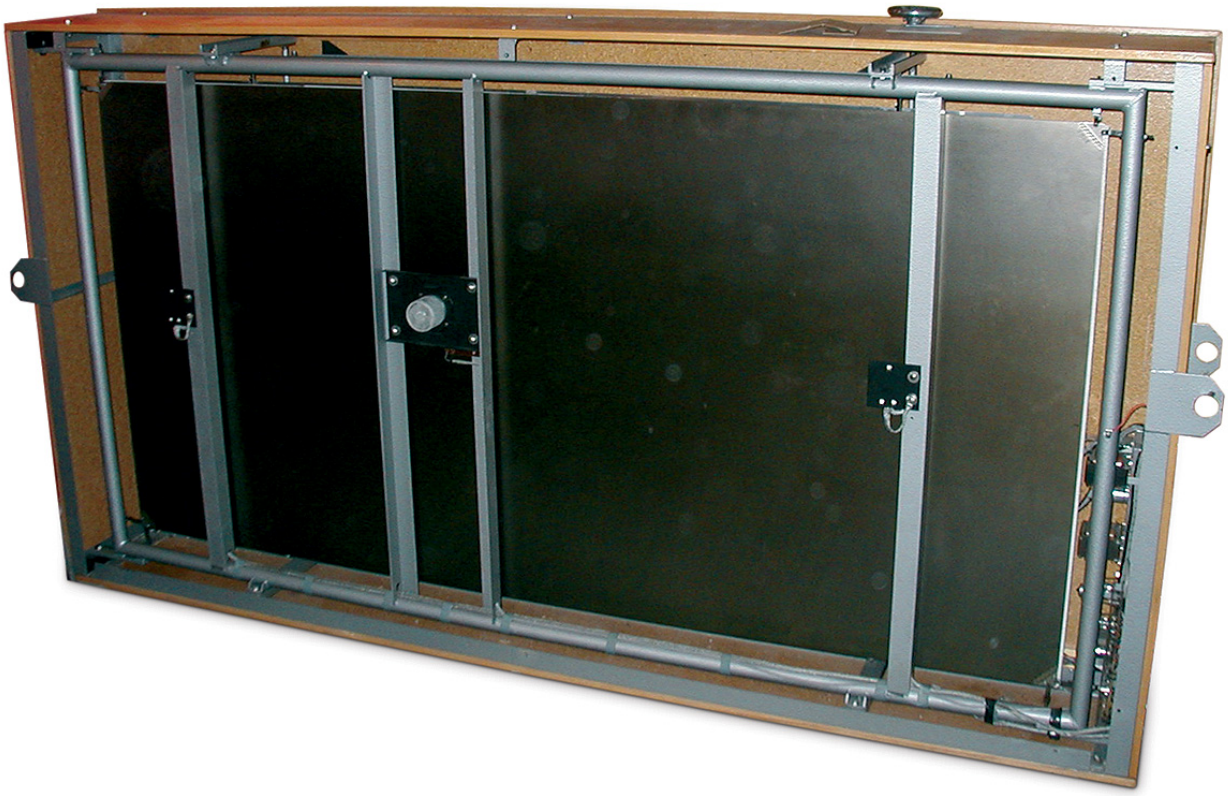

Abb. 3: Klassischer Plattenhall. In der Mitte ist der Treiber zu sehen, der die Platte anregt, links und rechts davon die Abnehmer, die den linken und rechten Ausgangskanal speisen. Durch ihre unterschiedliche Positionierung ist ein weites Stereopanorama gegeben.

Quelle: http://media.soundonsound.com/sos/feb10/images/aw_04_6.jpg

3.3 Digitaler (algorithmischer) Hall

Mit dem Aufkommen von digitaler Signalverarbeitung wurden auch Hall-Algorithmen entwickelt. Die meisten nutzen kaskadierte Allpass-, Bandpass-Filter oder rückgekoppelte Delays, um die verschiedenen Reflexionsstufen zu simulieren und Tiefpassfilter, um die Dämpfung der hohen Frequenzen umzusetzen. Da die Abstände der einzelnen Stufen in dieser Aufstellung gleich groß sind, überlagern sich besonders viele Moden und es kommt zu einem metallischen Klang der Hallfahne. Um Abhilfe zu schaffen, können die Verzögerungslängen zufällig moduliert werden, wodurch der Klang natürlicher wird, aber der Hall bei mehreren Aufnahmen mit gleichem Quellmaterial unterschiedliche Ergebnisse liefert. Um das zu verhindern, gibt es verschiedene Ansätze; meistens werden die Delays oder Filter unterschiedlich angeordnet und dimensioniert und auf die Zufallssteuerung wird verzichtet. Die Algorithmen können sehr gut klingen und durch viele steuerbare Elemente lassen sich sehr verschiedene klangliche Resultate erzielen.

Abb. 4: Der Lexicon 224 XL, einer der ersten digitalen Halleffekte. Das Bild zeigt nur die Bedienung, die eigentliche Hardware ist in einem großen 19 Zoll-Gehäuse untergebracht. Viele der preiswerteren frühen digitalen Halleffekte wiesen künstlichen Klang oder deutliche digitale Artefakte und Grundrauschen auf. Heutzutage liefern jedoch auch preiswerte, viel kompaktere Modelle oder Softwarelösungen sehr natürliche und vielseitige Ergebnisse.

Quelle: http://medias.audiofanzine.com/images/normal/lexicon-224xl-133934.jpg

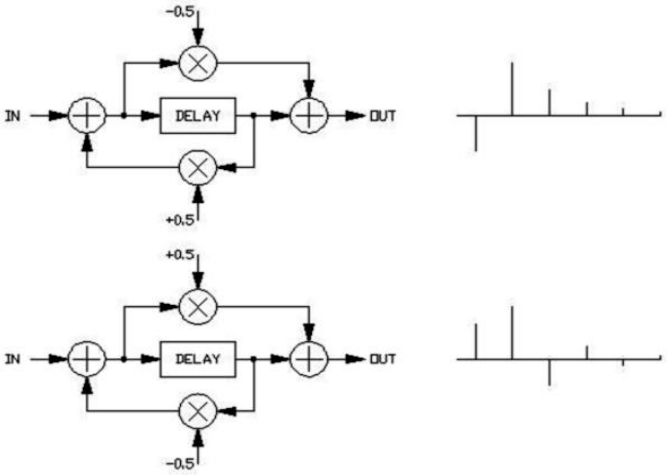

Abb. 5: Einzelnes Glied eines digitalen Halleffekts. Durch Rückkoppeln eines Delays oder Filters können einzelne Reflexionen simuliert werden, durch zahlreiche Kaskadierungen und unterschiedliche Verknüpfung können verschiedene Klangfarben erreicht werden.

Quelle: http://freeverb3.sourceforge.net/AP_RESP.jpg

4. Faltungshall

Die modernste Inkarnation der Halleffekte stellt wohl der Faltungs- oder Konvolutionshall dar. Statt der Simulation der Akustik eines natürlichen Raums, nutzt der Faltungshall die Tatsache, dass sich ein Raum als lineares zeitinvariantes System (LTS) betrachten lässt: Ein solches System reagiert auf ein eingehendes Signal mit einem ausgehenden Signal, und zwar in der Art, dass es das

ursprüngliche Signal höchstens linear verzerrt, also keine neuen Teiltöne eingefügt werden. Die Hallfahne eines Raums erfüllt diese Bedingung und dadurch lässt sich mit der Impulsantwort eines Raumes eine Faltung durchführen, um das Hallverhalten des Raums akkurat zu reproduzieren. Im nächsten Kapitel wird die mathematische Grundlage genauer erklärt. Die Impulsantwort entspricht der Reaktion des Raums auf ein Signal, das alle Frequenzanteile beinhaltet (Sinus-Sweep oder Impuls-Funktion). Im Wissen, wie der Raum auf alle Frequenzen reagiert, lässt sich auch ableiten, wie er auf einzelne Frequenzen und Teile des Spektrums reagiert - die Impulsantwort könnte also als akustischer Fingerabdruck eines Raumes bezeichnet werden. Es können auch von Geräten wie dem Feder- oder Plattenhall Impulsantworten erstellt werden, um dann ihren Klang mit einem Konvolutionshall perfekt zu imitieren. In jedem Fall ist ein realistisch klingendes Resultat gegeben, vorausgesetzt die zugrunde liegende Impulsantwort ist von guter Qualität.

Leider ist auch dieses Verfahren nicht perfekt: Die Impulsantwort beschreibt das korrekte Hallverhalten nur für jeweils einen festen Punkt im Raum für Hörer und Schallquelle. Diese entsprechen der Aufnahme, die der Impulsantwort zugrunde liegt. Soll also beispielsweise ein Orchester in einem Saal akustisch aufgeteilt und jedem Signal seine exakte Hallfahne zugeordnet werden, würde für jede gewünschte Position eine individuell angefertigte Impulsantwort benötigt werden.

Der größte Nachteil des Faltungshalls ist aber dass die Berechnungen, die für die Hallfahnen benötigt werden, sehr aufwändig sind und bis vor Kurzem nur von dedizierter Hardware, meist auf DSP-Basis, ausgeführt werden konnten. Inzwischen gibt es auch CPU-Implementierungen, jedoch arbeiten diese teilweise nur mit hohen Latenzen oder lassen sich nur mit einigen wenigen Impulsantworten betreiben. Da diese Software nicht offen ist, lässt sich auch erahnen, dass bei den Berechnungen teilweise Einsparungen vorgenommen werden, die für das menschliche Gehör verborgen bleiben oder kaum auffallen - eine hohe Optimierung ist in jedem Fall vonnöten.

5. Mathematische und technische Grundlagen

5.1 Lineare zeitinvariante Systeme

Ein lineares zeitinvariantes System (auch LTI-System für linear time-invariant, oder lineares Übertragungssystem) gibt für jedes eingehende Signal ein Signal aus, das bestimmte Bedingungen erfüllt: Es wird allenfalls linear verzerrt, also werden lediglich die Amplituden- und Phasenwerte seiner einzelnen oder aller seiner Frequenzanteile verändert. Jegliches Hinzukommen neuer Teiltöne würde diese Voraussetzung verletzen und es würde sich nicht mehr um ein lineares System handeln. Der Zusatz zeitinvariant schränkt weiterhin ein, dass eine Verschiebung des Eingangssignals auf der Zeitachse eine gleichartige zeitliche Verschiebung des Ausgangssignals bewirkt.

$$s(t) = e^{j\omega t} \rightarrow \quad s'(t) = H e^{j\omega t}$$

Die Schreibweise $e^{j\omega t}$ stellt eine sinusoidale Schwingung als komplexe Zahl dar; ihr Betrag ist dabei ihre Amplitude und ihr Winkel die Phase. Der Faktor H des linearen Systems ist ebenfalls komplex und kann so Amplitude und Phase der Ausgangsschwingung manipulieren, jedoch keine neuen Schwingungen in das Signal einmischen. Durch die Linearität des Systems gilt auch das

Superpositionsprinzip: Die Systemantwort auf die Summe zweier Signale ist gleich der Summe der Systemantworten auf die beiden einzelnen Signale.

Aus den Eigenschaften der linearen Systeme ergibt sich, dass wenn mehrere von ihnen in Reihe geschaltet werden, die Reihenfolge der Systeme irrelevant ist. Soll beispielsweise aus einem Tiefpass-Filter und einem Hochpass-Filter ein Bandpass-Effekt erzeugt werden, ist es gleichgültig, ob zuerst die hohen Frequenzen entfernt werden oder die tiefen. Generell sind alle Filter und Equalizer im idealisierten Fall als lineare Systeme zu betrachten - In der Praxis, besonders bei Hardware treten non-lineare Verzerrungen auf, wenn Übersteuerungen stattfinden, außerdem durch störende elektromagnetische Einstrahlungen, Quantisierungsfehler oder Varianzen von Bauteilen.

Auch der Nachhall eines Raumes ist im idealisierten Fall ein lineares zeitinvariantes System, denn die Reaktion des Raumes auf das eingehende Signal enthält keine neuen Harmonischen und die zeitlichen Versätze des Nachhalls treten immer nach gleichem Muster auf. Als kleines Experiment kann - zur Veranschaulichung, nicht als hinreichender wissenschaftlicher Beweis! - ein Filter oder Equalizer mit einem Halleffekt in Reihe geschaltet und die Reihenfolge vertauscht werden. Das resultierende Signal wird in beiden Fällen identisch sein, da es irrelevant ist, ob ein gefiltertes Signal den Raum anregt, oder ob die Reaktion des Raums auf ein Signal auf die gleiche Weise gefiltert wird.

