Universität Hamburg
Fakultät für Mathematik,
Informatik und Naturwissenschaften
Department Informatik

Diplomarbeit

Morphing der Klangfarbe von Musikinstrumenten durch Spektralmodellierung

Stefan Westerfeld

stefan@space.twc.de
Studiengang Informatik
Matr.-Nr. 5108388
Fachsemester 34

Erstgutachter Universität Hamburg:	Prof. Dr.-Ing. Wolfgang Menzel
Zweitgutachter Universität Hamburg:	Prof. Dr. Rolf Bader

Inhaltsverzeichnis

1 Einleitung

Ziel dieser Arbeit ist die Beschreibung eines als Software implementierbaren Verfahrens zum Morphing der Klangfarbe von Musikinstrumenten. Damit ist es möglich, Klänge zu erstellen, die von der Klangfarbe eines Instruments zur Klangfarbe eines anderen Instruments flüssig übergehen. Beispielsweise könnte ein Ton generiert werden, der anfangs wie ein Trompete klingt, sich dann aber graduell in den Klang einer Geige verwandelt (Morphing). Es ist damit ebenfalls möglich, Hybridinstrumente zu erstellen, deren Klangfarbe von beiden Ausgangsinstrumenten beeinflusst wird. Dabei wird immer vorausgesetzt, dass eine Aufnahme eines einzelnen Tons in der passenden Tonhöhe der jeweiligen Ausgangsinstrumente vorhanden ist.

Uns war dabei sehr wichtig, dass als Ausgangsmaterial Aufnahmen (Samples) der zu morphenden Instrumente verwendet werden sollten, sodass die Klangfarbe nicht programmiert oder modelliert werden muss, sondern aus vorliegenden Daten ermittelt wird. Grundsätzlich sollte gelten: sobald eine Aufnahme eines natürlichen Instruments vorhanden ist, kann unser Verfahren auch einen Übergang konstruieren.

Aufgrund der gegebenen Anforderungen scheiden einige klassische Verfahren zur Erzeugung von Klängen prinzipiell aus. Beispielsweise kann man zwar mittels physikalischer Modellierung (Physical Modelling) realistische Klänge einzelner Instrumente erstellen. Dies erfordert jedoch instrumentenspezifische Modellierung, kann also nicht wie von uns beabsichtigt aus einer Beispielaufnahme passieren. Physikalische Modelle eignen sich zwar zur Erstellung naturgetreuer Klänge, sind jedoch prinzipbedingt nicht so aufgebaut, dass ein Morphing der Klangfarbe mehrerer physikalischer Modelle möglich ist.

Einige weitere Verfahren zur Klangerzeugung sind die FM-Synthese, subtraktive Synthese und ähnliche, die den Klang komplett synthetisch erstellen. Hier sind zwar oftmals Übergänge zwischen einzelnen Klängen möglich, da die Parameter der Klangerzeugung langsam variiert werden können, zum Beispiel die Grenzfrequenz eines Tiefpassfilters oder die Pulsbreite eines Oszillators. Soll der Klang eines natürlichen Instruments imitiert werden, ist jedoch manuelle Modellierung nötig. Modelle verschiedener Instrumente verwenden dann nicht mehr unbedingt den gleichen Parameterraum, wodurch Morphing vollständig synthetischer Klänge wiederum oftmals nicht möglich ist.

Als letzte Alternative sei noch die direkte Verwendung der Samples angesprochen. Für einen Übergang einer Trompete zu einer Geige könnte man einen Crossfade verwenden, also das Trompetensample langsam ausblenden und das Geigensample langsam einblenden. Typischerweise enthalten beide Samples hierbei Teiltöne, die in etwa den Vielfachen der Grundfrequenz entsprechen. Diese haben jedoch im allgemeinen nicht die gleiche Phasenlage. Dadurch kommt es beim Überblenden zur teilweisen Auslöschung von Teiltönen, was je nach Einzelfall deutlich störend auffallen kann.

Unser Verfahren zum Morphing der Klangfarbe von Instrumenten basiert auf Spektralmodellierung. Bei dem zuvor erwähnten Crossfade und anderen auf Samples basierenden Methoden wird die aufgenommene Wellenform eines Instruments im Zeitbereich direkt verwendet. Stattdessen betrachten wir das Signal nicht im Zeitbereich, sondern im Frequenzbereich. Uns interessiert dabei das Spektrum, und dieses wird analysiert und modelliert. Diese Vorgehensweise korrespondiert zum menschlichen Hörvorgang, bei dem ebenfalls eine Zerlegung des Signals in ein Spektrum stattfindet, und anhand dieses Spektrums die Wahrnehmung der Klangfarbe entsteht.

Da sich die Klangfarbe eines Musikinstruments im Laufe der Zeit langsam verändert, ist es nicht sinnvoll, nur ein einzelnes Spektrum für ein Sample zu berechnen. Stattdessen betrachten wir viele einzelne Kurzzeit-Spektren, die jeweils den Klang zu einem bestimmten Zeitpunkt beschreiben. Diese werden zerlegt, sodass die einzelnen Teiltöne identifiziert werden, und der verbleibende Ge-

räuschanteil ermittelt wird. Wir orientieren uns in der Vorgehensweise an der Spectral Modelling Synthesis, die in [Ser89] sowie [SS90] beschrieben wird.

Um ein Morphing von zwei oder mehr Instrumenten zu erstellen, werden die Analysedaten der Ausgangsklänge geeignet kombiniert. Um die resultierende Klangfarbe an einem bestimmten Zeitpunkt zu ermitteln, wird jeweils das Modell jeweils genau eines Spektrums jedes Ausgangsinstruments verwendet, welches beschreibt, wie die Klangfarbe dieses Instruments an dieser Stelle ist. Aus diesen spektrales Modellen kann die vom Morphing erzeugte Klangfarbe als gewichtete Kombination der jeweiligen Teiltöne und Geräuschanteile berechnet werden.

In Kapitel 2 erläutern wir den Analysevorgang detailliert. Es wird beschrieben, wie wir die Aufnahme eines Instrumententons in einzelne kurze Frames aufspalten, aus denen wir mittels Fourier-Transformation Kurzzeit-Spektren berechnen. Diese zerlegen wir in die Teiltöne und in den Geräuschanteil, womit wir die Klangfarbe an einem bestimmten Zeitpunkt modelliert haben. Wir beschreiben auch weitere Anpassungen, wie die Behandlung des Einschwingvorgangs oder die Definition von Loops.

In Kapitel 3 wird die Synthese beschrieben, mit der wir aus den Analysedaten eines Klangs wieder ein Sample berechnen können. Im wesentlichen wird dieses aus den spektralen Modellen der bei der Analyse berechneten Kurzzeit-Spektren zusammengesetzt, diese bestehen jeweils aus Teiltönen und Geräuschanteil. Hierbei legen wir Wert darauf, diese Berechnungen effizient durchzuführen, damit die Synthese Teil von Echtzeitanwendungen sein kann.

In Kapitel 4 findet sich eine Erklärung, wie zwei oder mehrere Instrumente durch Morphing kombiniert werden können. Hierbei werden die zeitlich korrespondierenden spektralen Modelle der Klangfarbe der Ausgangsinstrumente jeweils gewichtet zum Modell der Klangfarbe des Resultats zusammengesetzt. Wir gehen auf mehrere alternative Verfahren für das Morphing und auf die Steuerung des Morphings durch den Benutzer ein. Neben der Synthese ist auch das Morphing echtzeitfähig.

In Kapitel 5 beschreiben wir einen Hörversuch, den wir mit Versuchspersonen durchgeführt haben. Hierbei wurden verschiedene Hörbeispiele generiert und durch die Testteilnehmer bewertet, um die Qualität der Analyse/Synthese und der verschiedenen Morphingverfahren zu ermitteln.

Auf der zu dieser Arbeit gehörenden CD finden sich alle Audiobeispiele, die im Hörversuch präsentiert und bewertet wurden. Für eine detaillierte Beschreibung des Audiomaterials verweisen wir auf Kapitel 5, eine grobe Übersicht findet sich in Kapitel 5.1.6.

2 Analyse der Samples

2.1 Überblick über das Analyseverfahren

2.1.1 Grundlagen der Analyse

Die additive Synthese ist eines der ersten Verfahren der Klangsynthese durch Computer. Dabei wird der Klang als Summe von Sinusschwingungen mit verschiedenen Frequenzen und Amplituden dargestellt. Damit die nötigen Parameter aus Aufnahmen von Instrumentklängen bestimmt werden können, wurde von [Moo73] der „hetrodyne filter“ entwickelt. Dieser bestimmt die Parameter allerdings nur zuverlässig, wenn keine zu schnelles Einschwingen (Attack) vorliegt, die Frequenz der Teiltöne nicht stark von ganzzahligen Vielfachen der Grundfrequenz abweichen, sowie keine starken Frequenzschwankungen vorliegen (wie etwa beim Vibrato).

Ein anderes Fundament für die Analyse bietet der Phasenvocoder, der in [FG66] beschrieben wird. Der Kern ist die Bestimmung von Kurzzeit-Amplituden- und Kurzzeit-Phasen-Spektren mittels einer Kurzzeit-Fourier-Transformation. Später in [Por76] wird die Analyse auf Basis der schnellen Fourier Transformation („FFT“) beschrieben, was vor allem ein Effizienzvorteil ist.

In [Moo78] wird eine auf dem Phasenvocoder aufbauende Signalanalyse beschrieben. Aus den einzelnen Parametern der Kurzzeit-Fourier-Transformation können so Frequenz- und Amplitudeninformation extrahiert werden, die später als Eingabe für die additive Synthese verwendet werden können.

Probleme ergeben sich bei diesem Ansatz wenn Teiltöne in der Frequenz schwanken („Vibrato“), da sie dann zwischen den einzelnen FFT-Bins springen. Weiterhin sind inharmonische Klänge zwar analysierbar (beispielsweise Klavieraufnahmen), jedoch kann es passieren, dass Teiltöne zwischen zwei FFT-Bins liegen.

Schließlich bildet weder der obengenannte „hetrodyne filter“ noch der Phasenvocoder eine gute Basis um einen Geräuschanteil zu modellieren, etwa das durch den Luftstrom verursachte Rauschen bei einem Flötenklang. Diese Nachteile werden von neueren Analyseverfahren behoben.

Eine bessere Analyse bieten die eigentlich für Sprache entwickelten Verfahren [MQ84] und [MQ86], die die Kurzzeit-Fourier-Transformation als Basis verwenden, und wie wir in dieser Arbeit in den einzelnen Frames nach lokalen Maxima („Peaks“) suchen. Diese entsprechen dann Teiltönen, die von Frame zu Frame verfolgt werden können, sodass die Analyse/Synthese frei von den Beschränkungen des Phasenvocoders ist.

Nahezu zeitgleich entwickelt und vom Aufbau her vergleichbar ist PARSHL [SS87]. Auch hier findet nach einer Kurzzeit-Fourier-Transformation eine Suche nach lokalen Maxima statt, diese entsprechen den Teiltönen, die von Frame zu Frame verfolgt werden.

Eine letzte für diese Arbeit relevante Verbesserung findet sich in [Ser89] sowie [SS90]. Die Modellierung der Sinusanteile der Frames verläuft wie in PARSHL. Jedoch werden diese aus den Kurzzeit-Spektren subtrahiert. Vom verbleibenden Restspektrum wird angenommen, dass es dem Geräuschanteil („Noise“) entspricht. Dieser wird separat gespeichert, sodass bei der Wiedergabe beispielsweise das Anblasgeräusch einer Flöte erhalten bleibt.

Die von uns implementierte und nachfolgend beschriebene Analyse basiert ebenso auf der Zerlegung des Signals in Frames. Für jeden Frame werden die Sinusanteile bestimmt, indem nach lokalen Maxima im Spektrum gesucht wird. Diese Teiltöne werden aus dem Spektrum subtrahiert. Der

Rest wird als Noise betrachtet und für jeden Frame als Noisekoeffizienten repräsentiert. Schließlich beschreiben wir für den Anfang des Tons eine Zeitbereichsanpassung, die vor allem bei schnellem Einschwingen („Klavier“) eine Verbesserung der Klangqualität erzielt.