Wie schon im 2. Abschnitt beschrieben, setzt sich die Hallfahne eines Raums aus Reflexionen, Brechungen und Beugungen zusammen. Es treten also vor allem zeitliche Versätze auf, die aufgrund der Absorption der Oberflächen und des Mediums im Raum (in der Regel Luft) einen beschränkten Frequenzbereich gegenüber dem Quellsignal aufweisen. Da diese aber immer im gleichen zeitlichen Muster auftreten und das eingehende Signal ansonsten nur linear verzerrt in diesen Reflexionen auftritt, kann auch jeder Raum als lineares zeitinvariantes System betrachtet werden. Diese Annahme lässt sich somit auch für jeden Halleffekt machen.

5.2 Die Impulsantwort

Die Impulsantwort (engl. impulse response, abgekürzt IR) stellt die Reaktion eines linearen Systems auf ein Signal mit allen Frequenzanteilen dar. Ein solches Signal ist der Dirac-Stoß, ein Impuls mit unendlich kurzer Länge und unendlich hoher Amplitude, aber mit der endlichen Fläche 1. Der Dirac-Stoß wird auch als Dirac-Funktion, Deltafunktion oder Einheitsimpuls bezeichnet. Für Audioanwendungen ist die Bandbreite begrenzt, also reicht eine Approximation dieser Funktion aus, um die relevanten Frequenzen abzudecken. Die entsprechende Wellenform wäre ein einzelnes Sample mit maximaler Amplitude, während die restlichen Werte auf null gesetzt sind. Wird dieses Signal in einen Feder- oder Plattenhall gespielt, resultiert bereits eine nutzbare Impulsantwort. Im Falle eines realen Raumes ist die Aufnahme leider weniger trivial: Der Puls muss hierbei mit möglichst linearen Lautsprechern abgespielt werden und mit möglichst linearen (Mess-)Mikrofonen abgenommen werden. Da Umgebungsgeräusche und Grundrauschen hier eine größere Rolle spielen, muss der Puls auch sehr laut abgespielt werden und Lautsprecher, die über das komplette hörbare Spektrum gleichmäßig den nötigen hohen Pegel liefern, sind selbst für viel Geld kaum verfügbar. Als Alternative zum Puls lässt sich auch ein Sinus-Sweep über Lautsprecher abspielen, also ein Signal, das aus einem tiefen Sinuston besteht, der über einen bestimmten Zeitraum immer höher gestimmt wird, bis er das hörbare Spektrum verlässt. Mit spezieller Software lässt sich aus diesem Signal dann eine Impulsantwort erzeugen (Dekonvolution). Diese Vorgehensweise liefert bessere Ergebnisse als das Abspielen und Aufnehmen eines Pulses, vor allem ist die Dynamik der resultierenden Impulsantwort höher. Als kostengünstige Alternative kann ein Puls anders erzeugt

werden als durch Lautsprecher, beispielsweise durch Platzenlassen eines Luftballons oder Abfeuern einer Schreckschusspistole. Ersteres ist jedoch zu leise und höchstens für kleine Räume praktikabel und beide Techniken haben den Nachteil, dass die resultierenden Pulse nicht alle Frequenzen aufweisen und die Ergebnisse recht unterschiedlich ausfallen können.

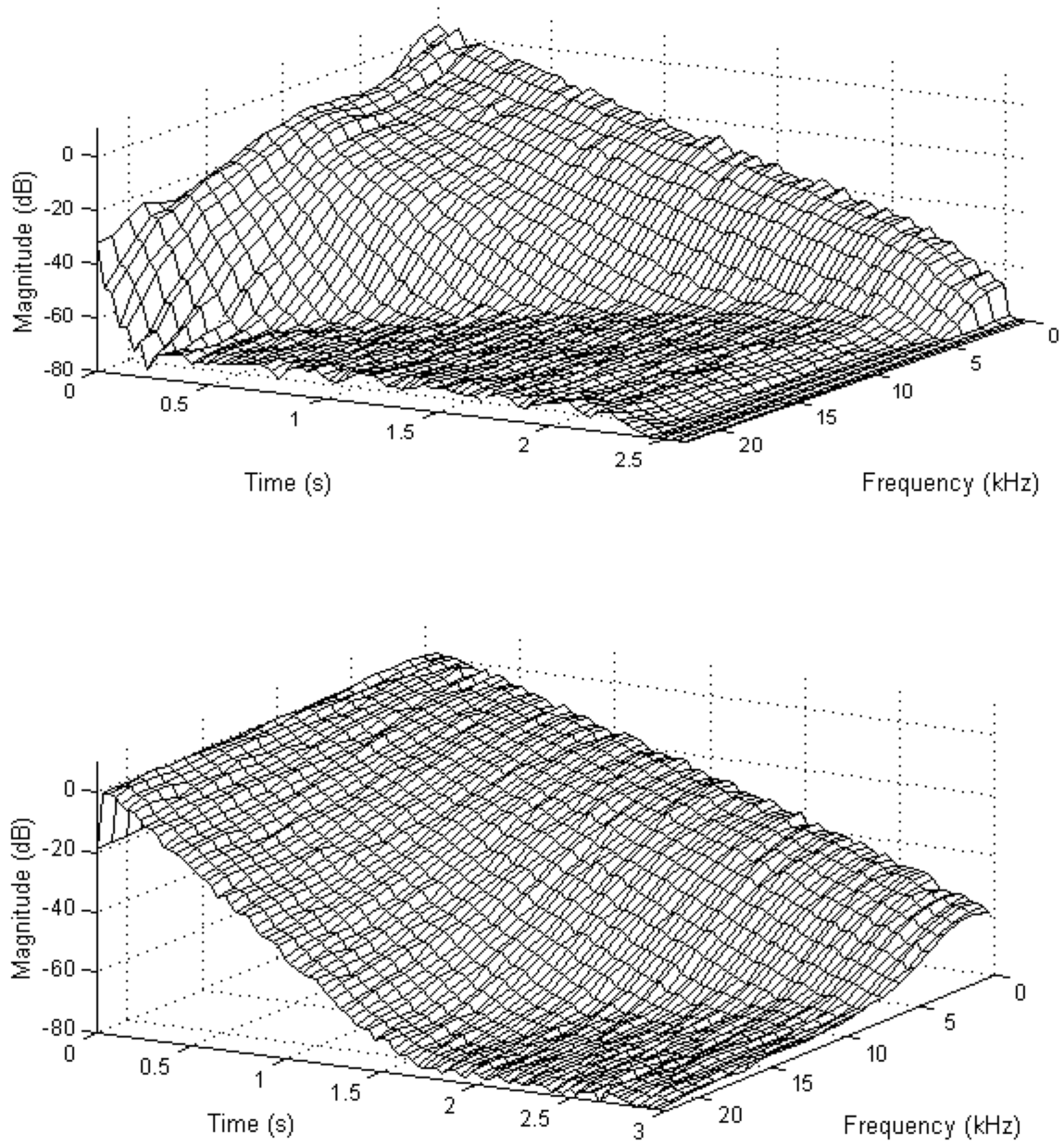

Fig. 3: EDRs (Energy Decay Relief) einer Halle (oben) und eines algorithmischen Halleffekts (unten). Durch diese Darstellung lässt sich die Entwicklung der Frequenzanteile einer Impulsantwort über ihre gesamte Dauer gut erkennen. Der natürliche Hall verliert schnell an hohen Frequenzen und weist unregelmäßige Spitzen auf. Der digitale Hall klingt sehr gleichmäßig ab und selbst die höchsten Frequenzanteile sind mehr als eine Sekunde lang messbar und unter Umständen hörbar.

Quelle: http://www.music.miami.edu/programs/Mue/Research/jfrenette/

Gerade letztere Techniken stellen auch für Amateure attraktive Möglichkeiten dar, selbst Impulsantworten zu erstellen und mit den Resultaten zu experimentieren. Im professionellen Bereich wird durch das Erstellen und Nutzen von individuellen Impulsantworten für die Nachvertonung von Filmmaterial oder Nachbearbeitung von Orchesteraufnahmen die bestmögliche Reproduktion von Raumhall ermöglicht. Abgesehen davon stehen im Internet zahlreiche kostenlose und kostenpflichtige Bibliotheken mit Impulsantworten zur Verfügung, die professionell aufgenommen wurden und zahlreiche Szenarien und räumliche Gegebenheiten abdecken. Mit diesen lassen sich bereits realistischere Hallfahnen erzeugen als mit algorithmischen Halllösungen. Für surreales Sounddesign bietet es sich zuletzt an, Klänge als Impulsantworten zu interpretieren, die ursprünglich nicht als solche gedacht waren, wie Beckenschläge oder synthetische, perkussive Klänge: Eine Faltung des Quellmaterials mit solchen Wellenformen resultiert in teilweise eindrucksvollen Klängen, die oft wie eine "Verschmelzung" von Signal und Impulsantwort wirken.

5.3 Faltung und schnelle Faltung

In den vorherigen Abschnitten wurde ein Raum mit seinem Hallverhalten als lineares zeitinvariantes System definiert und seine Impulsantwort als vollständige Beschreibung dieses Systems. Dieses Wissen reicht noch nicht aus, um einen Konvolutionshall zu implementieren. Es fehlt ist die zentrale mathematische Funktion, auf der er basiert: Die Faltung.

$$(f*g)(t)=\int_{-\infty}^{\infty} f(\tau)\,g(t-\tau)\,d\tau$$

Anschaulich ist die Faltung $(f*g)(t)$ bei Verschiebung eines Signals auf dem anderen die Überdeckung beider Signale für jeden Punkt. Der Dirac-Stoß ist das neutrale Element der Faltung, da bei seiner Verschiebung über ein anderes Signal die Abdeckung genau letzterem Signal entspricht.

$$(f*g)(n)=\sum_{k=0}^{n} f(k)\,g(n-k)$$

Auch im diskreten Fall sind die Signallängen theoretisch unendlich lang; In der Praxis wirkt sich jedoch nur ein bestimmter Teil der Signale auf das Produkt aus. Die Signallängen werden also als endlich betrachtet (in der Formel beträgt die Länge n Elemente), wobei die Voraussetzung gegeben ist, dass beide Signale gleich viele Elemente besitzen müssen. Sollte eines kürzer sein, kann es mit Nullen aufgefüllt werden, um die Zahl der Elemente des anderen zu erreichen (Zero-Padding). Das Ergebnis wird dabei nicht verfälscht.

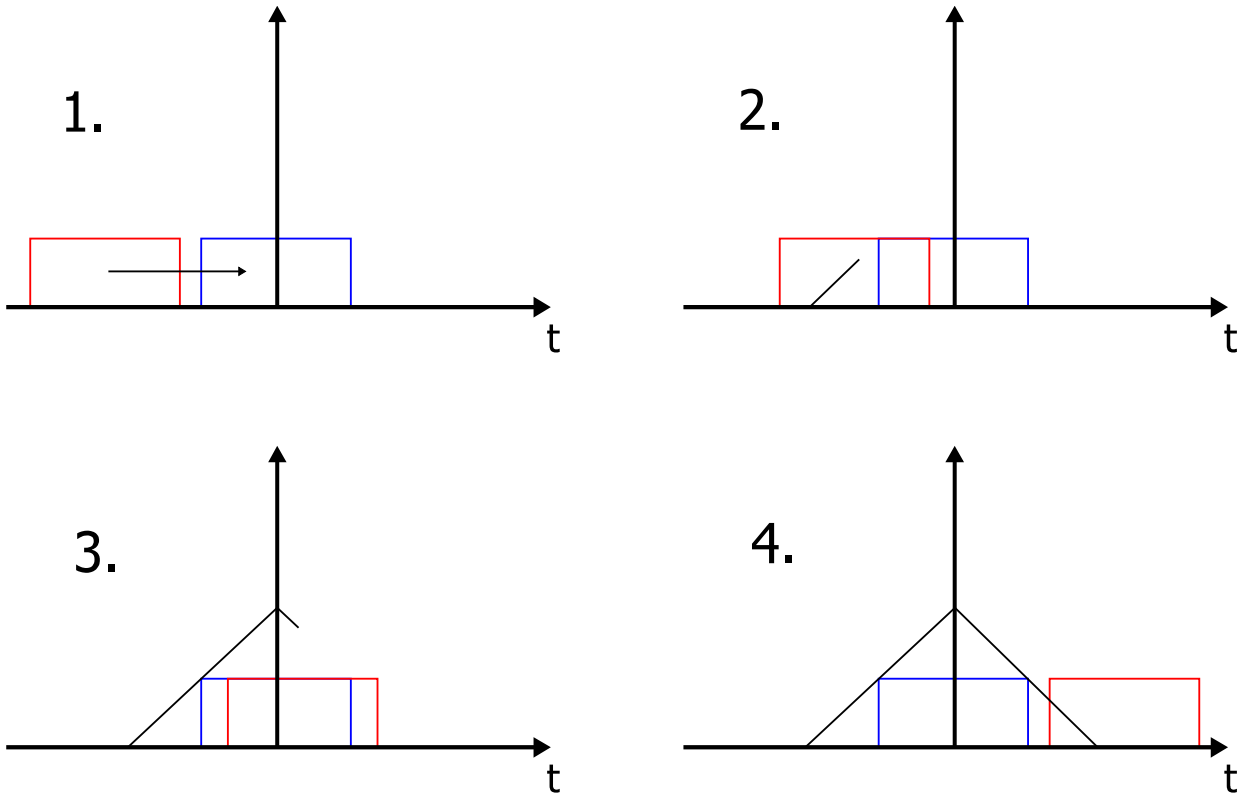

Fig. 4: Veranschaulichung der Faltung. Zwei Rechtecksignale werden gefaltet: Eines (rot) wird entlang der Zeitachse über das andere (blau) geschoben und für jeden Zeitpunkt wird die Fläche, die sie gemeinsam abdecken, aufgezeichnet. Die Fläche der Faltung entspricht nicht unbedingt der Summe der Flächeninhalte beider Eingabefunktionen, obwohl dies hier der Fall ist. Das neutrale Element ist der Dirac-Stoß, da dieser die Fläche 1 hat und unendlich kurz ist. Würde dieser über eine andere Funktion verschoben, würde lediglich die Funktion erneut aufgezeichnet werden.

Wird nun ein beliebiges Signal mit der Impulsantwort eines linearen Systems gefaltet, entspricht das resultierende Signal dem Ausgangssignal, das das System aus dem Signal erzeugt. Ein lineares zeitinvariantes System lässt sich also vollständig durch eine Impulsantwort beschreiben.

So sehr sich die Faltung als vielseitiges Werkzeug anbietet, nicht nur in der Signalverarbeitung, sondern auch um physikalische Probleme darzustellen, ist ihre praktische Anwendung durch ihren hohen Rechenaufwand in dieser Form wenig attraktiv. Der Aufwand der Faltung liegt in $O(n^2)$. Abhilfe schafft die schnelle Faltung, die auf dem Faltungstheorem beruht:

$$(f * g)(n) = F(\omega) \cdot G(\omega)$$

Laut diesem entspricht die Faltung zweier Funktionen im Zeitbereich der Multiplikation der Funktionen im Frequenzbereich. Mit Hilfe der Fouriertransformation lässt sich ein Signal im Zeitbereich in den Frequenzbereich übertragen. Das Bilden der Fouriertransformierten beider Funktionen, die komponentenweise Multiplikation ihrer Werte und anschließende Rücktransformation in den Zeitbereich liefert also die gleichen Ergebnisse wie die direkte Faltung. Um die Frequenzdarstellung eines Signals zu ermitteln, kann statt der diskreten Fouriertransformation die schnelle Fouriertransformation genutzt werden, die sich statt $O(n^2)$ in $O(n \log n)$ berechnen lässt. Ab einer bestimmten Eingabegröße ist die Berechnung von FFT (Fast

Fourier Transformation, englisch für schnelle Fouriertransformation), komponentenweiser komplexer Multiplikation und anschließender inverser FFT schneller als die naive diskrete Faltung. Für einige Anwendungen der digitalen Signalverarbeitung kann kritische Latenz dadurch entstehen, dass Signalteile gesammelt werden müssen, bis sich die Berechnung der Faltung effizient ausführen lässt. Hier wird oft der Kompromiss eingegangen, dass ein kurzer Teil des Signals direkt gefaltet und dann der nachfolgende, längere Teil mit Hilfe der schnellen Faltung effizient berechnet wird. Für den Halleffekt hat sich dies nicht als notwendig erwiesen, da die Hallfahne grundsätzlich mit einer gewissen Verzögerung auftritt und die Zeit, in der Samples gesammelt werden, noch in diesem zeitlichen Rahmen liegt.

Die Interpretation der Faltung als Multiplikation der Frequenzanteile mag auch zum Verständnis der Faltung beitragen: Die Impulsantwort hat für jede mögliche Frequenz einen komplexen Wert. Wird nun ein Signal mit der Impulsantwort gefaltet, werden lediglich die Frequenzanteile abgeschwächt oder verstärkt, je nachdem wie groß ihr Anteil in der Impulsantwort ist, sowie ihre Phasen verschoben. Damit lassen sich alle Manipulationen, die ein lineares zeitinvariantes System definieren, vollständig beschreiben.

5.4 Schnelle Fouriertransformation

Dieses Kapitel soll nur die Grundidee der schnellen Fouriertransformation beschreiben. Für verschiedene Implementationen und Verbesserungen des Grundalgorithmus bietet es sich an, Recherche zu betreiben und einschlägige Literatur zu sichten: Gängige Algorithmen sind unter anderem Cooley-Tukey und Sande-Tukey FFT, Base-N, Rader und Winograd oder die Bluestein FFT.

Zunächst ist es vonnöten, die diskrete Fouriertransformation zu verstehen:

Die diskrete Fouriertransformation ordnet einer Funktion im Zeitbereich eine spektrale Darstellung, also im Frequenzbereich zu. Im Zeitbereich ist für jeden möglichen Zeitpunkt ein konkreter Wert, eine Amplitude vorhanden. Hingegen ist im Frequenzbereich jeder möglichen Frequenz eine Amplitude und eine Phase zugeordnet. Ein einzelnes Element des Spektrums ist eine sinusoidale Wellenform und jedes beliebige Signal lässt sich theoretisch als Summe aus unendlich vielen dieser Sinusoiden darstellen. Für diskrete Signale ist die Zahl der möglichen Frequenzen begrenzt in der Form, das für n Elemente die maximale Frequenz n/2 (Nyquist-Frequenz) beträgt: Dies ist die höchste Frequenz, für die je ein Wellenberg und ein Wellental dargestellt werden können. Höhere Frequenzen würden durch Aliasing verfälscht, das Nyquist-Shannon-Abtasttheorem beschreibt diese Gesetzmäßigkeit. Konkret resultiert aus dieser Begrenzung der möglichen Frequenzen, dass sich im diskreten Bereich tatsächlich jedes mögliche Signal durch eine begrenzte Zahl an Sinusoiden darstellen lässt. Werden nacheinander diskrete und inverse diskrete Fouriertransformation auf ein Signal angewandt, wird das Signal ohne Fehler wiederhergestellt (abgesehen von arithmetischer Ungenauigkeit).

$$H_n = \sum_{k=0}^{N-1} h_k e^{2\pi i \frac{kn}{N}}$$

Die Amplitude und Phase eines Frequenzanteils sind in Betrag und Winkel des komplexen Vektors $h_k e^{2\pi i \frac{kn}{N}}$ enthalten. In der obigen Formel ist H_n der n-te Eintrag im Spektrum, N die Anzahl an

Einträgen. Hinter i verbirgt sich die imaginäre Einheit, die hier verwendete Schreibweise entspricht der Polarform für komplexe Zahlen. Sie ist in folgender Form äquivalent:

$$h \cdot e^{i*\varphi} = h \cdot (\cos\varphi + i \cdot \sin\varphi)$$

Dabei ist φ der Winkel des komplexen Vektors auf der komplexen (gaußschen) Zahlenebene; in der DFT wird der Winkel also durch k, n und N beeinflusst. Die inverse diskrete Fouriertransformation unterscheidet sich nur minimal:

$$h_n = \frac{1}{N} \sum_{k=0}^{N-1} H_k e^{-2\pi i \frac{kn}{N}}$$

Der Exponent ist hier negiert und die einzelnen resultierenden Werte im Zeitbereich h_k müssen durch die Gesamtzahl der Werte geteilt (normalisiert) werden. Somit kann ein effizienter Algorithmus mit leichten Abwandlungen für beide Richtungen genutzt werden. Ein solcher Algorithmus ist die Fast Fourier Transformation oder schnelle Fouriertransformation, die die DFT statt in $O(n^2)$ in $O(n \log n)$ berechnet.

Sei $W = e^{2\pi i/N}$, also eine komplexe Zahl.

$$H_k = \sum_{j=0}^{N-1} e^{2\pi i j k/N} h_j$$

$$= \sum_{j=0}^{N/2-1} e^{2\pi i k \frac{2j}{N}} h_{2j} + \sum_{j=0}^{N/2-1} e^{2\pi i k \frac{2j+1}{N}} h_{2j+1}$$

$$= \sum_{j=0}^{N/2-1} e^{2\pi i k \frac{j}{N/2}} h_{2j} + W^k \sum_{j=0}^{N/2-1} e^{2\pi i k \frac{j}{N/2}} h_{2j+1}$$

$$= H_k^e + W^k H_k^o$$

Diese Beweisführung wird als das Danielson-Lanczos-Lemma bezeichnet und wurde nach den beiden Wissenschaftlern benannt, die sie 1942 aufstellt haben. Jede diskrete Fouriertransformation lässt sich demnach in ihre geraden und ungeraden Summanden aufteilen, wobei aus den ungeraden eine komplexe Zahl extrahiert wird. Diese hat die Besonderheit, dass sie sich als Einheitsvektor auf der komplexen Zahlenebene betrachten lässt, dessen Ausrichtung von den unterschiedlichen Werten in k abhängt. Sie wird in der Literatur oft als "Twiddle-Faktor" bezeichnet. Nun lässt sich ein Divide-And-Conquer-Algorithmus aufbauen, indem aus den resultierenden diskreten Fouriertransformationen der Länge N/2 wieder die geraden und ungeraden Summanden getrennt werden und der komplexe Vektor W^k extrahiert wird. Dies lässt sich solange fortführen, bis die diskreten Fouriertransformationen nur noch einelementig sind - und die DFT einer einelementigen Menge ist sie selbst. Durch wiederholtes Aufteilen bis zu einelementigen Mengen wird die Divide-Phase abgeschlossen. Um den Algorithmus fertig zu bearbeiten, müssen nun nur noch die Twiddle-Faktoren der einzelnen Ebenen verrechnet werden. Die Extraktion und Berechnung der Twiddle-Faktoren ist der Grund für die Effizienz der FFT: Normalerweise würden unter den Vektoren Redundanzen auftreten, durch die spezielle Aufteilung der DFTs werden die redundanten Vektoren zusammengefügt. Durch das Zweiteilen der DFTs auf jeder Ebene entstehen log(n) Ebenen. Daraus lässt sich die Aufwandsklasse der FFT ableiten, die deutlich günstiger als die der naiven

Berechnung der DFT ausfällt. Das Aufteilen auf jeder Ebene in zwei Probleme halber Größe bringt eine Einschränkung der FFT zum Vorschein: Sie lässt sich so nämlich nur für Eingabegrößen der Größe 2^n bestimmen, außer die restlichen Werte, die zur nächsten Zweierpotenz fehlen, werden durch Nullen aufgefüllt (Zero Padding), was das Ergebnis nicht beeinträchtigt, oder sie werden durch normale DFT berechnet. Neben diesen direkten Herangehensweisen gibt es aber auch zahlreiche komplexere Lösungen für dieses Problem, so wurden Algorithmen entwickelt, die eine DFT in mehr als zwei Teile aufteilen können (Radix-N), oder gar auf den einzelnen Ebenen verschiedene Anzahlen an Aufteilungen ermöglichen (Mixed-Radix). Außerdem sind zahlreiche Erweiterungen des Basis-Algorithmus vorhanden, die zusätzliche Performanzsteigerungen versprechen - generell oder für spezielle Fälle.