2.1.2 Das Ausgangsmaterial für die Analyse

Die Analysephase dient dazu, die Daten zu ermitteln, die später für das Morphing von Klängen verwendet werden können. Für jedes Instrument werden Samples der einzelnen Töne benötigt. Typischerweise wird jeder Ton, der auf dem Instrument gespielt werden kann, einmal mit einem Mikrophon aufgenommen. Jede Aufnahme wird einer Midi-Note (und damit auch einer Grundfrequenz) zugeordnet.

Für die Trompete sind die Einzelaufnahmen in Tabelle 2.1 dargestellt.

Midi-Note	**Aufnahme**
...	...
60	Sample Trompete C-4
61	Sample Trompete C#4
62	Sample Trompete D-4
...	...

Tabelle 2.1: Aufnahmen der Einzeltöne der Trompete

Um später eine gute Qualität bei der Analyse zu erreichen, empfiehlt es sich die einzelnen Aufnahmen mindestens in CD-Qualität zu erstellen, also mit einer Abtastrate von 44100 Hz und 16 Bit pro Samplewert. Es werden Aufnahmen eines einzelnen Mikrophons verwendet, sodass die Anzahl der Kanäle für die Analyse immer eins ist (Mono-Aufnahmen).

2.1.3 Zerlegung des Signals in Frames

Der erste Schritt der Analyse ist die Zerlegung des Signals in überlappende Frames. Ein Frame enthält jeweils einen 40 ms langen Ausschnitt des aufgenommenen Signals. Bei tiefen Tönen werden längere Frames verwendet. Dies passiert um sicherzustellen, dass die einzelnen lokalen Maxima im Spektrum noch sauber voneinander getrennt werden können.

Nach der Zerlegung enthält jeder Frame nur einen kurzen Ausschnitt des Ausgangssignals, sodass sich die Klangfarbe innerhalb eines Frames idealerweise nicht oder kaum verändert.

Abbildung 2.1 zeigt einen Frame in der Mitte der Aufnahme für die Trompete auf Note C-4 (Midi-Note 60). Man sieht, dass der Klang in dieser Phase fast einem periodischen Signal entspricht. Daher lässt sich ein Frame nahezu vollständig als Summe von Sinusschwingungen darstellen.

2.1.4 Spektrales Modell eines Frames

Sei ein Frame vor der Analyse gegeben als ein Vektor von N Abtastwerten

$$x(0), x(1), ..., x(N-1)$$

so ist unser Ziel eine Zerlegung des Frames in einen deterministischen und einen stochastischen Anteil ($d(t)$ und $e(t)$), sodass

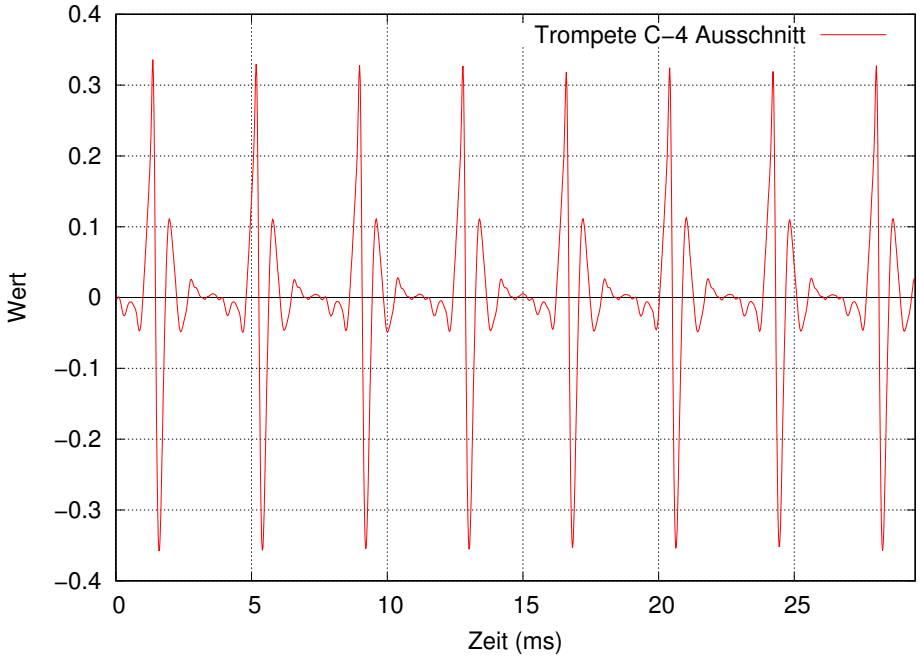


Abbildung 2.1: Ausschnitt aus einer Trompetenaufnahme „C-4“

$$\begin{aligned} x(t) &= d(t) + e(t) \\ &= \sum_{p=1}^{P} A_p cos\left(2\pi \frac{F_p}{F_s} + \Phi_p\right) + e(t) \end{aligned}$$

wobei $e(t)$ den Anteil des Signals enthält, der sich nicht als Summe von Sinusschwingungen darstellen lässt.

Der deterministische Anteil besteht bei dieser Darstellung aus P Sinusschwingungen, wobei jede einzelne davon durch eine Amplitude A_p, eine Frequenz F_p und eine Phase Φ_p beschrieben wird. In den folgenden Kapiteln wird gezeigt, wie die Parameter A_p, F_p und Φ_p durch Analyse des Ausgangssignals berechnet werden können. Außerdem wird die weitere Analyse des stochastischen Anteils $e(t)$ erklärt, der sich durch Modellierung des Spektrums von $e(t)$ auch parametrisch beschreiben lässt.

Unter der Annahme, dass $e(t)$ nur noch den Geräuschanteil oder Rauschanteil des Frames enthält, kann man weiter $e(t)$, den stochastischen Anteil als frequenzabhängiges Rauschen modellieren. Dadurch kann der Klang des gesamten Ausgangssignals als Klang von aufeinanderfolgenden Frames mit einem deterministischen Anteil aus Sinusschwingungen und einem stochastischen Anteil, dem frequenzabhängigen Rauschen modelliert werden.

2.2 Mathematische Grundlagen

2.2.1 Die kontinuierliche Fourier-Transformation

Die kontinuierliche Fourier-Transformation ordnet jeder kontinuierlichen Frequenz ω einen komplexen Wert $X(\omega)$ zu. Die zu analysierende Funktion $x(t)$ ist hierbei ebenfalls kontinuierlich.

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$$

Die Fourier-Transformation ist invertierbar, sodass aus der Fouriertransformierten $X(\omega)$ wieder die Funktion $x(t)$ gewonnen werden kann.

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}d\omega$$

Für die Analyse der Aufnahmen bietet die kontinuierliche Fourier-Transformation zwei wichtige Eigenschaften. Einerseits wird die Funktion $x(t)$ als Kombination von periodischen Funktionen mit der Frequenz ω dargestellt. Eine solche Darstellung ist später für das Morphing entscheidend, da diese Repräsentation dem menschlichen Hören eher entspricht, als die aufgenommene Schwingungsfunktion $x(t)$ selbst. Die andere Eigenschaft ist, daß aus der Funktion $X(\omega)$ die Originalfunktion, in diesem Fall sogar ohne Verlust von Information, wieder ermittelt werden kann. Diese Eigenschaft ist für das Abspielen der Klänge notwendig.

Allerdings gibt es auch zwei Probleme mit der kontinuierlichen Fourier-Transformation, weswegen wir sie nicht direkt zur Analyse verwenden können. Wie beschrieben ist $x(t)$ eine kontinuierliche Funktion. Das heißt, jedem kontinuierlichen Zeitwert t wird ein Funktionswert $x(t)$ zugeordnet. In der Praxis stehen uns aber durch die Zeitquantisierung bei der Aufnahme der Daten nur 44100 Abtastwerte (entsprechend der Samplefrequenz) zur Verfügung. Daher wird im nächsten Kapitel die diskrete Fourier-Transformation eingeführt, die diesem Umstand gerecht wird.

Ein weiteres Problem ist, daß die kontinuierliche Fourier-Transformation immer eine Frequenzzerlegung der ganzen Funktion vornimmt. In der Praxis verändern sich aber die Eigenschaften des Klanges eines Instruments mit der Zeit. Die Amplituden der Sinusschwingungen aus denen der Klang zusammengesetzt ist und auch die Frequenzen dieser Sinusschwingungen sind nicht für die gesamte Aufnahme gleich, weswegen die Analyse viele kleine Ausschnitte des Signals einzeln transformiert. Diese Technik, die Kurzzeit-Fourier-Transformation, wird im Kapitel 2.2.5 beschrieben.

2.2.2 Die diskrete Fourier-Transformation DFT

Liegt eine Funktion bereits als Vektor von N Abtastwerten $x(0), x(1), ..., x(N-1)$ vor, kann die diskrete Fourier-Transformation wie folgt berechnet werden:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-2\pi j\frac{nk}{N}}$$

Hierbei entstehen für einen Vektor x der Länge N ein Vektor X mit genau N komplexen Werten. Die diskrete Fourier-Transformation ist ebenfalls invertierbar. Um den Vektor x aus dem Vektor X zu erhalten, kann die Formel

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{2\pi j\frac{nk}{N}}$$

verwendet werden. Falls der Ausgangsvektor x nur aus reellen Zahlen besteht, was in unserer Anwendung der Fall ist, gilt zusätzlich

$$X(N-k) = \overline{X(k)}, X(0) \in \mathbb{R}, X(N/2) \in \mathbb{R}$$

Dadurch sind die Ergebnisse der zweiten Hälfte von $X(k)$ nur die konjugiert komplexen Werte der ersten Hälfte, und können für die Analyse ignoriert werden. Viele FFT Implementationen bieten daher eine gesonderte Funktion zum Berechnen einer reellen FFT/IFFT (siehe nächstes Kapitel) an, die entsprechend schneller berechnet werden kann als die komplexe Version.

Die Ergebnisse der diskreten Fourier-Transformation $X(k)$ - wir betrachten hier nur die erste Hälfte, bis einschließlich $X(N/2)$ - lassen sich in Amplituden und Phasen einer zugehörigen Sinusschwingung interpretieren. Um die Amplitude zu erhalten, verwendet man den komplexen Betrag und normiert ihn. Der Normierungsfaktor ist deshalb $N/2$ und nicht N, da wir hier nur die erste Hälfte der DFT verwenden. Sei

$$X(k) = a + bi$$

so ergibt sich die Amplitude der Sinusschwingung als

$$A(k) = \frac{|X(k)|}{N/2} = \frac{\sqrt{a^2 + b^2}}{N/2}$$

Die Phase der Sinusschwingung erhält man mit folgender Formel:

$$\Phi(k) = arg(X(k)) = atan2(b, a)$$

wobei $atan2$ die C-Funktion ist, die das komplexe Argument aus dem Real- und Imaginärteil berechnet. Zusammenfassend kann man sagen, dass jedem Wert k eine Sinusschwingung der Form

$$x_k(n) = A(k)cos\left(2\pi\frac{nk}{N} + \Phi(k)\right)$$

zugeordnet ist, und dass sich das Ausgangssignal als Summe aller dieser Einzelschwingungen ergibt.

$$x(n) = \sum_{k=0}^{N/2} x_k(n) = \sum_{k=0}^{N/2} A(k)cos\left(2\pi\frac{nk}{N} + \Phi(k)\right)$$

Bei dieser Analysemethode werden nur Frequenzen berücksichtigt, die genau zur Länge der DFT periodisch sind. Diese Einschränkung wird später im Kapitel 2.2.5 über die Kurzzeit-Fourier-Transformation aufgehoben.

2.2.3 Die schnelle Fourier-Transformation FFT

Um die diskrete Fourier-Transformation zu berechnen, müssen N Summen $X(0)...X(N-1)$ berechnet werden, die jeweils aus N Elementen bestehen. Daher hat die diskrete Fourier-Transformation eine Berechnungskomplexität von $O(N^2)$. Für viele Fälle kann eine schnelle Fourier-Transformation

verwendet werden, um Rechenzeit zu sparen. Im Rahmen dieser Arbeit reicht bereits der Spezialfall, in dem N eine Zweierpotenz ist, also $N = 2^k$. In diesem Fall kann das Ergebnis mit einer Berechnungskomplexität von $O(N \log N)$ ermittelt werden.