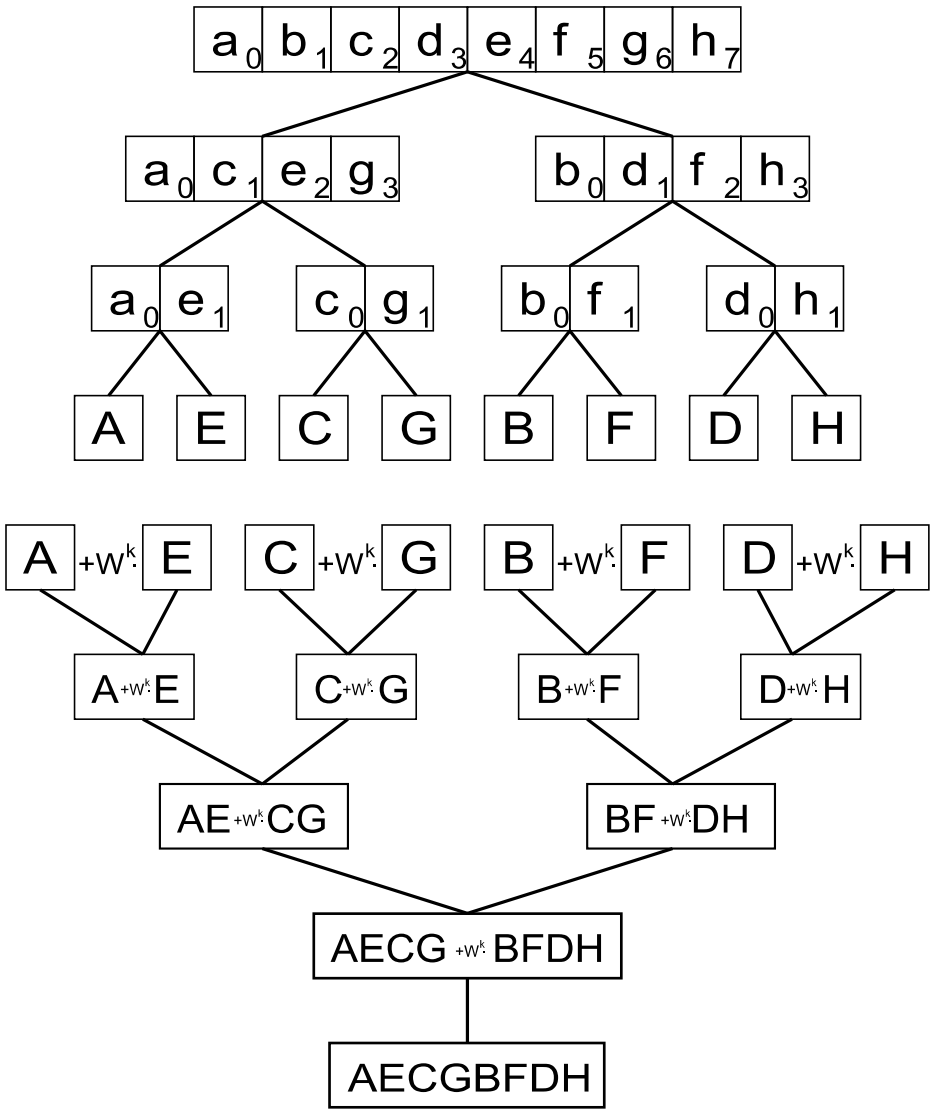

Fig. 5: Graphische Darstellung der schnellen Fouriertransformation. Kleinbuchstaben stehen für Daten im Zeitbereich, Großbuchstaben für solche im Frequenzbereich. Der Algorithmus wird hier wie folgt ausgeführt:

1. log(n)-maliges Aufteilen in gerade und ungerade Indizes

2. DFT einer einelementigen Menge ist diese selbst

3. log(n)-maliges Verrechnen der Twiddle-Faktoren, bis das komplette Spektrum vorhanden ist.

Falls möglich, ist es empfehlenswert, auf eine bereits vorhandene Implementation der FFT zurückzugreifen, denn eine Eigenentwicklung mag zwar lehrreich sein, aber sie wird wohl kaum an die Performanz von professioneller Software heranreichen, die von Expertengruppen über mehrere Jahre entwickelt wurde. Einige Beispiele für FFT-Bibliotheken werden im folgenden Kapitel vorgestellt.

6. Software und APIs

6.1 OpenCL

OpenCL (Open Compute Language) ist eine Schnittstelle, die es ermöglicht, parallele Programmierung plattformübergreifend zu bewerkstelligen. OpenCL wird von modernen GPUs, CPUs, sowie anderen parallelen Rechnerarchitekturen wie IBMs Cell Broadband Engine oder Power-Architektur unterstützt. Erste GPUs für Tablets und Smartphones sind bereits verfügbar, die OpenCL-Code ausführen können[1]. Die aktuellste Version von OpenCL ist die 2.0, allerdings unterstützt aktuell verfügbare Hardware höchstens die Vorgängerversion 1.2 - Im Rahmen dieser Arbeit wurde die Version 1.1 genutzt, da die genutzte GPU nur diese unterstützt.

OpenCL lässt sowohl daten- als auch aufgabenparallele Implementationen zu, sowie das Arbeiten mit mehreren Plattformen, beispielsweise mit mehreren GPUs oder auch CPU und GPU im Verbund. Bei allen OpenCL-Programmen wird zwischen Host-Code und Device-Code unterschieden: Der Host ist das System, auf dem die Anwendung ausgeführt wird und wird mit gewöhnlichen Programmiersprachen wie C++ oder Java programmiert. Die Devices sind OpenCL-fähige Plattformen, die im Host vorhanden sind und umfassen GPUs, die CPU selbst und jede andere Art von OpenCL-fähiger Hardware. Das Verhältnis von Host zu Device ist vergleichbar zu einem Server und Client: Die Devices haben wie der Host Recheneinheiten (Compute Units) und eigenen Speicher. Die Arbeit der Devices (Clients) wird zentral vom Host (Server) verwaltet, alle Eingabewerte werden vom Host zur Verfügung gestellt und alle Ausgabewerte nach der Berechnung wieder von den Devices zum Host zurücktransferiert.

Es stehen zahlreiche API-Aufrufe zur Verfügung, um Devices auszuwählen, Daten zu transferieren oder Aufgaben ausführen zu lassen. Dabei werden Aufrufe, die die Device ansteuern und nicht unmittelbar abgeschlossen sind, in eine oder mehrere Command Queues (Warteschlangen) geschrieben, so dass die einzelnen Aufrufe nicht miteinander interferieren können. Mit Hilfe der Warteschlange und Events kann außerdem sichergestellt werden, dass voneinander abhängige Aufrufe nicht unabhängig voneinander ausgeführt werden.

Für die Programmierung der Devices, die die eigentliche Rechenarbeit beschreibt, steht ein C-Derivat zur Verfügung namens OpenCL C, das auf C99 basiert und teilweise eingeschränkt ist - beispielsweise ist hier keine Rekursion möglich. Der Code wird zur Laufzeit kompiliert und die Funktionen, die vom Host aus aufgerufen werden können, die sogenannten Kernel, werden je nach Wahl des Benutzers von mehreren Threads gleichzeitig ausgeführt. Dabei sind festgelegte Bezeichner vorhanden, mit denen sich im Device-Code unter anderem ermitteln lässt, welche Nummer (ID) der ausführende Thread hat oder wie groß das Gesamtproblem ist.

1 https://developer.qualcomm.com/discover/chipsets-and-modems/adreno-gpu

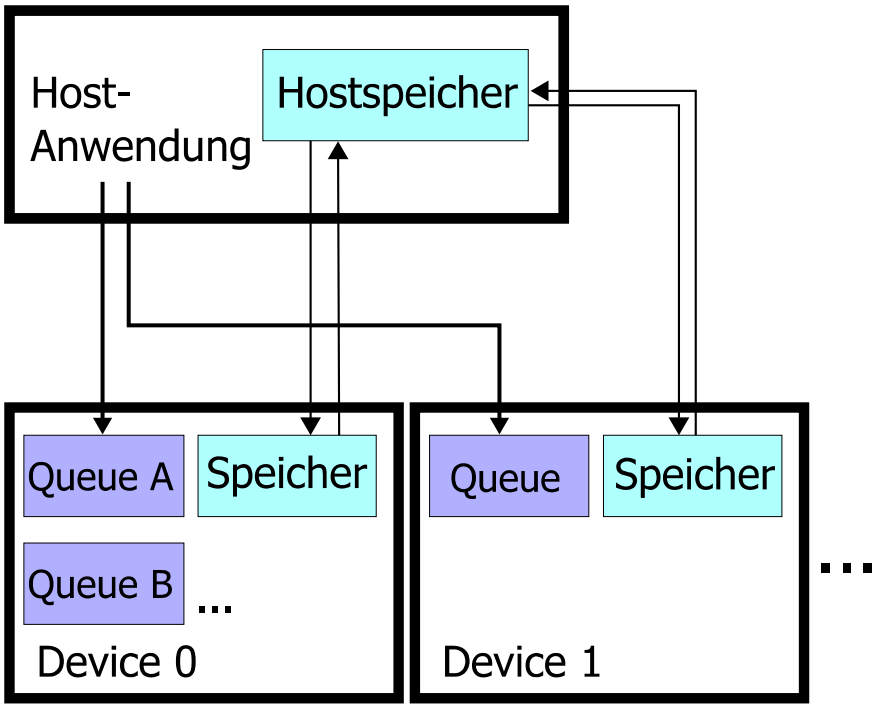

Fig. 6: Beziehung von Host und Devices. Die Host-Anwendung kann über die Command Queues die Devices steuern, wobei eine OpenCL-Device mehrere Queues haben kann.

Moderne GPUs und andere mögliche Devices können teilweise mehrere tausend Threads gleichzeitig bearbeiten. Um die Arbeit effizient verwalten zu können, unterteilt OpenCL die Threads in mehrere Workgroups, die jeweils eine bestimmte Zahl an Threads (Work-Items) zusammenfassen. Die Arbeitsgruppen können mehrdimensional sein, was vor allem beim Arbeiten mit mehrdimensionalen Problemen hilfreich ist (beispielsweise Arbeiten an Texturen oder Volumendaten, oder Modelle physikalischer Vorgänge). Die Unterteilung dient aber nicht nur der Übersichtlichkeit: Die Speicherverwaltung ist ebenfalls auf dieses Konzept ausgelegt: Der Speicher der GPU ist in einen globalen, lokalen und privaten und konstanten Adressraum unterteilt:

Global: Alle Threads können auf diesen Bereich zugreifen, hier treten also auch am ehesten Konflikte auf. Dieser Bereich deckt den Großteil des Device-RAMs ab.

Lokal: Alle Threads einer Workgroup haben Zugriff auf diesen gesonderten Speicherbereich.

Privat: Ein einzelner Thread hat Zugriff auf seinen eigenen privaten Speicherbereich.

Konstant: Alle Threads können auf diesen Adressraum zugreifen, jedoch nur lesend. Daten können nur vom Host aus initialisiert werden. Der Bereich ist nur sehr klein und kann schon für große Lookup-Tables nicht mehr ausreichen. Der Speicher ist jedoch besonders schnell und kann für oft aufgerufene Konstanten verwendet werden.

Die maximale Größe einer Workgroup, sowie die Größen der verschiedenen Speicherbereiche sind abhängig von der Device. Um für jedes Gerät die optimale Performanz zu erreichen, lassen sich diese Werte mittels API-Aufrufen ermitteln.

Bei Echtzeitanwendungen wie dem Konvolutionshall, die einen kontinuierlichen Datenfluss segmentweise bearbeiten, ergibt sich ein mögliches Bottleneck durch den Datentransfer zwischen Host und Device: Alle zu bearbeitenden Daten müssen auf die Device kopiert werden, in der Regel

über PCI-Express und nach der Bearbeitung wieder zurück. Die vorhandene Bandbreite ist mehr als ausreichend für Audioanwendungen, allerdings entsteht durch jeden Transfer eine nicht unerhebliche zeitliche Verzögerung. Es ist also wichtig, dass die Daten, die zwischen Host und Device ausgetauscht werden, in möglichst großen Paketen übermittelt werden - Für den Konvolutionshall ist das möglich, da selbst Abstände der einzelnen Hallfahnen um die 50ms immer noch klanglich zu überzeugen vermögen. Andere Audioanwendungen, bei denen niedrige Latenzen relevant sind, können so jedoch nicht von dem Potential der GPU Gebrauch machen.