Es gibt verschiedene Möglichkeiten, die FFT eines Eingangsvektors (oder deren inverse Transformation, oft IFFT genannt) zu berechnen. Welche davon die schnellste ist hängt von verschiedenen Faktoren ab, etwa der verwendeten CPU, der Verfügbarkeit von SIMD Instruktionen wie SSE, SSE2,... der Größe des Eingangsvektors und dem verwendeten Compiler. Die für die Implementation genutzte fftw-Bibliothek (`www.fftw.org`) prüft daher in einer einmal erforderlichen Planungsphase vor dem Ausführen der FFT verschiedene in der Bibliothek enthaltene Algorithmen auf ihre Geschwindigkeit, sodaß nicht nur die Komplexität von $O(N \log N)$ garantiert ist, sondern auch der für das Zielsystem beste vorhandene Algorithmus verwendet wird. Eine Beschreibung der Details findet sich in [FJ05].

2.2.4 Was die fftw-Bibliothek wirklich berechnet

Es gibt verschiedene Verfahren wie FFT Implementationen die Daten normieren. Damit die Anwendung einer FFT und einer inversen FFT nacheinander das identische Ergebnis erzeugt, kann bei der Rücktransformation durch die Anzahl der Werte N geteilt werden. So haben wir es in Kapitel 2.2.2 beschrieben. Eine andere Alternative die Hin- und Rücktransformation zu normieren wäre jeden Schritt durch $\sqrt{N}$ zu teilen.

Die fftw-Bibliothek verwendet keine dieser Normierungen, sondern berechnet (in unserer Notation) als FFT

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-2\pi j \frac{nk}{N}}$$

und als Rücktransformation

$$y(n) = \sum_{k=0}^{N-1} X(k) e^{2\pi j \frac{nk}{N}}$$

Damit ergibt die die Anwendung der FFT auf das Signal $x(n)$ und danach die Anwendung der inversen FFT nicht das Originalsignal. Die Eingabe wird stattdessen dabei mit N multipliziert.

$$y(n) = Nx(n)$$

Wenn im folgenden Normierungsfaktoren beschrieben werden, die sich auf die Analyse oder Synthese mittels FFT beziehen, werden diese immer für die fftw-Bibliothek passend gewählt.

2.2.5 Die Kurzzeit-Fourier-Transformation STFT

Während die oben beschriebenen Transformationen für ein Eingangssignal $x(n)$ eine einzige Transformierte $X(k)$ für das ganze Signal berechnet, ändert sich die Zusammensetzung eines Instrumentenklangs normalerweise mit der Zeit. Für die Analysephase bedeutet dies, dass verschiedene Spektren zu verschiedenen Zeitpunkten erstellt werden sollen, bei denen jeweils nur ein Ausschnitt der Ausgangsfunktion $x(n)$ analysiert wird. Eine solche zeitabhängige Analyse basierend auf der Fourier-Transformation wird in [All77] beschrieben. Die Kurzzeit-Fourier-Transformation kann man definieren als

$$X_l(k) = \sum_{n=0}^{N-1} w(n) x(n + lH) e^{-2\pi j \frac{nk}{N}}, l = 0, 1, ...$$

Statt einer einzigen Ausgabe für die gesamte Funktion $x(n)$ erhält man mit der Kurzzeit-Fourier-Transformation eine Folge $X_0(k), X_1(k), ...$; jedes dieser Spektren wurde nur aus einem Ausschnitt des Signals $x(lH), x(lH+1), ..., x(lH+N-1)$ berechnet. Es gibt drei Parameter (außer der Eingangsfunktion $x(n)$), die das Ergebnis der STFT beeinflussen.

Die Länge des Analysewindows N beeinflusst wie lang die Ausschnitte des Signals sind, die zur Berechnung des Spektrums herangezogen werden. Je kleiner N gewählt wird, desto höher wird die Zeitauflösung der Analyse. Allerdings sinkt damit auch die Frequenzauflösung.

Der Abstand der einzelnen Ausschnitte H („Hopsize“) beeinflusst wie weit die Ausschnitte aus der Funktion $x(n)$ voneinander entfernt sind.

Die Fensterfunktion $w(n)$ („window function“) legt fest, wie ein Ausschnitt aus der Funktion $x(n)$ ermittelt wird. Fast immer wird eine Fensterfunktion verwendet, die an den Rändern gegen 0 geht, und in der Mitte etwa 1 ist. Diese Funktion sorgt für eine Eingrenzung der Analyse in der Zeit auf den durch den Parameter l gegebenen Bereich.

2.2.6 Der Einfluss der Fensterfunktion auf die STFT

Was bewirkt die Wahl der Fensterfunktion $w(n)$ beim Versuch, ein Signal welches aus Sinuskomponenten besteht, zu zerlegen? Um das zu beantworten, folgen wir der mathematischen Darstellung aus [Ser89]. Wir nehmen an, dass das zu analysierende Signal aus einer einzelnen komplexen Sinusschwingung besteht:

$$x(n) = Ae^{j\omega_x n}$$

Die Fensterfunktion $w(n)$ sei gegeben, zusammen mit ihrer Fourier-Transformation

$$W(\omega) = \sum_{n=0}^{N-1} w(n)e^{-j\omega n}$$

Nun berechnen wir die DFT von $x(n)$:

$$\begin{aligned} X(\omega) &= \sum_{n=0}^{N-1} w(n)x(n)e^{-j\omega n} \\ &= \sum_{n=0}^{N-1} w(n)Ae^{j\omega_x n}e^{-j\omega n} \\ &= A\sum_{n=0}^{N-1} w(n)e^{-j(\omega-\omega_x)n} \\ &= AW(\omega-\omega_x) \end{aligned}$$

Um den Einfluss der Fensterfunktion zu untersuchen, reicht es also aus, die Fourier-Transformierte des Fensters zu betrachten. Jede Sinusschwingung im Eingangssignal taucht als verschobene Version der transformierten Fensterfunktion im Ergebnis-Spektrum der STFT auf. In Abbildung 2.2 sieht man verschiedene Fensterfunktionen im Zeitbereich und in Abbildung 2.3 ihre Transformierte im Frequenzbereich.

Die Sinusschwingung erzeugt mit jeder Fensterfunktion auf ein Hauptmaximum, welches bei der STFT an der Frequenz der Sinusschwingung verschoben wird. Außer dem Hauptmaximum gibt es aber noch zahlreiche Nebenmaxima, sodass jede Sinuskomponente im Eingangssignal Auswirkungen auf das gesamte Spektrum hat.

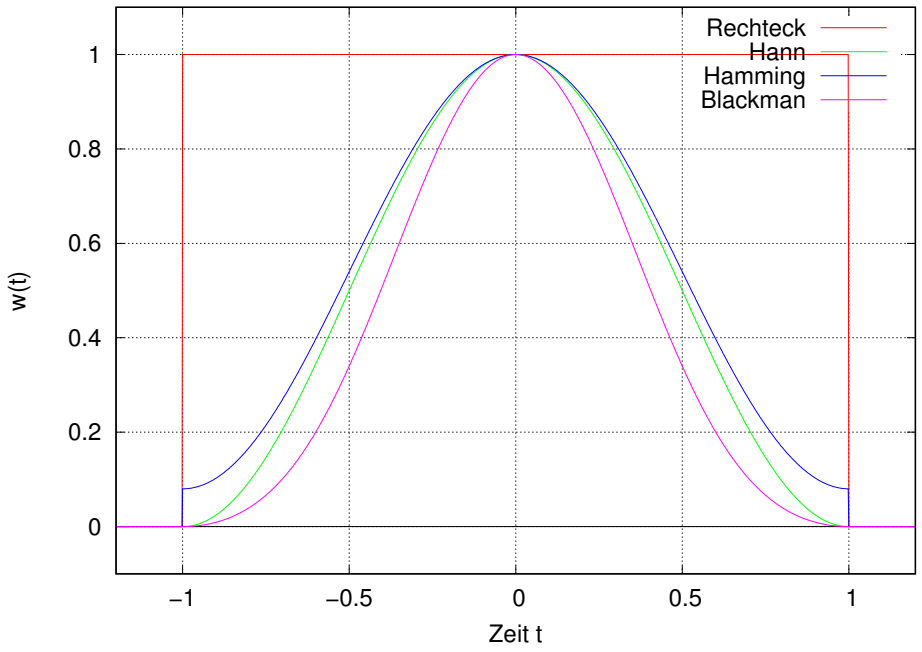


Abbildung 2.2: Verschiedene Fensterfunktionen im Zeitbereich

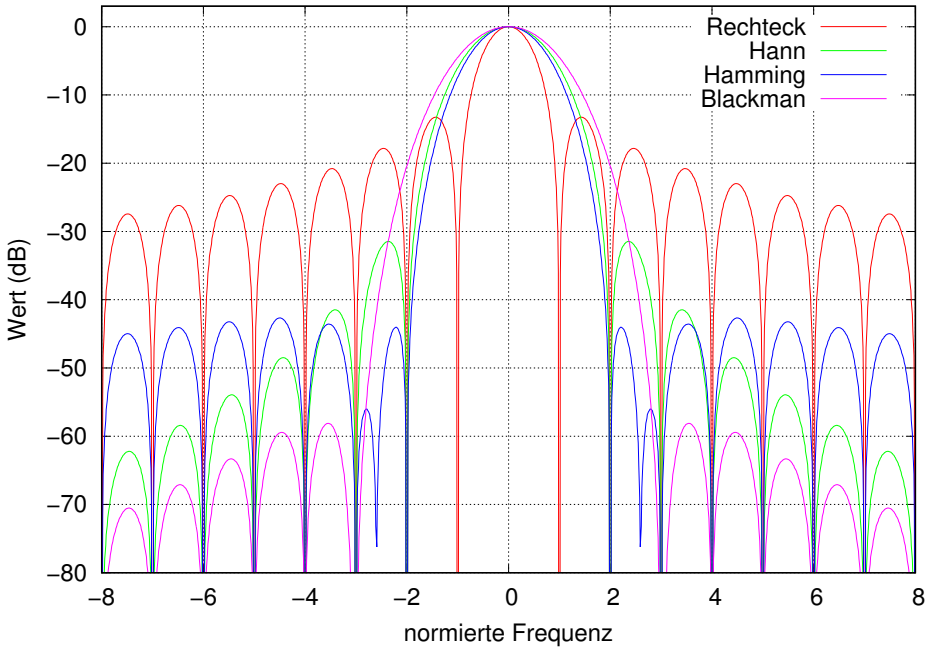


Abbildung 2.3: Fensterfunktionen im Frequenzbereich

Zwei Dinge sind für die Analyse wünschenswert, damit ein Signal mit mehreren Sinuskomponenten gut analysiert werden kann. Einerseits sollte das Hauptmaximum möglichst schmal sein, damit Sinuskomponenten mit ähnlichen Frequenzen sich im STFT Spektrum nicht gegenseitig überlagern. Andererseits sind niedrige Nebenmaxima vorteilhaft, da auch diese die Analyse von Sinuskomponenten mit ähnlichen Frequenzen stören.

Das einfachste Fenster, das Rechteckfenster, hat das schmalste Hauptmaximum, allerdings hat das erste Nebenmaximum -13 dB, liegt also am höchsten von allen Fensterfunktionen. Die Breite des Hauptmaximums ist beim Hann-Fenster und Hamming-Fenster identisch und größer als beim Rechteckfenster. Dafür liegt das erste Nebenmaxima bei $-31{,}5$ dB bzw. $-41{,}5$ dB. Das Blackmanfenster hat das breiteste Hauptmaximum, jedoch liegt das erste Nebenmaximum auch mit -58 dB

am tiefsten von allen betrachteten Fensterfunktionen.

Es gibt viele weitere Fensterfunktionen, eine gute Übersicht findet sich in [Har78]. In [Ser89] wird vorgeschlagen, das parametrisierbare Kaiserfenster zu verwenden, weil dort ein Parameter α eingeht, der die Breite des Hauptmaximums und die Höhe des ersten Nebenmaximums steuert. Je größer α gewählt wird, desto niedriger liegt das erste Nebenmaximum und desto breiter wird das Hauptmaximum.