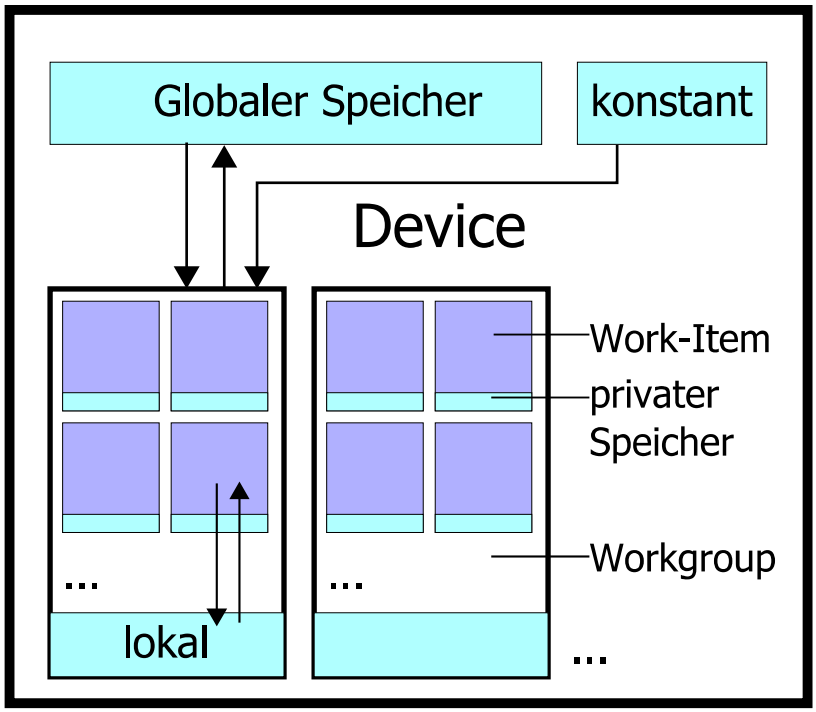

Fig. 7: Speichermodell von OpenCL.

Die für die Entwicklung der Software genutzte Grafikkarte ist eine NVIDIA GeForce GTX 650 Ti Boost. Ursprünglich wurde auf einer AMD Radeon HD 4870 gearbeitet, jedoch stellte sich heraus, dass diese GPU mit AMDs OpenCL-Umgebung APP (Accelerated Parallel Processing) keinen OpenCL-Code fehlerfrei ausführen kann. Laut AMD werden die Karten der 4xxx-er-Reihe nicht mehr offiziell unterstützt, da diese keine Hardware für OpenCL-Event-Handling haben, jedoch führten auch Tests, die ohne solche Events auskamen, zu keinen positiven Resultaten.

6.2 VST

VST steht für Virtual Studio Technology und wurde von Steinberg Media Technologies 1996 als neues Feature mit ihrem bereits seit 1989 vertriebenen MIDI-Sequencer Cubase eingeführt. Seit 1991 war Cubase in der Lage, neben MIDI-Daten auch Audiodateien aufzunehmen und wiederzugeben und 1996 wurde Cubase VST vorgestellt, das Echtzeiteffekte berechnen konnte, die sowohl MIDI-Daten, als auch Audiostreams innerhalb des Sequencers manipulieren können. Außerdem wurde 1999 zusätzlich VSTi vorgestellt, wobei das "i" für Instrument steht. Mit diesen über MIDI steuerbaren Musikinstrumenten ist es letztendlich möglich, Musik komplett am Rechner zu erstellen ohne vorhandenes Audiomaterial oder Aufnahmen von realen Instrumenten. Die

sogenannten VST-Plugins waren zum damaligen Zeitpunkt eine besondere Innovation, denn durch sie wurde die Musikproduktion (ausschließlich) am Computer deutlich intuitiver und das erste Mal gab es somit eine vollwertige Alternative zu hardware-basierten Aufnahmeverfahren. Andere Softwarehersteller zogen nach und haben eigene Schnittstellen für ihre Produkte bereitgestellt, doch heutzutage ist die VST-Schnittstelle am verbreitetsten und die meisten Sequencer, sowie sonstige Softwarelösungen für Aufnahme und Bearbeitung von Audiomaterial können nativ VST-Plugins laden oder beinhalten Wrapper, mit denen sich die Plugins indirekt einbinden lassen. Steinberg pflegt ein offenes SDK, das sowohl Amateur-Entwickler als auch professionelle Studios nutzen können. Dadurch ist eine Vielzahl an Plugins und Instrumenten gegeben, die alle Aufgaben der modernen Musikproduktion abdecken, aber auch experimentielle und unkonventionelle Möglichkeiten bieten.

Die VST-Schnittstelle ist für VSTi und VST-Effekte ausgelegt, die MIDI oder Audio-Daten bearbeiten, wobei alle diese Formen kombiniert werden können: Ein mit MIDI gesteuerter Synthesizer beispielsweise kann einen Audioeingang haben. Somit kann er zusätzlich zu seiner eigenen Klangerzeugung das Eingangssignal manipulieren und selbst wieder einen Audiostream ausgeben. Die Zahl der Kanäle ist dabei beliebig, folglich können auch Plugins für Surround-Sound realisiert werden.

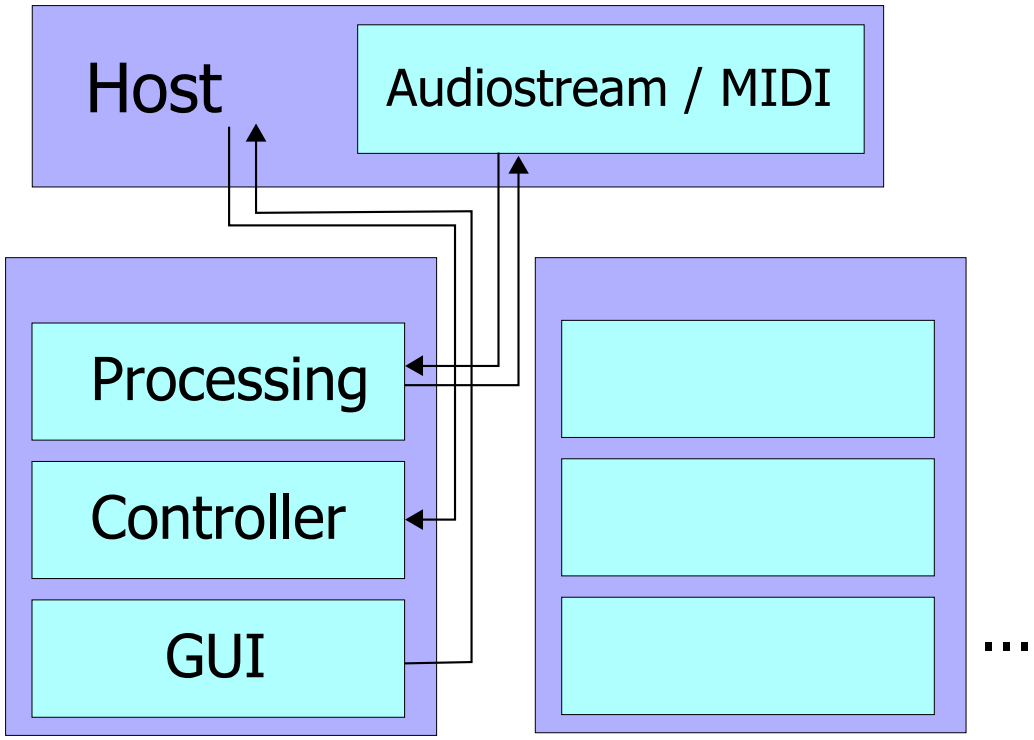

Fig. 8: Aufbau der VST-Schnittstelle. Über den Host werden Benutzereingaben an die Plugins weitergeleitet, die Plugins geben ihre Aktualisierungen für die graphische Oberfläche an die Host-Software zurück. Audio- und MIDI-Streams des Hosts werden kontinuierlich von den Plugins stückweise ausgelesen und müssen in begrenztem zeitlichen Rahmen bearbeitet und zurückkopiert werden.

Zentrales Merkmal der VST-Architektur ist die Trennung von Audioprozessor, Pluginsteuerung und Oberfläche. Diese dient nicht nur der Übersichtlichkeit: Vor allem sind den drei Modulen verschiedene Prioritäten zugeordnet. Die Signalbearbeitung und Klangerzeugung sind immer am Wichtigsten, sowie die Steuerung, während die Aktualisierung der Oberfläche im Ernstfall am ehesten vernachlässigbar ist.

Die aktuellste Version zum Zeitpunkt dieser Arbeit ist die VST 3 API, jedoch wurde im Rahmen der Entwicklung auf die Version 2.4 zurückgegriffen. Dies geschah zum einen aus Kompatibilitätsgründen, aber auch, da die Komplexität der neuen Version sehr viel höher ist - ein einfaches Plugin zu erstellen hat sich bereits als problematisch erwiesen und selbst das Kompilieren der Beispielplugins war nicht ohne weiteres zu bewerkstelligen. Auch die Entwicklung unter der Version 2.4 ist nicht ganz trivial: Ohne weiteres ist dies nur in Visual Studio mit Hilfe zahlreicher spezieller Projekteinstellungen möglich[2]. Andere APIs mit ähnlicher Funktionalität sind sehr viel simpler und intuitiver in ihrer Einrichtung, doch durch die enorme Verbreitung der VST-Schnittstelle bietet es sich trotzdem an, die damit verbundenen Problematiken auf sich zu nehmen.

6.3 Sonstiges

CUDA[3]

Da die Software letztendlich auf einer Plattform mit NVIDIA-GPU entwickelt wurde, wurde diese API kurz angetestet. CUDA stand ursprünglich für Compute Unit Device Architecture und ist eine proprietäre Schnittstelle, die NVIDIA exklusiv für seine eigenen GPUs anbietet. Da sie vor OpenCL eingeführt wurde, war sie lange Zeit der Standard für GPGPU-Programmierung und wurde vor allem in der Wissenschaft zum Standardwerkzeug. Inzwischen löst OpenCL mehr und mehr CUDA ab, trotzdem versucht NVIDIA weiterhin seinen Standard am Leben zu erhalten. So bietet NVIDIA Kernel-Debugging für CUDA, aber (noch) nicht für OpenCL, so dass AMDs GPUs letztendlich die bessere Wahl für OpenCL-Entwicklung sind.

Libsndfile/Secret Rabbit Code[4]

Diese C-Libraries sind unter der Gnu Lesser General Public License verfügbar und bieten simplen Im- und Export von WAV-Files, sowie einigen anderen Audioformaten (libsndfile), sowie qualitativ hochwertige Sampleratenkonvertierung (Secret Rabbit Code, oder auch libsamplerate). Diese ist notwendig, um beispielsweise eine Impulsantwort, die mit 44,1 kHz aufgenommen wurde, auf einen 48 kHz-Audiostream anzuwenden. Ohne die Sampleratenkonvertierung würde die Impulsantwort als zu tief und lang interpretiert werden. Die Implementation der Sampleratenkonvertierung mag trivial wirken, jedoch entstehen bei naiven Lösungen unerwünschte Artefakte, die selbst in mancher professionellen Softwarelösung in nicht unerheblichem Maße vorhanden sind[5].

FFTW[6]

FFTW (Fastest Fourier Transform in the West) ist eine sehr leistungsfähige FFT-Library, die verschiedene gängige CPU-Architekturen optimal nutzt und beliebige Eingabegrößen zulässt - auch für prime Größen ist ein Aufwand von O(n log n) garantiert. Die Software ist unter der GNU General Public License veröffentlicht und bietet im non-kommerziellen Bereich eine attraktive Lösung zur Berechnung der FFT auf der CPU. Für die im Rahmen dieser Arbeit entwickelte Software wurde FFTW für die CPU-Implementierung verwendet.

2 http://teragonaudio.com/article/How-to-make-VST-plugins-in-Visual-Studio.html

3 http://www.nvidia.de/object/cuda-parallel-computing-de.html

4 http://www.mega-nerd.com/

5 http://src.infinitewave.ca/

6 http://www.fftw.org/

ClAmdFFT[7]

NVIDIA hat mit cuFFT schon lange eine Library für FFT-Berechnung auf der GPU bereitgestellt, die sich großer Beliebtheit unter Anwendern erfreut, jedoch in CUDA realisiert ist und so nur NVIDIAs GPU-Architekturen unterstützt. Inzwischen hat AMD mit clAmdFFT eine Library nachgelegt, die in OpenCL geschrieben ist und somit auf jeder Architektur laufen sollte, die eine OpenCL-Device darstellt. Dabei ist zu beachten, dass AMD die Kernel optimiert hat für die eigenen GPUs.

Zunächst schien diese Library die erste Wahl für diese Arbeit zu sein, allerdings stellte sich bald heraus, dass die genutzte Grafikkarte generell nicht (mehr) OpenCL-fähig ist. Nach dem Wechsel auf NVIDIA-Hardware wurde zunächst nach Alternativen zu der Library gesucht, doch erneute Versuche führten schließlich zu positiven Ergebnissen, wobei auch die Ausführungszeit alle Erwartungen erfüllte.

ViennaCL[8]

ViennaCL ist eine kostenlose Open-Source Library, die verschiedene algebraische Probleme auf parallelen Architekturen lösen kann. Dabei werden neben OpenCL auch CUDA und OpenMP unterstützt. Leider ist die FFT-Implementierung noch in der Testphase und unvollständig und das Umwandeln der Daten in das Format von viennaCL und das Einrichten der API sind aufwändiger als zu erwarten wäre. Aus diesen Gründen wurde die Idee, diese Library zu nutzen, sehr bald wieder verworfen.