2.3 Analyse der einzelnen Frames

2.3.1 Wahl der STFT Parameter

Der erste Schritt der Analyse des Signals besteht in der Anwendung der Kurzzeit-Fourier-Transformation STFT, die in Kapitel 2.2.5 beschrieben wurde. Die dabei verwendete Framelänge M, Hopsize H und Fensterfunktion $w(n)$ ist entscheidend für die weitere Verarbeitung.

Als Fensterfunktion $w(n)$ verwenden wir das Hann-Fenster. Dieses hat ein relativ schmales Hauptmaximum. Die Höhe der Nebenmaxima fällt (anders als beim Hamming-Fenster mit vergleichbar schmalem Hauptmaximum) mit zunehmender Entfernung vom Hauptmaximum ab. Es wären viele andere Fensterfunktionen denkbar, aber Tests mit unseren Eingangsdaten ergaben gute Ergebnisse mit einem Hann-Fenster.

Die nächste Aufgabe ist die Wahl der Framelänge M. Durch diese wird bestimmt, welchen Kompromiss zwischen Zeitauflösung und Frequenzauflösung wir eingehen. Je besser die Zeitauflösung (je kleiner M), desto schlechter ist die Frequenzauflösung.

In Abbildung 2.4 zeigt den Einfluss verschiedener Framelängen auf die Zeit-Frequenz-Zerlegung, die von der STFT vorgenommen wird. Die analysierte Gesangsaufnahme hatte eine Grundfrequenz von 129,2 Hz, sodass wir Teiltöne etwa bei $k \cdot 129{,}2$ Hz, $k = 1{,}2, ...$ erwarten. Diese wiederum zeigen sich als von der Fensterfunktion erzeugte runde Maxima für jeden Teilton.

Bei einer Framelänge von 40 ms sind die Teiltöne gut voneinander zu trennen. Wird die Zeitauflösung durch kleinere Framelängen erhöht, sinkt die Frequenzauflösung: bei eine Framelänge von 20 ms überlappen die Teiltöne bereits, und bei einer Framelänge von 10 ms lässt sich gar nicht mehr von unterscheidbaren Teiltönen sprechen.

Versuchen wir andererseits die Frequenzauflösung zu erhöhen, indem wir längere Framelängen von 80 ms, 160 ms oder 320 ms verwenden, verlieren wir mit steigender Framelänge die Fähigkeit die leichten Frequenzschwankungen insbesondere der höheren Teiltöne zu verfolgen. Da die Frequenz in diesem Fall innerhalb des Frames schwanken kann (insbesondere bei sehr großen Framelängen), ergibt die STFT nicht mehr wie erwartet die Transformierte des Fensters mit einem genau bei der Frequenz des Teiltons liegenden Maximum, sondern ein verschmiertes Ergebnis.

Insgesamt ergibt sich eine Framelänge von 40 ms bei dieser Grundfrequenz als gute Wahl für die Framelänge der STFT. Dieser Wert kann bei höheren Grundfrequenzen auch verwendet werden, da dort die Teiltöne auseinander rutschen. Bei tieferen Frequenzen rutschen die Teiltöne jedoch näher und näher zusammen, bis schließlich keine saubere Trennung der Teiltöne mehr möglich ist. Daher verwenden wir bei der Analyse die Formel

$$Framelänge = max\left(40, \frac{1000}{Grundfrequenz} \cdot 4\right) \text{ ms}$$

zur Bestimmung der Framelänge. Bei einer Grundfrequenz von 100 Hz und darunter wird die Framelänge vergrößert, sodass die Trennung der Teiltöne auch bei tieferen Tönen durch eine entspre-

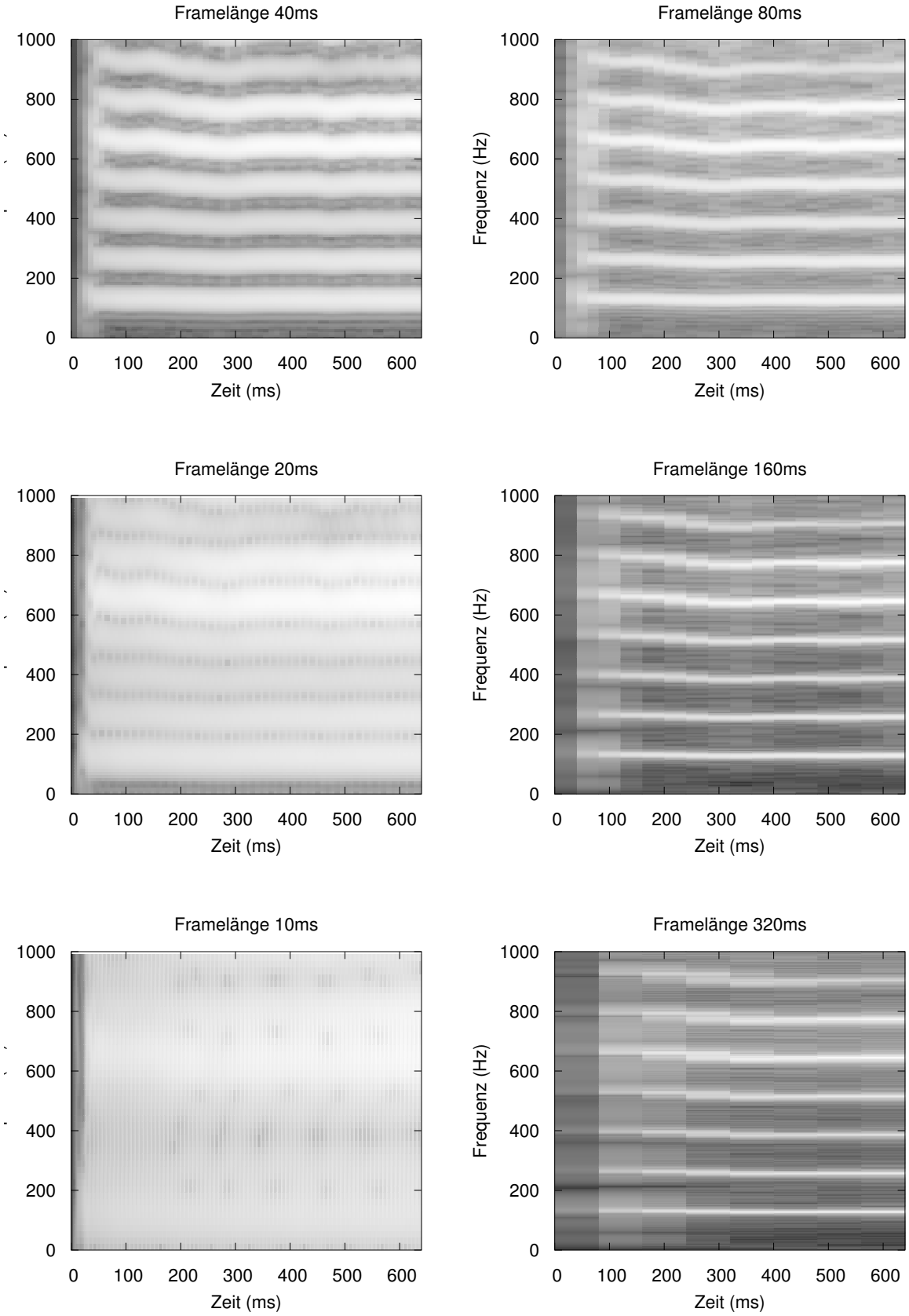


Abbildung 2.4: Einfluss verschiedener Framelängen auf die STFT einer Gesangsaufnahme mit Grundfrequenz 129,2 Hz

chend verlängerte Framelänge gewährleistet bleibt. Durch Umrechnung der Framelänge in Samples erhalten wir M.

Als Hopsize wählen wir

$$H = \frac{M}{4}$$

wodurch die Analyseframes deutlich überlappen, sodass sich die Parameter von Frame zu Frame nicht sehr stark ändern. Auf der anderen Seite ist diese Hopsize ein guter Kompromiss, da die Überlappung nicht so stark ist, dass die Menge der berechneten Analysedaten zu groß wird.

2.3.2 Anwendung der STFT

Die Kurzzeit-Fourier-Transformation übernimmt die Zerlegung des Signals in Frames, sowie die spektrale Analyse jedes einzelnen Frames. In den nächsten Schritten werden die Ergebnisse der STFT so weiterverarbeitet, dass jeder Frame als Summe von P einzelnen Sinusschwingungen mit Amplitude A_p, Frequenz F_p und Phase Φ_p dargestellt wird, sowie der stochastische Anteil bestimmt wird.

Um die STFT wie in Kapitel 2.2.5 beschrieben zu berechnen, wäre eine diskrete Fourier-Transformation (DFT) nötig. Die Länge dieser DFT entspricht hierbei der Länge des Analysefensters. Wir berechnen stattdessen aus zwei Gründen eine größere FFT, in der die verbleibenden Eingangswerte mit Nullen aufgefüllt werden („Zeropadding“).

Einerseits ist die Berechnung der FFT (im Gegensatz zur DFT) immer effizient möglich, da Algorithmen mit der Komplexität $O(N \log N)$ verfügbar sind. Andererseits entspricht das Auffüllen mit Nullen im Zeitbereich einer Interpolation des Spektrums im Frequenzbereich. Dies verbessert das Ergebnis der nachfolgenden Analyse. Die Suche nach der genauen Position eines lokalen Maximums und damit der Frequenz eines Sinusanteils in Kapitel 2.3.3 profitiert von der höheren Frequenzauflösung der Ausgabe der FFT ebenso wie die Bestimmung der Phase dieses Sinusanteils.

Wir bezeichnen die STFT Framelänge mit M und die stattdessen verwendete (größere) FFT Länge mit N. Als Ausgangspunkt für die Bestimmung der Länge der FFT nehmen wir die kleinste Zahl M^*, die größer gleich M ist und eine Zweierpotenz ist.

$$\begin{aligned} M^* &\geq M \\ M^* &= 2^m, m \in \mathbb{N} \\ N &= 4M^* \end{aligned}$$

Diese wird um N zu erhalten nochmals mit einem Faktor, im Rahmen dieser Arbeit immer 4, multipliziert um das Spektrum stärker zu interpolieren.

Um die Verteilung der STFT-Eingangswerte auf die FFT-Eingangswerte zu verstehen, ist es nützlich die Eingangswerte mit einer Zeit zu versehen. Der mittlere Eingangswert, an dieser Stelle ist auch die Fensterfunktion maximal, erhält die Zeit 0. Davor liegende Eingangswerte erhalten negative Zeiten, also $t = -1, -2, ...$ danach liegende Eingangswerte erhalten positive Zahlen, also $t = 1,2, ...$.

Die STFT-Eingangswerte mit positiver Zeit ($t \geq 0$) werden am Anfang der FFT-Eingangswerte gespeichert, die STFT-Eingangswerte mit negativer Zeit ($t < 0$) am Ende der FFT-Eingangswerte, alle anderen Eingangswerte werden mit Nullen aufgefüllt. Damit diese Aufteilung möglich ist, muss es genau einen „mittleren Eingabewert“ ($t = 0$) geben, bei dem die Fensterfunktion maximal ist; um dies sicherzustellen wählen wir die STFT Framelänge M immer so, dass sie ungerade ist.

In Abbildung 2.5 sieht man ein Beispiel eines STFT Frames und in Abbildung 2.6 die zugehörigen FFT Eingabedaten. Insgesamt wird durch diese Anordnung der Werte auch erreicht, dass die

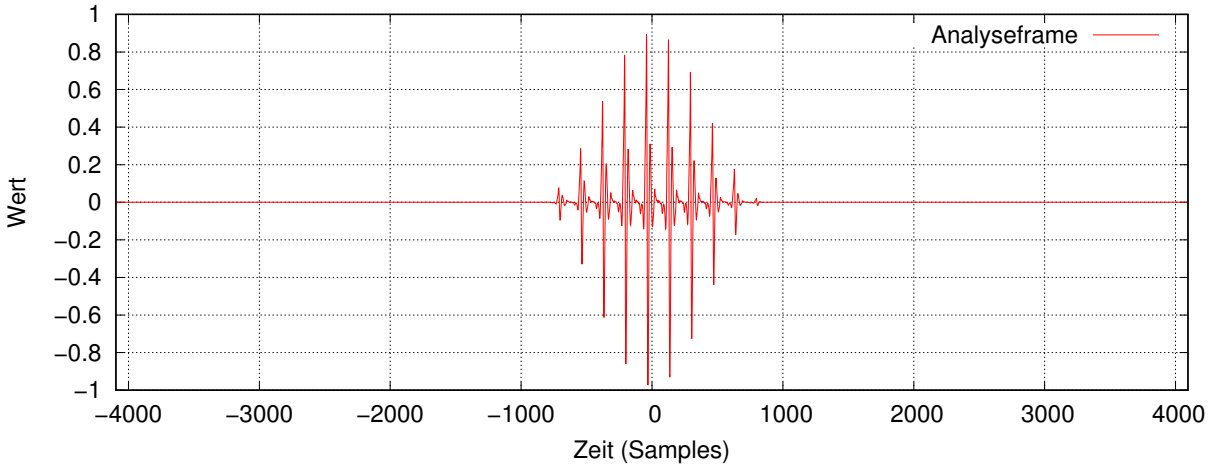


Abbildung 2.5: STFT-Analyseframe mit angewendeter Fensterfunktion

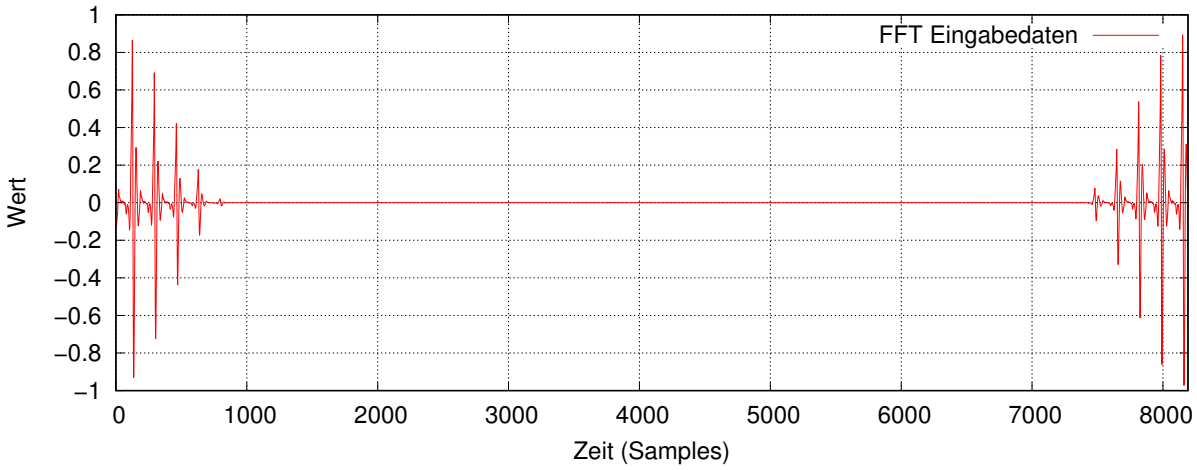


Abbildung 2.6: Zugehörige FFT Eingabewerte

Phasen der Ausgabe der FFT durch das Anwenden der Fensterfunktion nicht beeinflusst werden, sodass aus dem Ergebnis der FFT bei lokalen Maxima nicht nur die Amplitude sondern auch die Phase des entsprechenden Sinusanteils ermittelt werden kann.

2.3.3 Suche nach lokalen Maxima im Spektrum

Nach der Anwendung der Kurzzeit-Fourier-Transformation erhalten wir für jeden Frame ein komplexes Spektrum

$$X(0), X(1), ...X(N-1)$$

In diesem Spektrum wird jetzt nach lokalen Maxima gesucht, da jede im Frame enthaltene Sinuskomponente ein lokales Maximum erzeugt haben sollte. Wir suchen also ein k mit der Eigenschaft

$$|X(k)| > |X(k-1)|$$
$$|X(k)| > |X(k+1)|$$

oder für den Spezialfall, dass das Maximum an einer Stelle auftritt, an der zwei Werte exakt gleich sind, mit der Eigenschaft

$$|X(k)| > |X(k-1)|$$
$$|X(k)| = |X(k+1)|$$
$$|X(k+1)| > |X(k+2)|$$

Nicht jedes so gefundene lokale Maximum gehört tatsächlich zu einem Sinusanteil in dem entsprechenden Block. Es könnte sich auch um ein Nebenmaximum handeln, oder ein zufällig durch Noise entstandenes Maximum.

Um daher die gefundenen Kandidaten einzugrenzen berechnen wir die normierte Höhe des Maximums

$$h = db(|X(k)|) - db(|X_{max}|)$$

wobei X_{max} zur Normierung verwendet wird, und dem maximalen Wert aller $X(k)$ aller Frames entspricht. Für diese Arbeit fordern wir, dass die normierte Höhe $h > -90$ sein muss. Die hierbei verwendete Umrechnung in Dezibel db ist wie folgt definiert:

$$db(x) = 20 \log_{10}(x)$$

Als weiteres Kriterium berechnen wir den Anfang des Maximums als kleinste Zahl k_s und das Ende des Maximums als größte Zahl k_e, sodass gilt:

$$|X(k_s)| \leq |X(k_s + 1)| \leq ... \leq |X(k - 1)| \leq |X(k)|$$
$$|X(k)| \geq |X(k + 1)| \geq ... \geq |X(k_e - 1)| \leq |X(k_e)|$$
$$b = (k_e - k_s)\frac{M}{N}$$

Die Zahl b ist die normierte Breite des Maximums. Es werden nur solche Maxima ausgewählt, deren normierte Breite mindestens einen Wert von 2,9 hat.

Theoretisch ist die normierte Breite unserer Hann-Fensterfunktion exakt 4, sodass ein Signal welches ausschließlich aus einer perfekten Sinusschwingung besteht eine Breite von 4 erreichen würde. In der Praxis wird keine perfekte Sinusschwingung vorliegen; verschiedene Sinusschwingungen beeinflussen sich im Spektrum durch die Anwendung der Fensterfunktion zudem gegenseitig. Zusätzlich kann das Eingabesignal Noise enthalten, und schließlich kann durch die Fourier-Transformation nur die Amplitude der FFT-Bins bestimmt werden, sodass b dadurch entweder etwas kleiner oder etwas größer als die tatsächliche Breite ist. Wir wählen daher die etwas konservativere Schranke von 2,9 als minimale Peakbreite, dies wurde mit verschiedenen Eingabesignalen getestet.

In Abbildung 2.7 ist der Verlauf des Spektrums um ein lokales Maximum eines idealen Sinusanteils dargestellt, so wie er durch die Hann-Fensterung entsteht. Zusätzlich ist unsere untere Schranke von einem normierten Frequenzabstand von 2,9 eingetragen.

Sei nun also k eine Stelle im Spektrum, die ein Maximum mit einer ausreichenden Breite b darstellt. Dann interessiert, uns welche Frequenz F, welche Amplitude A und welche Phase Φ der Sinusanteil des Ausgangssignals hatte. Man kann aus dem Wert k, an dem das Maximum auftritt, grob die Frequenz ermitteln, da jedem Wert k eine Frequenz zugeordnet ist. Da aber k immer eine natürliche Zahl ist (also keine Nachkommastellen hat), ist diese Schätzung sehr ungenau. Wir verwenden daher die von [Ser89] beschriebene Methode (mit einer etwas anderen mathematische Herleitung).

Zunächst ermitteln wir die dB-Werte der zum Maximum gehörenden höchsten drei Spektrumwerte. Eine Interpolation, die auf den dB-Werten (statt auf den linearen Werten) basiert, wäre bei einer Gauss-Fensterfunktion exakt, da diese auch im Spektrum zu einer Gaussfunktion wird. Bei unserer Fensterfunktion ist die dB-Interpolation zwar nicht mehr exakt, aber dennoch besser als eine Interpolation der linearen Werte.

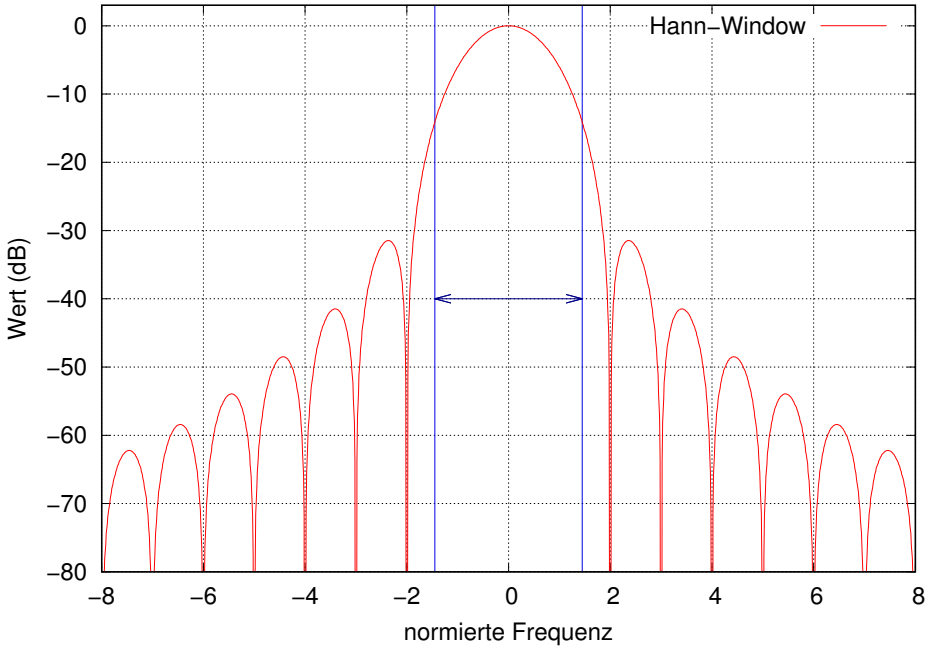


Abbildung 2.7: Minimale Peakbreite verglichen mit Hann-Fenster (Frequenzbereich)

$$\alpha = db(|X(k-1)|)$$
$$\beta = db(|X(k)|)$$
$$\gamma = db(|X(k+1)|)$$

Nun suchen wir Parameter der Parabelfunktion

$$y(x) = ax^2 + bx + c$$

sodass

$$y(-1) = \alpha$$
$$y(0) = \beta$$
$$y(1) = \gamma$$

Abbildung 2.8 stellt ein Beispiel für die Funktion $y(x)$ dar. Nach a, b und c aufgelöst ergeben sich die Parameter:

$$a = \frac{\alpha - 2\beta + \gamma}{2}$$
$$b = \frac{\gamma - \alpha}{2}$$
$$c = \beta$$