7. Implementation

7.1 Der Algorithmus

Die Zielplattform, also die VST API, bietet ein Interface namens processReplacing(), in dem die eigentliche Signalverarbeitung implementiert wird. Die Parameter sind ein Eingangspuffer, Ausgangspuffer sowie der Wert "sampleFrames", der die Länge beider Puffer festlegt. Diese Funktion wird während der Laufzeit des VST-Hosts jedes Mal aufgerufen, wenn neue Audiodaten vorhanden sind. Die Länge der Segmente lässt sich im Host einstellen, wobei kleine Werte niedrige Latenzen bedeuten und große in der Regel bessere Performanz. Die eingestellte Länge ist der Wert, der in "sampleFrames" übergeben wird. Die beiden Puffer können beliebig viele Kanäle haben, so dass auch Stereo- oder Surroundbetrieb möglich sind.

Zunächst gilt es, das Originalsignal (dry) durchzuschleifen, um zu diesem dann das Hallsignal (wet) hinzuzumischen. Für jeden Durchlauf wird der Eingangspuffer in das Array "CollectedIns" eingefügt, sodass die Eingangspuffer mehrerer Durchläufe darin aufgereiht sind, bis die Gesamtlänge dem Wert "density" entspricht. Dieser Wert wird vom Nutzer gewählt und legt fest, in welchen zeitlichen Abständen Hallfahnen generiert und zugemischt werden. Wohlklingende Werte finden sich zwischen 20ms und 90ms. Geringere Werte führen zu Kammfilter-artigen

7 http://developer.amd.com/tools-and-sdks/heterogeneous-computing/amd-accelerated-parallel-processing-math-libraries/

8 http://viennacl.sourceforge.net/

Effekten, während bei größeren Werten unter Umständen die einzelnen Hallfahnen hörbar sind, so dass ein ungewollter Rhythmus entsteht. Zu beachten ist, dass der "density"-Wert in Samples gewählt wird, also abhängig von der gewählten Samplerate ist. Für 44,1 kHz würden 1764 Samples 40ms entsprechen, für 96 kHz wären es 3840 Samples.

Das Befüllen von "CollectedIns" sorgt dafür, dass jedes Stück Audiostream auf die entstehenden Hallfahnen einwirkt. Würden in regelmäßigen Abständen kürzere Abschnitte als Grundlage für die Faltungen genutzt werden, könnten kurze Impulse, wie Perkussion, Vinylknistern oder Schüsse - übergangen werden und ihr Einfluss auf die Hallfahne würde somit ausbleiben, obwohl er sehr wichtig für das Klangbild wäre. Ist "CollectedIns" nun groß genug, wird der Inhalt mit der Impulsantwort gefaltet, die im Voraus bereits geladen wurde und im Falle der Nutzung der FFT außerdem bereits transformiert wurde. "CollectedIns" wird für die erneute Beschreibung wieder zurückgesetzt und das Resultat der Faltung wird in einen Puffer geschrieben, wie er in Figur 9 gezeigt wird: Neben der Addition der neuen Hallfahne auf den Puffer werden in jedem Durchlauf die restlichen Werte um die Länge der Eingangs- und Ausgangspuffer verschoben. Dies entspricht dem zeitlichen Fortschritt, den das neue Puffermaterial gegenüber dem letzten Aufruf hat.

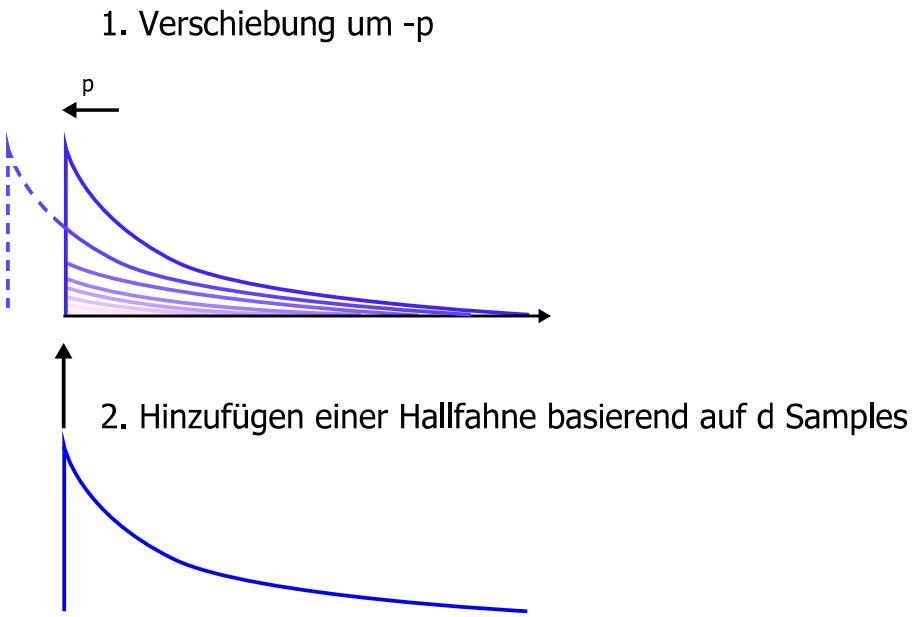

Fig. 9: Verwaltung der Hallfahnen. Zunächst werden die bestehenden Werte im Puffer in negativer Richtung um die Größe des Eingabepuffers p verschoben, also in die Vergangenheit. Dies geschieht in jedem Programmdurchlauf. Wurde eine neue Hallfahne generiert, wird sie in den Puffer geschrieben. Sie basiert auf der Impulsantwort und d Samples(d ist der gewählte "density"-Wert).

Zum Schluss werden zum aktuellen Eingangssignal die ersten Einträge des Hallfahnenpuffers gemischt, entsprechend der durch den Benutzer wählbaren Variable "wet". Außerdem kann der Nutzer durch "crossover" entscheiden, wie beim Stereosignal der linke und rechte Kanal interagieren: Werden einfach linker Kanal der Impulsantwort und der linke des Eingangs gefaltet und getrennt ausgegeben (analog dazu die rechte Seite), mag das zwar meist gut klingen. Angenommen aber ein Signal würde lediglich auf dem linken Kanal vorhanden sein und der rechte wäre still, würde es auch nur auf der linken Seite hallen, was einem natürlichen Raum in keinster Weise entspricht.

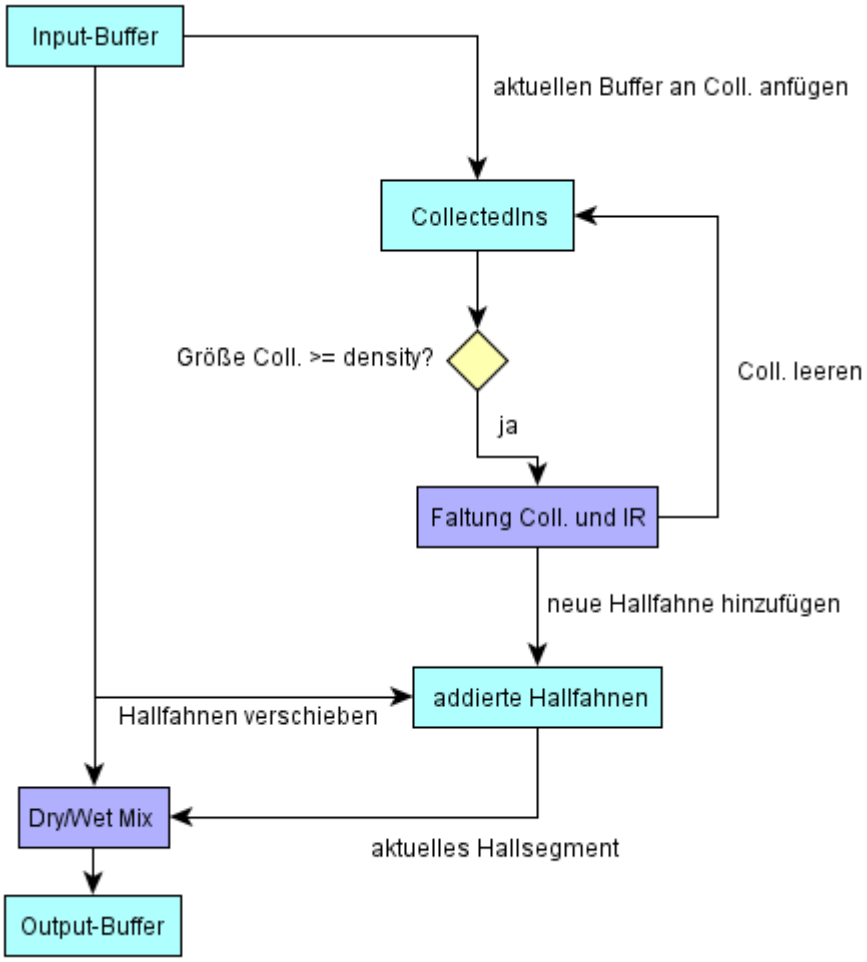

Fig. 10: Ablauf des Algorithmus. Für jeden Programmdurchlauf werden die aktuellen Werte des Input-Buffers zu "CollectedIns" hinzugefügt und an den Ausgang weitergeleitet. Außerdem wird ein Segment des Halls der Länge des Input-Buffers zum Originalsignal gemischt und die gesammelten Hallfahnen um diese Länge verschoben. Ist die Anzahl an Samples in "CollectedIns" mindestens so groß wie der gewählte "density"-Wert, wird der Inhalt mit der Impulsantwort gefaltet und die neue Hallfahne in den Puffer eingefügt.

Beide Hallfahnen zu addieren und auf beide Kanäle zu legen würde ebenfalls kein attraktives Resultat bewirken, da im natürlichen Raum links und rechts unterschiedliche Hallmuster auftreten. Durch die Variable "crossover" kann der Nutzer von der kompletten Trennung der linken und rechten Hallfahne, über Mono-Mischung bis hin zu komplett links und rechts vertauschten Hallfahnen wählen. Letztere Einstellung ist vor allem für experimentielles Sounddesign attraktiv. Während der Laufzeit kann der "crossover"-Wert moduliert werden, um noch surrealere Eindrücke von Räumlichkeit zu erschaffen.

Für die Faltung ist es unumgänglich, dass beide Signale die gleiche Länge haben. Um diese Voraussetzung zu erfüllen, lässt sich das kürzere Signal mit Nullen auffüllen (Zero-Padding, siehe 5.3). Eine Alternative, die sich leider als fehlerhaft erwiesen hat, wird in 9.1 beschrieben.

7.2 Offline-Testprogramm

Für die Entwicklung wurde, da die VST-Plugins sich nicht gut debuggen lassen, zunächst eine Offline-Variante entwickelt. Hierbei wird in ein normales C++-Programm eine Audiodatei gelesen, die dann iterativ in Blöcken bearbeitet wird, wonach das Resultat in eine neue Audiodatei geschrieben wird. Dadurch lässt sich die entstehende Wellenform samplegenau auf Artefakte oder mögliche Fehler überprüfen. Außerdem lässt sich ein solches Programm ohne weiteres Schritt für Schritt debuggen und die Fehlercodes von OpenCL oder anderen Libraries lassen sich auslesen. Die eigentliche Berechnung der Hallfahnen und das Zumischen dieser zum Ausgangssignal sind dabei komplett in eine gesonderte Funktion ausgelagert, so dass sich eine funktionierende Implementation reibungslos in das VST-Plugin übertragen lässt.

Das Testprogramm wurde auch für Performanz-Messungen und Benchmarking genutzt. Das Eingangssignal kann hier mit einer festen Länge gewählt werden, wodurch durch Vergleich der Ausführungszeiten des Programms direkte Schlüsse zur Performanz gezogen werden können. Diese Ergebnisse lassen sich besonders gut miteinander vergleichen.

7.3 VST-Plugin

Die VST-Schnittstelle ist objektorientiert aufgebaut und liefert für die Implementation einige hilfreiche Anhaltspunkte. Die Hauptklasse des Plugins hat einen Konstruktor, der aufgerufen wird, sobald der Host das Plugin lädt. Für ein einfaches Testprogramm, in dem nur eine Impulsantwort eingesetzt wird, die nicht während der Laufzeit geändert werden kann, kann diese im Konstruktor geladen und transformiert werden, letzteres falls das Plugin mit schneller Faltung betrieben wird. Außerdem können hier der benötigte Speicher alloziert und alle Variablen mit Default-Werten initialisiert werden. Darüber hinaus ist es zwingend erforderlich einige API-Aufrufe zu tätigen, die bei allen VST-Plugins erwartet werden. Unter anderem muss die Unique ID gesetzt werden, die das Plugin identifiziert und gewährleistet, dass in dem Fall, dass zwei Entwickler denselben Namen für ihr Plugin wählen, sich trotzdem beide simultan in einem Host laden lassen. Die Aufrufe definieren auch den Typ des Plugins (Instrument, Effekt oder beides) und wie viele Ein- und Ausgänge dieses hat.