Das Maximum der Parabel m lässt sich nun aus diesen Parametern berechnen (durch Bildung und Nullsetzen der Ableitung $y'(x)$):

$$m = -\frac{b}{2a}$$

Es ergibt sich eine exakte Position des lokalen Maximums

$$k^* = k + m$$

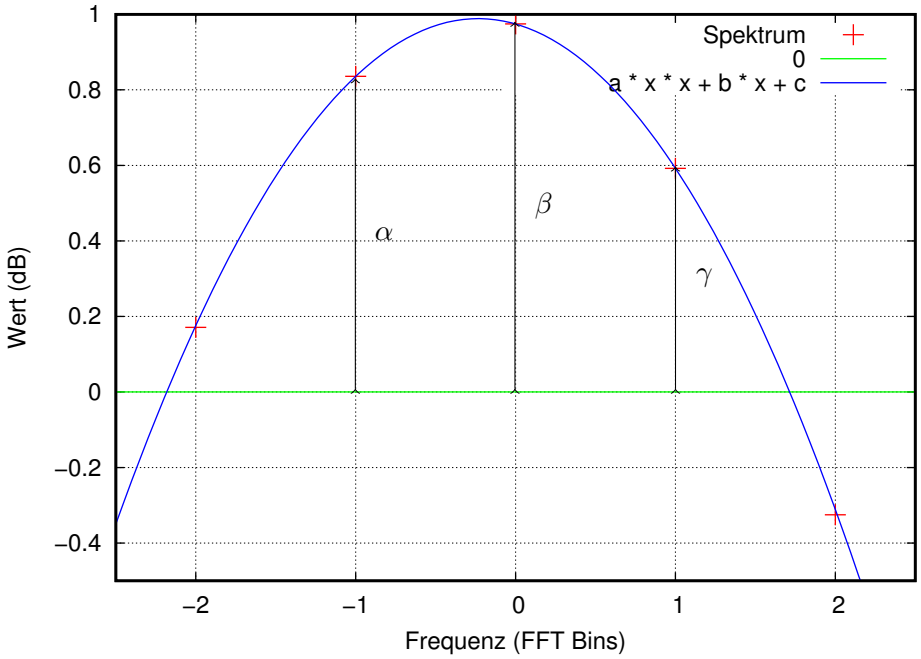


Abbildung 2.8: Parabelfunktion $y(x)$ zur Interpolation der Spektrumwerte

und damit die gesuchte Frequenz

$$F = \frac{k^*}{N/2} F_s$$

wobei N die Anzahl der FFT-Bins inklusive Zeropadding ist, und F_s die Samplingfrequenz. Die zugehörige Amplitude ergibt sich durch Einsetzen von m in $y(x)$:

$$A = d \cdot 10^{y(m)/20}$$

wobei d ein Normierungsfaktor ist, der von der verwendeten Fensterfunktion abhängt. Dabei ist $W(0)$ durch die Fourier-Transformation der Fensterfunktion $w(n)$ gegeben, die in diesem Fall aber einfach im Zeitbereich berechnet werden kann.

$$d = \frac{2}{W(0)}$$
$$W(0) = \sum_{n=0}^{N-1} w(n)$$

Um die Phase zu ermitteln, wird für den Realteil und den Imaginärteil des Spektrums jeweils eine Parabel (wie oben) bestimmt, sodass es zwei Parabelfunktionen $y_{re}(x)$ und $y_{im}(x)$ gibt, die durch die Real- und Imaginärteile der FFT Bins um das lokale Maximum herum gehen. Setzt man m (der Wert bei dem die Funktion $y(x)$ maximal wird) in diese Funktionen ein, erhält man interpolierte Werte von Realteil und Imaginärteil des Spektrums genau beim lokalen Maximum. Daraus lässt sich die Phase des Sinusanteils

$$\Phi = arg(y_{re}(m) + iy_{im}(m)) = atan2(y_{im}(m), y_{re}(m))$$

berechnen. Die C-Funktion $atan2$ berechnet das komplexe Argument und damit die Phase aus dem Real- und Imaginärteil.

Innerhalb der Software wird eine etwas andere Phase verwendet. Die oben angegebene Phase Φ gehört zu einer Kosinusschwingung. Die Phase, die wir verwenden, gehört zu zu einer Sinusschwingung, sodass eine Konstante $(\pi/2)$ zu Φ addiert werden muss. Weiterhin ist Φ die Phase in der Mitte des Analyseframes. Wir speichern jedoch als Φ^* die Phase zum Anfang des Frames.

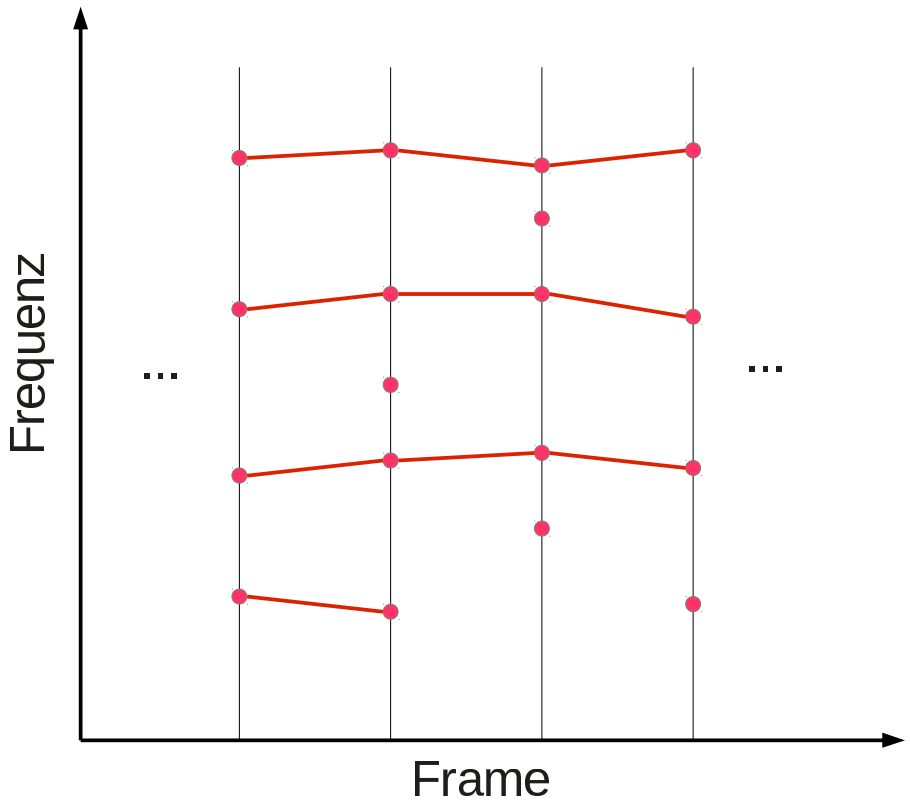


Abbildung 2.9: Verbindung der Maxima benachbarter Frames

Zusammengefasst verwenden wir daher für die Phase

$$\Phi^* = normalize\_phase\left(\Phi + \frac{\pi}{2} - \frac{2\pi F}{F_s} \cdot \frac{N-1}{2}\right)$$

wobei $normalize\_phase(x)$ eine Funktion ist, die die Phase in den Bereich $[0, 2\pi]$ bringt. N ist die Anzahl der Werte im Analyseframe. F ist die oben berechnete Frequenz. F_s ist die Samplingfrequenz.

2.3.4 Verbinden der lokalen Maxima benachbarter Frames

Nach der Ermittlung einzelner lokaler Maxima in den einzelnen Frames werden lokale Maxima jeweils benachbarter Frames verbunden, sofern dies möglich ist. Um dies für einen Frame mit der Nummer n und den Vorgänger des Frames mit der Nummer $n-1$ zu erreichen wird für jedes Maximum des Frames n das nächstliegende Maximum des Frames $n-1$ ermittelt. Liegen diese beiden um weniger als 5% auseinander, werden sie verbunden.

Wie in Abbildung 2.9 angedeutet nehmen wir an, dass der Klang typischer Instrumente, für die die Analyse funktionieren soll, aus Teiltönen (Sinusanteilen) besteht, die von Frame zu Frame sehr ähnliche Frequenzen haben. Für jeden Teilton erwarten wir also, dass er von jedem Frame zum jeweils nächsten Frame immer verbunden wird, sodass grafisch gesprochen für die stabilen Teiltöne als Resultat dieses Schrittes lange Linien entstehen.

Oftmals werden bei Aufnahmen allerdings an manchen Stellen Teiltöne nicht in jedem Frame in der Analyse zu finden sein, oder stark schwanken, sodass es durchaus zu für einen Teilton zu unterbrochenen Linien kommen kann. Andererseits können auch beispielsweise durch Noise entstandene Maxima benachbarter Frames verbunden werden, sodass eine Verbindung nicht immer mit einem Teilton gleichzusetzen ist.

2.3.5 Überprüfung der verbunden Maxima

In Kapitel 2.3.3 wurden bereits Kriterien verwendet, um lokal innerhalb des Frames k zu entscheiden, ob ein Maximum tatsächlich einem Sinusanteil entspricht, oder durch Nebenmaxima oder Noise entstanden ist. Dadurch dass die lokalen Maxima nun verbunden wurden kann ein weiteres Kriterium gegeben werden. Sei X^* das größte Maximum mit einer direkten oder indirekten Verbindung zu $X(k)$. Dann berechnen wir

$$h = db(|X^*|) - db(|X_{max}|)$$

und behalten nur die lokalen Maxima des Frames, für die h groß genug ist.

Im Rahmen dieser Arbeit fordern wir $h > -90$. Dies ist eine extrem konservativ gewählte Schranke und im Endeffekt schränkt sie mit unseren anderen Kriterien die Menge der übernommen Maxima nicht weiter ein. Auch die anderen Parameter zur Elimination von lokalen Maxima in den obigen Kapiteln sind eher konservativ gewählt.

Für die Analyse bedeutet dies, dass wir oftmals lokale Maxima behalten, die nicht hörbar zum Instrumentenklang beitragen, oder zufällig oder durch Nebenmaxima entstanden sind. Da der Schwerpunkt dieser Arbeit das Morphing ist, stören diese aber später nicht. Im Morphing werden sie zwar mitbehandelt, bleiben aber für den Gesamtklang unwesentlich. In der Synthese werden sie dann mitsynthetisiert, aber bleiben auch hier unhörbar.

Die Selektion der Maxima, die später gespeichert und vom Morphing verwendet werden, wird niemals perfekt sein. Da es aber nicht sehr schädlich ist, zu viele lokale Maxima als Ausgabe der Analyse zu erhalten, aber sehr wohl hörbare Artefakte erzeugen kann, wenn zu wenige lokale Maxima als Ausgabe zu erhalten, wählen wir die Kriterien nicht zu restriktiv. Dadurch erhalten wir eher zu viele Sinusanteile in jeder Framebeschreibung, was zwar Rechenzeit in der Synthese und im Morphing verursacht, aber andererseits keine so problematischen klanglichen Konsequenzen hat wie Kriterien, die zu viele Sinusanteile eliminieren.

2.3.6 Bestimmung des Spektrums des stochastischen Anteils

Der nächste Schritt nach der Bestimmung aller P Teiltöne eines Frames, beschrieben als Amplitude A_p, Frequenz F_p und Phase Φ_p ist die Berechnung des verbleibenden Spektrums, das nicht durch diese Teiltöne erzeugt wird. In Kapitel 2.1.4 hatten wir unser Signal als Summe des deterministischen Anteils $d(t)$ und der Geräuschanteils $e(t)$ beschrieben:

$$x(t) = d(t) + e(t) = \sum_{p=1}^{P} A_p cos\left(2\pi \frac{F_p}{F_s} + \Phi_p\right) + e(t)$$

Wir berechnen also zunächst den deterministischen Anteil (der durch die Addition der Teiltöne erzeugt wird):

$$d(t) = \sum_{p=1}^{P} A_p cos\left(2\pi \frac{F_p}{F_s} + \Phi_p\right)$$

und deren Fourier-Transformation $D(k)$, analog zu Kapitel 2.3.2, mit dem gleichen Fenster, der gleichen Verteilung der Eingabe auf die FFT Bins und dem gleichen Zeropadding. Somit können wir die beiden Spektren $X(k)$ und $D(k)$ verwenden, um den Geräuschanteil des Frames zu bestimmen, der nicht durch Sinusanteile beschreibbar ist. Dieses Spektrum nennen wir $E(k)$ und berechnen es wie folgt:

$$E(k) = \begin{cases} 0 & \text{für } |D(k)| \geq |X(k)| \\ \frac{|X(k)|-|D(k)|}{|X(k)|} X(k) & \text{sonst} \end{cases}$$

Wenn also das Spektrum des deterministischen Anteils betragsmäßig größer oder gleich ist als das Spektrum des analysierten Signals, so liegt zu dieser Frequenz gar kein Geräuschanteil vor. Wenn es betragsmäßig kleiner ist, erhält diese Rechnung die Phase von $X(k)$, reduziert aber den Betrag des Signals um den Betrag des deterministischen Anteils.