Über weitere Funktionen lässt sich eine provisorische Benutzeroberfläche einrichten, mit deren Hilfe sich das Plugin für Testzwecke bedienen lässt. Für den Faltungshall wurden zunächst der Dry/Wet-Regler, der das Verhältnis Eingangssignal zu Hallsignal steuern lässt, sowie Regler für die bereits beschriebenen Variablen "density" und "crossover" eingerichtet. Optionale Erweiterungen lassen sich unkompliziert einbinden und außerdem kann im Nachhinein eine ansprechendere Oberfläche gestaltet werden.

7.4 CPU-Variante

Für die CPU-Variante wurde die FFTW-Library gewählt, die hohe Performanz verspricht und die verschieden langen Impulsantworten ohne weitere Anpassungen transformieren kann. In der Library sind Funktionen vorhanden, um eine rein reell-wertige Eingabe mit n Werten in ein n/2+1 langes

komplexes Spektrum zu transformieren und aus diesem wieder ein n langes Signal zurück zu transformieren. Hierbei werden die sogenannten "hermitischen Symmetrien" ausgenutzt, die bei rein reell-wertigen Eingaben auftreten. Abgesehen davon, dass diese Funktionen schneller arbeiten können als die für komplexe Eingaben, werden hier die Hälfte der komplexen Multiplikationen gespart, die für die Faltung vonnöten sind. Das Ausführen einer Transformation mit FFTW erfordert die vorherige Definition eines sogenannten Plans, in dem Pointer auf die Eingabe- und Ausgabedaten, Anzahl der Werte, Transformationsrichtung und anderes festgelegt werden. Wird der Plan aufgerufen, testet FFTW, auf welche Art sich die spezifisch gestellte Aufgabe am effizientesten lösen lässt. Dies dauert zwar einige Zeit, muss aber nur einmal gemacht werden.

Diese Art der Implementierung hat das Problem, dass die Durchläufe des Plugins, bei denen eine Faltung durchgeführt wird, sehr viel rechenintensiver sind als die übrigen. Um diese Spitzen auszugleichen, bietet es sich an, die Faltung in einen zweiten Thread auszulagern, dessen Ergebnisse vom ersten einfach nur dann abgefragt werden, wenn er diesem über ein Event diesem mitteilt, dass die Berechnung abgeschlossen ist. Diese Variante verspricht hohe Stabilität, jedoch zu dem Preis, dass das klangliche Ergebnis von der Performanz des Systems abhängt und kaum noch vorhersehbar ist. FFTW bietet ebenfalls Multithreading für die Ausführung einer einzelnen FFT, allerdings sollte im Kontext der vorgesehenen Nutzung beachtet werden, dass das Plugin sich in der Praxis die vorhandene Rechenzeit mit zahlreichen anderen Plugins teilen muss. Es hat sich ergeben, dass die Verwaltung der Threads mehr Overhead erzeugt als sie Speedup bringen, tatsächlich war die Multi-Threading-Variante in Testläufen deutlich langsamer.

7.5 GPU-Variante

Für die GPU-Implementation ist es besonders wichtig, die Zahl der Datentransfers zwischen GPU und System bei entsprechend größeren Datenmengen niedrig zu halten. Der größte Rechenaufwand fällt im beschriebenen Algorithmus auf die Faltung und diese Aufgabe lässt sich sehr gut parallelisieren. Der meiste Code entspricht also der CPU-Variante, die gesammelten Eingabepuffer werden für die Faltung auf die GPU verlagert und mit der bereits berechneten Hallfahne dann komplett auf der GPU gefaltet. Es ist in OpenCL möglich, mehrere Kernel seriell auf dem Device-Speicher arbeiten zu lassen, ohne dass zwischendurch Ergebnisse zum Host zurückgesendet werden müssen. Somit können nacheinander FFT, komplexe Multiplikationen und iFFT effizient bearbeitet werden.

Zunächst wurde überprüft, ob es möglich ist, die Faltung direkt, also ohne Ausnutzung des Faltungstheorems zu bewerkstelligen. Der entsprechende Kernel ist sehr simpel, jedoch auch sehr ineffizient. Im nächsten Kapitel wird unter anderem diese Variante untersucht. Bevor letzten Endes clAmdFft für die schnelle Faltung genutzt wurde, wurde mit selbstgeschriebenen FFT-Kernel experimentiert. Aufgrund der fehlenden Möglichkeit der Rekursion in OpenCL C ist eine einfache Implementation, wie sie in 5.4 vorgeschlagen wird, so jedoch nicht möglich. Ein verbreiteter Algorithmus der getestet wurde, ist nicht für parallele Berechnung geeignet. Um das Potential der GPU richtig auszunutzen, ist es notwendig, einen Algorithmus aufzubauen, der auf eine Problemgröße genau zugeschnitten ist. Die Kernel für individuelle Problemgrößen lassen sich zwar auch zur Laufzeit generieren, jedoch ist eine solche Umsetzung ein sehr komplexes Unterfangen.

Für die Nutzung von clAmdFft sind einige API-Aufrufe notwendig, um die Kommunikation mit dem bestehenden OpenCL-Kontext aufzubauen. Außerdem werden sogenannte Pläne angelegt, die mit denen aus FFTW vergleichbar sind. Die verschiedenen Eckdaten der Transformationen werden hier festgelegt, aber im Gegensatz zu den Plänen von FFTW, die bereits Pointer auf die zu bearbeitenden Daten verlangen, werden die entsprechenden Daten erst beim Aufruf einer Transformation angegeben. Zu beachten ist außerdem, dass hier OpenCL-Speicherobjekte statt der Pointer auf Hostspeicher verlangt werden.

8. Evaluation

8.1 Das Versuchsszenario

Der kritische Teil des Algorithmus ist zweifelsohne die Faltung, in diesem Kapitel soll daher ausschließlich die Performanz der verschiedenen Faltungsvarianten verglichen und bewertet werden. Die vier zu überprüfenden Implementationen sind direkte und schnelle Faltung, jeweils auf der CPU und auf der GPU umgesetzt.

Dazu wird ein festgelegtes Szenario aufgebaut, was auf der Offline-Implementierung basiert: Für ein eine Million Samples langes Signal (ca. 22 Sekunden) wird das Programm ausgeführt, wobei alle Kombinationen aus den verschiedenen Implementationen der Faltung mit drei unterschiedlich langen Impulsantworten überprüft werden. Die Impulsantworten sind 20000 Samples (etwas weniger als eine halbe Sekunde bei CD-Qualität), 100000 Samples (entsprechend etwa 2,3 Sekunden) und 300000 Samples lang (knapp 7 Sekunden). Die Puffergröße, die simuliert wird beträgt 512 Samples (ca. 10ms) und die "Density"-Einstellung einmal 1024 Samples und einmal 2048 Samples. Diese Werte repräsentieren ein typisches Anwendungsszenario, wobei die Performanz des Plugins weitestgehend unabhängig von der Puffergröße des Hosts ist. Um eine ausreichend genaue und unverfälschte Zeitmessung zu erreichen, wird mit der in "time.h" enthaltenen clock()-API die Zeit gestartet, nachdem alle Konstruktoren und sonstigen Vorbereitungen wie das Laden der Quelldatei und der Impulsantwort oder die Transformation dieser abgeschlossen sind. Anschließend wird die Zeit gestoppt, sobald die 1000000 Samples abgearbeitet sind, jedoch bevor die resultierende Wellenform ausgelesen wird. Die clock()-API bietet Präzision bis auf die Millisekunde genau und ist damit ausreichend (idealerweise sollte die gemessene Zeit die 22 Sekunden des Originalsignals unterschreiten). Für die Ergebnisse, die nicht länger als eine Minute sind, wurden fünf Durchläufe gemacht und die Messungen gemittelt. Für längere Arbeitszeiten wurden nur bei ungewöhnlich hohen Abweichungen wiederholte Tests ausgeführt, um das Ergebnis zu verifizieren beziehungsweise zu widerlegen. Die teilweise extrem lang ausführenden direkten Faltungen auf der CPU wurden nur mit 10000 Samples Quellmaterial ausgeführt und die Ergebnisse dann mit 100 multipliziert, um einen Vergleich mit den restlichen Werten zu ermöglichen.

Das Testsystem basiert auf einem Intel Core2Quad Q6700 mit 4x2,66 GHz Taktung, 8GB DDR2-800 RAM und der bereits erwähnten NVidia Geforce GTX 650 Ti Boost mit 2GB RAM. Als Betriebssystem kommt Windows 7 Professional zum Einsatz.

8.2 Ergebnisse und Auswertung

Folgende Messergebnisse wurden aufgezeichnet (Werte in Sekunden):

Density: 1024

Länge der Impulsantw.	20000 Samples	100000 Samples	300000 Samples
CPU direkte Faltung	2516,550	69309 (19,5 Stunden)	626360 (~1 Woche)
CPU schnelle Faltung	22,682	83,133	252,029
GPU direkte Faltung	19,843	502,165	4601,182
GPU schnelle Faltung	6,069	22,339	67,688

Density: 2048

Länge der Impulsantw.	20000 Samples	100000 Samples	300000 Samples
CPU direkte Faltung	915,682	34544 (9,5 Stunden)	317629 (~3 1/2 Tage)
CPU schnelle Faltung	7,719	25,966	76,612
GPU direkte Faltung	9,267	244,312	2330,622
GPU schnelle Faltung	4,757	12,180	35,271

Zunächst fällt der extreme Unterschied der Ausführungszeit zwischen den direkten und schnellen Faltungen auf: Von 20000 auf 100000 Samples wird die Ausführungszeit circa 25-mal höher für CPU- und GPU-Variante, von 100000 auf 300000 Samples etwa 10-mal höher, obwohl die Eingabemenge nur verdreifacht wurde. An den direkten Faltungsoperationen wird auch deutlich, dass die GPU hier ihre Stärken perfekt ausspielen kann. Es sollte hier zwar erwähnt werden, dass die direkte Faltung auf der CPU lediglich auf einem Thread läuft, dennoch ist die Ausführungszeit hier enorm hoch und für die Verwendung in einem VST-Plugin in keinster Weise niedrig genug. Bis auf die schnelle Faltung auf der CPU lässt sich auch beobachten, dass die Verdoppelung des Density-Werts die Ausführungszeit etwa halbiert. Dies ist zu erwarten, da nur noch die Hälfte der Hallfahnen zu berechnen sind. Für die schnelle Faltung auf der GPU fällt der Speedup etwas geringer aus, für die schnelle Faltung auf der CPU ist er aber mehr als dreimal so hoch - wie diese Werte zustande kommen obliegt einer näheren Untersuchung.

Die eigentliche Fragestellung, ob die Faltungsalgorithmen für ein VST-Plugin genutzt werden können, lässt sich mit Hilfe der Messwerte gut beantworten. Die schnelle Faltung kann in besonders günstigen Fällen bereits auf der CPU bearbeitet werden, wobei jedoch zu beachten ist, dass nur sehr kurze Impulsantworten genutzt werden können, beziehungsweise weitere Optimierungen angebracht wären. Die Nutzung der CPU birgt das Problem, dass diese im Host auch von vielen anderen Threads beansprucht wird, so dass der einzelne Effekt nicht zu sehr ins Gewicht fallen darf.

Bei der GPU hingegen, die derzeit im Audiobereich kaum genutzt wird, muss lediglich beachtet werden, dass der Anwender mehrere Instanzen des Effekts starten kann, so dass diese sich ihre Ressourcen teilen müssen. Die Ausführungszeiten legen nahe, dass mit Hilfe von Optimierungen selbst lange Hallzeiten möglich und kurze sogar in der hier vorgestellten Form realisierbar sind. Außerdem ist FFTW sehr viel weiter für die genutzte CPU optimiert als clAmdFft für die genutzte (NVIDIA) GPU, eine besser optimierte Library könnte hier noch drastischere Differenzen bewirken, wie sie bei den direkten Faltungen zu beobachten waren.