Die Analyse eines Frames eines Oboentons mit einer Grundfrequenz von 440 Hz ist in den folgenden Abbildungen zu sehen. Das Spektrum $X(k)$ ist in Abbildung 2.10 dargestellt. Das aus den P Parametern der Amplitude A_p, Frequenz F_p und Phase Φ_p rekonstruierte Signal ist d(t). Dessen Spektrum $D(k)$ findet sich in Abbildung 2.11. Schließlich kann das Spektrum des verbleibenden stochastischen Anteils ermittelt werden. Dieser wird gewonnen indem das Spektrum $D(k)$ der Sinusanteile aus dem Spektrum $X(k)$ entfernt wird. Abbildung 2.12 zeigt dieses Restspektrum $E(k)$.

2.3.7 Unterteilung des stochastischen Anteils in Frequenzbänder

Um den stochastischen Anteil auf wenige Parameter pro Frame zu reduzieren, unterteilen wir das im letzten Kapitel bestimmte Spektrum des stochastischen Anteils in 32 Bänder. Als Parameter wird dann lediglich die in jedem Band enthaltene Energie gespeichert. Bei der Synthese wird umgekehrt ein zufälliges Spektrum generiert, und lediglich die Energie aus jedem Band so synthetisiert, dass die bei der Analyse ermittelten Parameter eingehalten werden.

Für die Unterteilung in Bänder wird die Mel-Skala verwendet. Jedes Band hat eine konstante Breite von 125 mel. Durch diese Aufteilung wird abgebildet, dass das menschliche Ohr bei höheren Tönen Frequenzunterschiede weniger genau wahrnimmt. Anders gesagt unterteilen wir das Spektrum so, dass die Bänder am Anfang des Spektrums enger beieinander liegen, und weiter auseinander liegen, je höher die Bandgrenzen sind. Dies ergibt eine Approximation des menschlichen Hörens, welches die Frequenzbänder als etwa gleich breit wahrnimmt, da sie auf der Mel-Skala die gleiche Breite haben. Tabelle 2.2 stellt die Grenzen der Frequenzbänder am Anfang und am Ende des Spektrums dar. Auf Abbildung 2.13 sind die Breiten der 32 Bänder in Hertz dargestellt.

Band	**Start (mel)**	**Ende (mel)**	**Start (Hz)**	**Ende (Hz)**	**Breite (Hz)**
0	30	155	18,9	103,2	84,3
1	155	280	103,2	197,4	94,2
2	280	405	197,4	302,7	105,3
3	405	530	302,7	420,3	117,6
4	530	655	420,3	551,7	131,4
...	...	...	...	...	...
29	3655	3780	17 229,5	19 332,6	2103,1
30	3780	3905	19 332,6	21 682,4	2349,8
31	3905	4030	21 682,4	24 309,9	2625,4

Tabelle 2.2: Frequenzbänder zur Beschreibung des stochastischen Anteils

Zur Umrechnung von Mel und Hertz gilt folgende Formel:

$$freq(mel) = 700(e^{\frac{mel}{1127}} - 1)$$

Für jedes Frequenzband bestimmen wir zunächst die enthaltene Energie, indem wir die zugehörigen $E(k)^2$ der FFT Bins aufsummieren. Indem wir die Energie durch die Breite des Bandes $high_b - low_b$

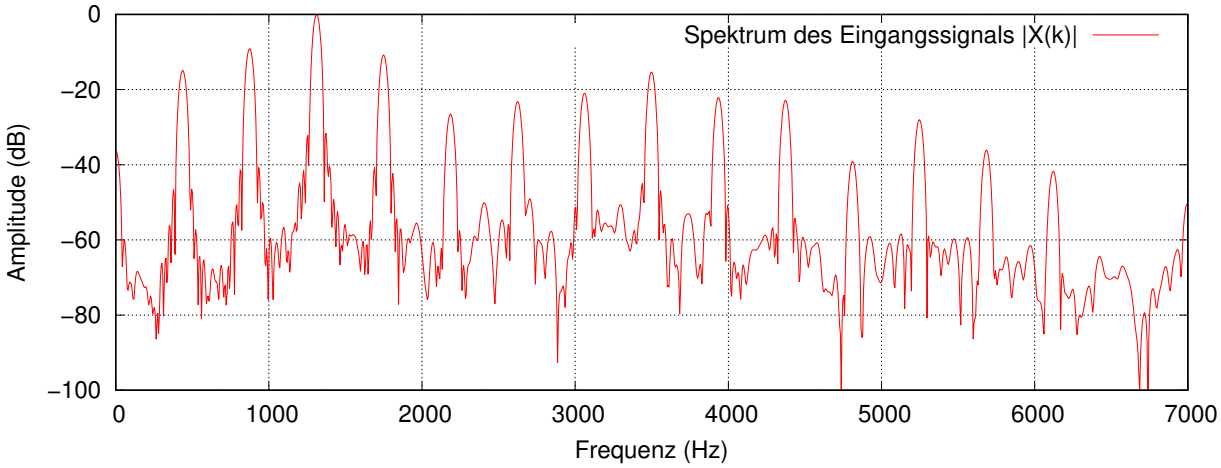


Abbildung 2.10: Spektrum von $x(t)$

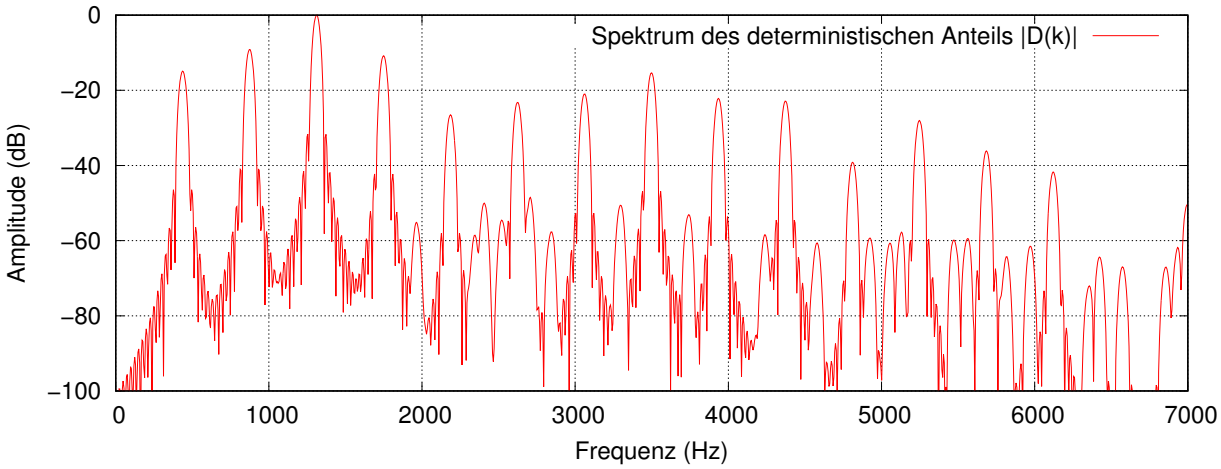


Abbildung 2.11: Spektrum von $d(t)$

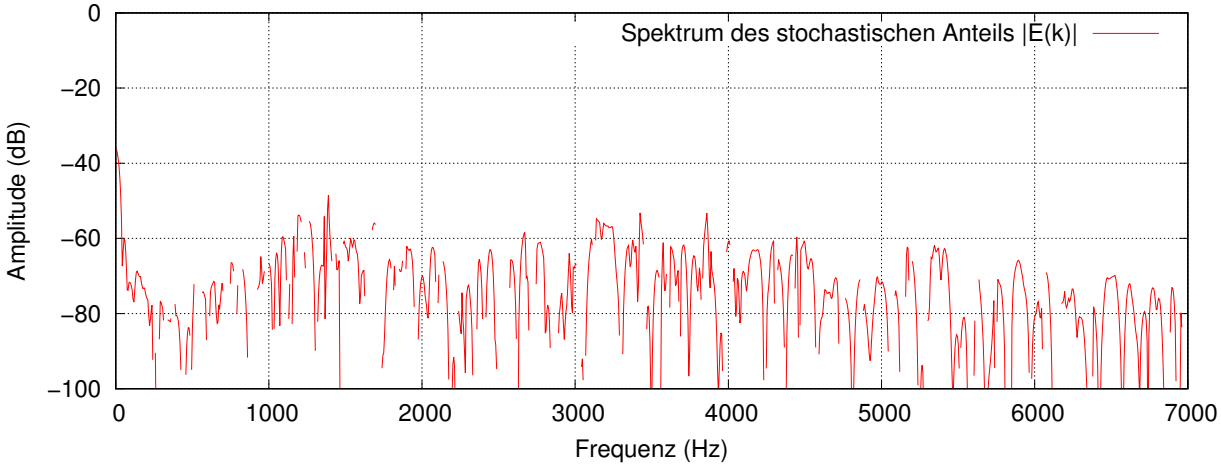


Abbildung 2.12: Spektrum des stochastischen Anteils $|E(k)|$

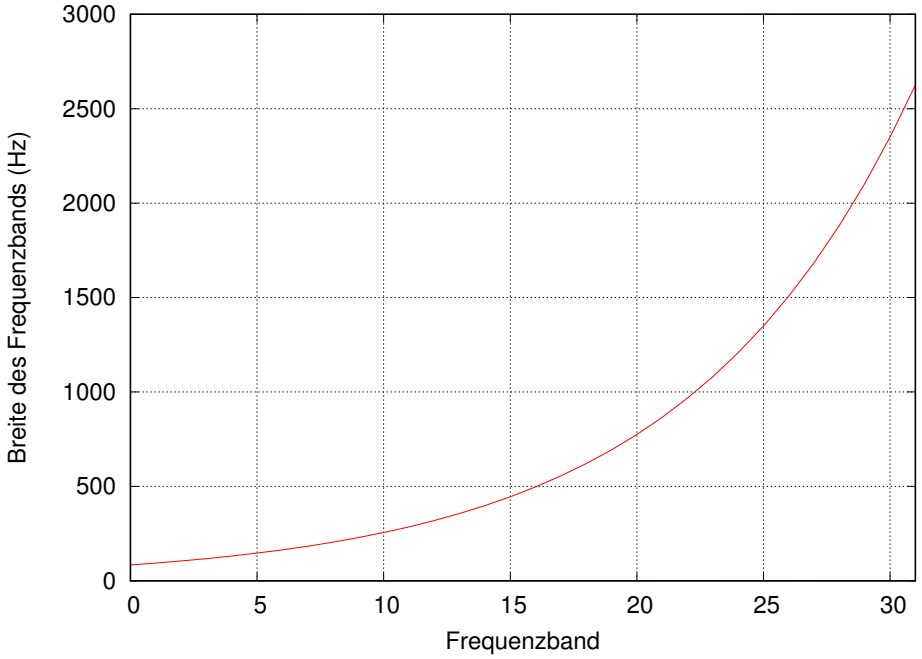


Abbildung 2.13: Die unterschiedliche Breite der Frequenzbänder

teilen und die Wurzel berechnen erhalten wir den durchschnittlichen Betrag $|E(k)|$. In der Synthese ist es dieser Betrag, den alle im Band liegenden Werte haben müssen, damit die Energie der Energie des Bandes der in der Analyse beobachteten Energie entspricht. Pro Band berechnen wir also $NOISE_b$:

$$NOISE_b = \sqrt{\sum_{low_b \leq k < high_b} \frac{E(k)^2}{(high_b - low_b) norm}}$$

$$norm = \frac{1}{2} \cdot mix\_freq \cdot \sum_{n=0}^{N-1} w(n)^2$$

Dabei sind low_b und $high_b$ die Bandgrenzen in FFT Bins. Der zusätzliche Normierungsfaktor $norm$ wird nun Schritt für Schritt erklärt.

Grundsätzlich gilt, dass die Energie des Eingangssignals der FFT der Energie des Spektrums der FFT entspricht. Da wir aber die reelle Variante verwenden, betrachten wir nur die eine Hälfte des Spektrums, die andere Hälfte ergibt sich über die in Kapitel 2.2.2 erwähnte Symmetrie. Indem wir den Faktor $1/2$ in $norm$ verwenden, wird die in die Summe eingehende Energie verdoppelt, da wir durch $norm$ dividieren, und somit der zweite symmetrische FFT Anteil bei der Bestimmung von $NOISE_b$ mitberücksichtigt.