9. Optimierung und Erweiterung

Die Resultate der Evaluation legen nahe, durch verschiedene Ansätze den Rechenaufwand der Faltung weiter zu senken. Neben weiteren Optimierungen an den jeweils verwendeten FFT-Algorithmen, die nur geringe Performanzsteigerungen versprechen, können durch Einsparungen in der Berechnung, die Qualitätsverluste mit sich bringen, erheblich schnellere Rechenzeiten erzielt werden. Dabei ist wichtig, die Einsparungen so zu wählen, dass das menschliche Gehör den Unterschied möglichst nicht wahrzunehmen vermag.

Außerdem wurden in vorliegender Arbeit verschiedene Überlegungen vorgestellt, die dazu beitragen, die Nutzbarkeit und die Vielseitigkeit des Plugins zu verbessern, um den Anwendungsbereich des Faltungshalls zu erweitern. Der Vorteil des algorithmischen Halls, durch mehr Einstellungsmöglichkeiten den Hallklang für jede spezielle Situation anpassen zu können, kann so auch teilweise auf den Faltungshall übertragen werden.

9.1 Segmentielle Faltung

Eine Idee zur Steigerung der Performanz, die bereits sehr früh aufgekommen ist, ist die, auf das Zero-Padding zu verzichten und mehrere kurze Faltungen anstelle der einen, sehr großen Faltung vorzunehmen. Der Grundgedanke war, dass wenn ein Signal aufgeteilt wird und jeweils einmal transformiert und zurücktransformiert wird, das Originalsignal verlustfrei wiederhergestellt wird. Für die Umsetzung wird die Impulsantwort in Segmente, die in der Größe dem Wert "density" entsprechen zerteilt und dann jedes dieser Segmente mit den Werten in "CollectedIns" gefaltet. Während die Performanz durch diese Vorgehensweise tatsächlich enorm gesteigert wurde, wurde leider auch das klangliche Resultat verfälscht: In den Hallfahnen taucht ein Echo auf, wobei die Länge der Intervalle "density" entspricht. Es lässt sich auch einfach widerlegen, dass diese Optimierung richtige Ergebnisse liefert: Angenommen, das Signal wird in Segmente der Länge 1 aufgeteilt. Die Fouriertransformation eines einzelnen Wertes ist der Wert selber, also wäre die Faltung einfach nur noch die Multiplikation zweier Werte im Zeitbereich, welche bekanntermaßen nicht der Multiplikation im Frequenzbereich entspricht. Das Resultat würde klanglich einem Ringmodulator entsprechen, welcher nämlich genau die Multiplikation im Zeitbereich bewerkstelligt.

9.2 Verlustbehaftete Faltung

Da die meisten Impulsantworten für natürliche Räume keine der oberen Frequenzen einnehmen, kann für diese Fälle überlegt werden, die FFT-Größen für die Transformationen in das komplexe Spektrum zu halbieren, um so nur die Frequenzen bis zur Hälfte der maximal hörbaren zu berechnen. Der Performanzgewinn ist beträchtlich, jedoch ist zu beachten, dass die Faltung auch für experimentielles Sounddesign genutzt werden kann und dass dort die hohen Frequenzanteile dann doch von Bedeutung sein können. Eine solche Funktionalität sollte vom Nutzer frei wählbar sein oder das Spektrum der genutzten Impulsantwort kann auf hohe Frequenzanteile überprüft werden können. Sind diese nicht vorhanden - oder verschwindend gering - können die FFT-Größen ohne klangliche Einbußen reduziert werden.

9.3 Mögliche Erweiterungen

Im Falle eines guten algorithmischen Halls lässt sich der Klang des Halls durch zahlreiche Parameter sehr genau anpassen, um bei verschiedenen Gegebenheiten die gewünschten Ergebnisse zu erzielen. Beim Faltungshall ist es schwerer, den Klang durch Veränderung von Parametern so zu ändern, dass das Ergebnis glaubwürdig bleibt - vor allem ist die Auswahl einer passenden Impulsantwort der entscheidende Faktor. Durch Manipulation einer Impulsantwort lassen sich allerdings unterschiedliche Variationen bewerkstelligen.

Für das menschliche Gehör spielt der Pre-Delay für die Lokalisierung einer Schallquelle im Raum eine entscheidende Rolle. Das Kürzen oder Verlängern dieser Phase durch Anhängen von Stille gibt zusätzlichen Einfluss auf die Positionierung einzelner Elemente. Spreizen oder Stauchen einer Impulsantwort lässt den dargestellten Raum kleiner oder größer erscheinen. Dies lässt sich nur bis zu einem gewissen Grad nutzen, abgesehen von surrealen und experimentiellen Klangexperimenten. Für diese kann auch der Startpunkt in der Impulsantwort geändert werden: Zum einen kann der Hall dadurch dichter wirken, allerdings auch dunkler. Letzteres kommt durch den in den meisten Fällen vorhandenen schnelleren Abfall der hohen Frequenzen den tiefen gegenüber zustande. Wird die Impulsantwort rückwärts eingelesen, resultiert ein Hall, der nicht ausklingt, sondern anschwillt, bis die Länge der Impulsantwort erreicht ist. Für besonders skurrile Ergebnisse, die allerdings einen sehr digitalen Charakter haben können, lassen sich die einzelnen Elemente der fouriertransformierten Signale und Impulsantworten unterschiedlich anordnen, um völlig unerwartete Frequenzanteile in die Hallfahnen einzuarbeiten.

Als zusätzliche Erwägung wäre noch ein variabel einstellbarer Hochpassfilter für die Hallfahne denkbar. Ein Signal mit starken Bässen kann durch einen Hall, der diese ebenfalls begünstigt, schnell an Definition und Klarheit verlieren. Hier ist es deswegen sinnvoll, die tiefen Frequenzen des Eingangssignals beizubehalten und die der Hallfahne herauszufiltern. Daraus resultiert ein unrealistisches Ergebnis, das aber für die Musikproduktion besser nutzbar ist; hierbei sind vor allem hohe Frequenzen für die Halldarstellung wichtig.

10. Zusammenfassung und Ausblick

Der Faltungshall bietet eine perfekte Nachbildung eines natürlichen Raumhalls, wobei die Idee, die ihm zugrunde liegt, es ermöglicht, alleine mit einer Impulsantwort die gewünschten Ergebnisse zu erzielen - es ist somit kein detailliertes Wissen über die Beschaffenheit des gewählten Raums vonnöten. Im Gegensatz zu algorithmischen Lösungen ist auch kein tiefgehendes Wissen über die physikalischen Vorgänge, die zum Nachhall führen, eine zwingende Voraussetzung.

Aktuelle Hardware bietet ausreichend Leistung, um einen Faltungshall auf einem gewöhnlichen Rechner auszuführen und nicht mehr nur auf speziellen DSP-basierten Plattformen. Dabei bietet sich die GPU mit ihrer parallelen Architektur besonders gut an - für die Faltung sind die Performanzsteigerungen beträchtlich. Um lange Nachhallzeiten zu ermöglichen, ist geboten, die eigentliche Implementation des Algorithmus zu optimieren, wobei abgewogen werden muss, wie weit Qualitätseinbußen verkraftbar sind. Ohne solche Optimierungen wird der Effekt jedenfalls nicht in einem Echtzeitkontext nutzbar sein. Auch ist es fraglich, ob andere Audioeffekte von der GPU-Beschleunigung in gleicher Weise wie der Faltungshall profitieren können. Wird auf niedrige Latenzen abgezielt, kann die PCI-Express-Schnittstelle sich schnell als Flaschenhals erweisen.

Der GPU-beschleunigte Faltungshall bietet noch einen großen Vorteil gegenüber einer CPU-Implementation, abgesehen von der besseren Performanz. Denn da die GPU in Audiosoftware bis jetzt kaum Verwendung findet und durch die genannten Einschränkungen auch weiterhin nur beschränktes Potential in dieser Hinsicht bietet, kann ein GPU-beschleunigter Faltungshall die CPU erheblich entlasten. Die neue Generation von Apples Mac Pro, der seit langem die vorrangige Plattform für Multimedia-Anwendungen ist, ist mit 2 leistungsfähigen Workstation-GPUs ausgestattet[9], so dass ein solches Plugin auch zahlreiche Anwender finden kann.

Obwohl OpenCL plattformunabhängig ist und damit zukunftssicher und für viele Anwender nutzbar, wird langfristig paralleles Rechnen wohl noch in eine andere Richtung gehen. Denn die Trennung von Device und Host, oder in Hardwareform respektive GPU und CPU, führt zu dem Problem des Bottlenecks, das durch den PCI-Express-Bus oder sonstige Anbindung gegeben ist. Um alle Probleme mit Potential zur Parallelisierung effizient lösen zu können, muss Hardware entwickelt werden, bei der die klassische CPU und die parallele Recheneinheit gemeinsam einen Speicher nutzen. Erste CPUs mit integrierten GPUs sind bereits vorhanden, aber die Leistungsfähigkeit letzterer ist noch sehr viel geringer als die einer diskreten GPU. Der steigende Einfluss von portablen Lösungen, bei denen das Zusammenführen von CPU und GPU auf einem Chip bereits Standard ist, könnte auch Entwickler von Desktop- und Workstation-Hardware zum Umdenken bewegen.

9 http://www.apple.com/chde/mac-pro/

11. Glossar

DSP (Digital Signal Processor)

Digitale Signalprozessoren sind darauf ausgelegt, diskrete Signale in Echtzeit zu bearbeiten und sind dafür in vielen Punkten anders aufgebaut als klassische CPUs oder Mikrocontroller. Sie bieten optimierte Befehlssätze, die in der Signalverarbeitung häufig benötigte Operationen besonders schnell bearbeiten können.

Equalizer (EQ)

Effekt in der Signalbearbeitung, der frequenzabhängige Verstärkung oder Abschwächung ermöglicht.

GPGPU (General Purpose Computation on Graphics Processing Unit)

Bearbeitung verschiedener Algorithmen auf der GPU, die nicht-klassische Grafikberechnungen beinhalten. Aufgrund der Architektur moderner GPUs vor allem für parallelisierbare Aufgaben geeignet.

Hallfahne

Die Antwort eines Raums auf ein Signal, das in ihm erklingt. Effekte segmentieren in der Regel das Eingangssignal und generieren dann für jedes Segment eine Hallfahne, die dann mit den restlichen Hallfahnen und dem Eingangssignal gemischt wird.

MIDI (Musical Instrument Digital Interface)

MIDI-Daten dienen der Steuerung von Instrumenten und können sowohl Musikinformationen als auch Änderungen der Klangeigenschaften oder Verwaltungsoperationen kommunizieren.

PCM (Pulse Code Modulation)

Gängiges Verfahren zur digitalen Repräsentation einer Wellenform. Hierbei wird das Signal in regelmäßigen Abständen abgetastet (Samplerate) und für jeden der Abtastpunkte wird die Amplitude des Signals an der entsprechenden Stelle durch einen Zahlenwert digital repräsentiert als sogenanntes Sample. Die Abtastrate bei CD-Qualität beträgt 44,1 kHz, die Amplitudenauflösung 16 bit. VST arbeitet intern mit 32 bit float-Werten, professionelle Analog-Digital-Konverter arbeiten in der Regel mit 24 bit Auflösung.

Sample

Einzelner Wert einer diskret abgetasteten Wellenform (siehe auch PCM) - in gewisser Weise das Gegenstück der digitalen Signalverarbeitung zum Pixel in digitalen Bildern.

Quellen

[1] Will Pirkle. Designing Audio Effect Plug-Ins in C++. Focal Press, Burlington MA, 2013.

[2] Thomas Görne, Prof. Dr. Ulrich Schmidt (Hrsg.). Tontechnik. Carl Hanser Verlag, München, 2008.

[3] Heinrich Kuttroff. Akustik - Eine Einführung. S. Hirzel Verlag, Stuttgart, 2004.

[4] William H. Press, Saul A. Teukolsky, William T. Vettering, and Brian P. Flannery. Numerical Recipes Example Book (C++) - The Art of Scientific Computing. Cambridge University Press, Cambridge, 2. aufl. edition, 2002.

[5] OpenCL 1.1 Reference Pages. http://www.khronos.org/registry/cl/sdk/1.1/docs/man/xhtml/

[6] VST SDK 2.4 Documentation. http://www.steinberg.net/en/company/developer.html (In den entsprechenden SDKs enthalten)

[7] Davide Andrea Mauro. Audio convolution by the mean of GPU: CUDA and OpenCL implementations. Proceedings of the Acoustics 2012 Nantes Conference.

[8] Fons Adriaensen. Design of a Convolution Engine optimised for Reverb. 4th International Linux Audio Conference, 2006.