Die Abtastrate mix_freq wird bei der Normierung einbezogen, da bei der Synthese möglicherweise eine andere Abtastrate als bei der Analyse verwendet wird.

Um den Einfluss der Multiplikation des stochastischen Eingabesignals $e(n)$ mit der Fensterfunktion $w(n)$ zu sehen, betrachten wir den Beitrag des n-ten Samples zur Gesamtenergie. Ohne Fensterfunktion wäre dieser $e(n)^2$, mit Fensterfunktion wäre dieser $w(n)^2 \cdot e(n)^2$. Im Mittel wird die Energie jedes Eingangswerts mit dem Faktor

$$wnorm^* = \frac{1}{N} \sum_{n=0}^{N-1} w(n)^2$$

multipliziert. Die Länge des Analysefensters N muss auch in der Normierung berücksichtigt werden, um die Ergebnisse mit unterschiedlich langen Analyseframes vergleichbar zu machen. Insgesamt

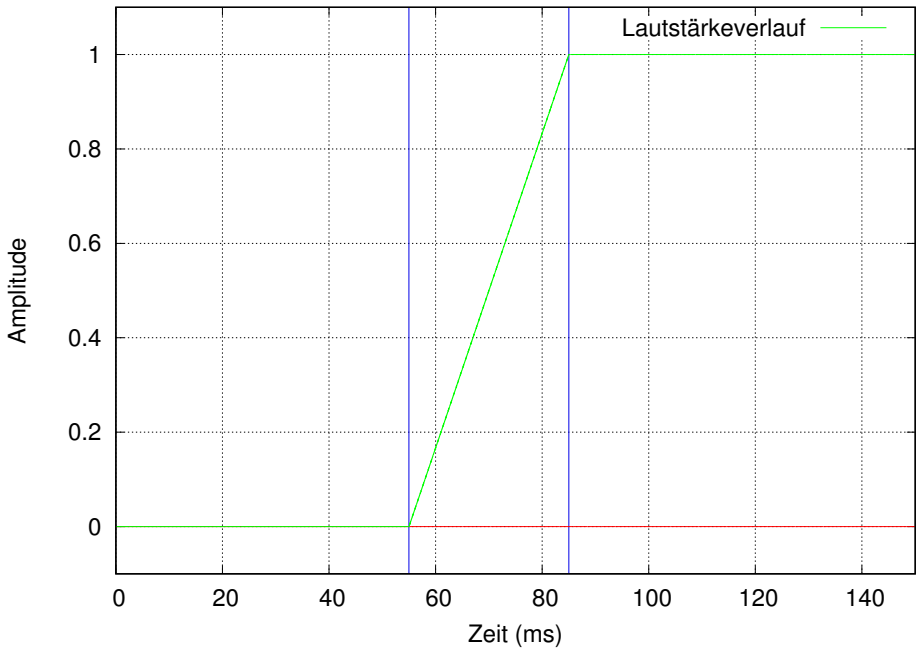


Abbildung 2.14: Durch Attack-Parameter festgelegter Lautstärkeverlauf

ergibt sich für die Normierung *norm* der fensterfunktionsbedingte Anteil

$$wnorm = N \cdot wnorm^* = \sum_{n=0}^{N-1} w(n)^2$$

2.4 Bestimmung der Attack-Parameter

Durch die Analyse des Signals wird eine Unterteilung des Signals in Frames vorgenommen, und bei der Synthese werden diese Frames wieder synthetisiert, und dann addiert. Schnelle Lautstärkeänderungen werden dadurch über die gesamte Framelänge verteilt. Beim Test des Verfahrens mit verschiedenen Instrumenten wurde dadurch besonders der harte Anschlag von Klavieraufnahmen so verschmiert, dass der Unterschied zwischen Synthese und Original sehr gut zu hören war. Im Folgenden wird eine Erweiterung der Analyse/Synthese vorgestellt, die schnelle Lautstärkeänderungen am Anfang des Tons (Attack) besser beschreibt, und damit den Klang von Klavieraufnahmen deutlich verbessert.

2.4.1 Die Suche nach den optimalen Attack-Parametern

Es gibt zwei Parameter, die für den Anfang des Tons ermittelt werden müssen. Der Zeitpunkt des Anfangs (davor sind alle Samples Null), der im Folgenden mit `attack_start_ms` bezeichnet wird, und der Zeitpunkt wo der Ton vollständig vorhanden ist, den wir `attack_end_ms` genannt haben. Der daraus resultierende Lautstärkeverlauf ist in Abbildung 2.14 zu sehen. Die beiden vertikalen blauen Linien entsprechen den beiden Attack-Parametern.

Das von uns implementierte Verfahren benutzt eine Fehlerfunktion `attack_error()`, die für jede Belegung von `attack_start_ms` und `attack_end_ms` einen Fehler angibt, der durch diese Belegung verursacht wird. Am Anfang werden die Parameter initialisiert. Dann wird durch Verändern der Parameter versucht, den von der Fehlerfunktion angegebenen Fehler zu reduzieren. Wird der Fehler geringer, wird die neue (veränderte) Belegung der Parametern verwendet. Wird der Fehler größer,

Listing 2.1: Algorithmus zur Bestimmung der Attack-Parameter

```
attack_start_ms = 0
attack_end_ms = 10

no_mod = 0

while (no_mod < 3000)
{
  R = attack_delta (no_mod)

  attack_start_ms* = attack_start_ms + random (-R, R)
  attack_end_ms* = attack_end_ms + random (-R,R)

  error = attack_error (attack_start_ms, attack_end_ms)
  error* = attack_error (attack_start_ms*, attack_end_ms*)

  if (error* < error) {
    attack_start_ms = attack_start_ms*
    attack_end_ms = attack_end_ms*
    no_mod = 0
  }
  else {
    no_mod += 1
  }
}
```

wird die alte Belegung verwendet. Der Algorithmus als Pseudo-Code findet sich in Listing 2.1.

Die `no_mod` Variable gibt die Anzahl erfolgloser Modifikationen der Attack-Parameter an. Wenn sie den Wert 3000 erreicht, gehen wir davon aus, dass die Parameter nicht weiter verbessert werden können. Zusätzlich zu dieser Funktion übernimmt die `no_mod` Variable auch die Steuerung der Länge der Veränderung der Parameter `R`. Ist `no_mod` niedrig, gehen wir davon aus, dass es einen relativ großen Schritt geben könnte, der die Parameter deutlich verändert und einen kleineren Fehler verursacht. Je länger wir erfolglos waren, desto kleiner wählen wir die Schritte, da in dieser Situation die zuvor versuchten größeren Änderungen keine Verbesserung erzielt haben. Die Tabelle 2.3 gibt die Funktion `attack_delta` an.

no_mod	**attack_delta (no_mod)**
no_mod < 500	100
no_mod < 1000	20
no_mod < 1500	1
no_mod < 2000	0,2
no_mod < 2500	0,01

Tabelle 2.3: Die Funktion attack_delta()

Die Funktion `random(a,b)` gibt eine Zufallszahl zwischen a und b zurück.

Insgesamt funktioniert diese iterative Suche nach Attack-Parametern - zum Beispiel auf den Klaviersamples - recht gut. Dies ist wohl auch damit zu begründen, dass der Suchraum lediglich zweidimensional ist, und dass die Fehlerfunktion oft bei einer kleinen Wertänderung der Eingabewerte auch eine kleine Änderung des Fehlers ergibt. Die verwendete Fehlerfunktion wird im folgenden Kapitel beschrieben.

2.4.2 Die Fehlerfunktion für die Attack-Parameter

Die Grundidee der Fehlerfunktion ist, einen durch die beiden Attack-Parameter gegebenen Lautstärkeverlauf bei der Synthese zu verwenden, und das Resultat mit dem Eingangssignal zu vergleichen. Idealerweise wären die beiden Signale gleich, und der Fehler wäre Null. In der Praxis sollte der Fehler umso kleiner sein, je näher die Attack-Parameter dem Verlauf des Attack-Envelope des Originals kommen.

Das synthetisierte Signal lässt sich als Summe der deterministischen Anteile der einzelnen Frames berechnen. Diese werden mit dem Overlap-Add-Verfahren (Überlappen und Addieren) mit einem Synthesewindow $w^*(t)$ zum rekonstruierten Signal zusammengesetzt. Als Synthesewindow kommt eine Hanning-Fensterfunktion zum Einsatz. Diese wird so normiert, dass sich die überlappenden Fenster später zu 1 summieren (dies ist möglich weil unsere Framelänge immer genau vier mal so groß ist wie die Hopsize H: $M = 4H$).

Wir verwenden den Index l um die Amplituden, Frequenzen und Phasen (A_p^l, F_p^l, Φ_p^l) des Frames l zu bezeichnen. Aus diesen kann das zugehörigen Framesignal $s^l(t)$ zusammengesetzt werden:

$$\begin{aligned} s^l(t) &= d^l(t)w^*(t) \\ &= \sum_{p=1}^{P^l} A_p^l cos\left(2\pi\frac{F_p^l}{F_s}t + \Phi_p^l\right) w^*(t) \end{aligned}$$

Dabei ist t der Zeitindex des Frames, und geht von $t = 0$ bis $M - 1$, M ist die STFT Framelänge. Um den nächsten Schritt zu vereinfachen, legen wir fest, dass das Synthesewindow $w^*(t)$ für jeden Zeitindex außerhalb dieses Bereichs Null ist.

Die Einzelframes lassen sich nun zum Gesamtsignal zusammensetzen, wobei sie verschoben aufaddiert werden (H ist die „Hopsize“):

$$s(t) = \sum_{l=0}^{L} s^l(t - lH)$$

Als Fehlerfunkion ergibt sich daher:

$$attack\_error(\alpha, \beta) = \sum_{t=0}^{T} (x(t) - env(t, \alpha, \beta)s(t))^2$$

Die obere Grenze der zu betrachtenden Samples T lässt sich auf einen Teil der Länge des Eingabesignals einschränken, sofern das Ende des Attack-Envelopes β einen Maximalwert niemals übersteigt. Bei unserer Implementation gilt immer, dass $\beta < 200$ (ms) ist, sodass sich dadurch T einschränken lässt.

Die Envelopefunktion ist wie folgt definiert:

$$env(t, \alpha, \beta) = \begin{cases} 0 & \text{für} \quad ms(t) < \alpha \\ \frac{ms(t)-\alpha}{\beta-\alpha} & \text{für } \alpha \leq \quad ms(t) < \beta \\ 1 & \text{für } \beta \leq \quad ms(t) \end{cases}$$

$$ms(t) = 1000\frac{t}{F_s}$$

2.4.3 Weitere Details zur Attack-Parameter Bestimmung

Im vorangegangenen Kapitel wurde schon angedeutet, dass die Attack-Parameter auf bestimmte Bereiche beschränkt werden. Der zufälligen Veränderung im Algorithmus in Kapitel 2.4.1 werden also Grenzen gesetzt. Es wird immer sichergestellt, dass die Länge des Attacks der Envelopefunktion mindestens 5 ms umfasst, also $attack\_end\_ms - attack\_start\_ms > 5$ ist. Diese Schranke verhindert, dass das synthetisierte Signal später so schnell eingeblendet wird, dass ein Klickgeräusch entsteht. Zusätzlich wird $attack\_end\_ms$ so beschränkt, dass es maximal 200 ms ist.

Wir nehmen an, dass der durch den Attack-Envelope gegebene Verlauf dem Attack-Envelope des Eingabesignals entspricht. Daher wurden die Amplituden-Parameter, die aus dem STFT Spektrum der Frames abgeleitet wurden systematisch unterschätzt, wenn Samples vor dem $attack\_start\_ms$ mit eingeflossen sind. Eine grobe Korrektur dafür ist, dass jede Amplitude jedes Frames mit einem Faktor $scale^l$ multipliziert wird. M ist wie immer die Größe des Frames. Mit S^l bezeichnen wir die Anzahl der Samples, für die der Syntheseenvelope nicht Null ist.

$$scale^l = \begin{cases} \frac{M}{S^l} & \text{für } S^l > \frac{M}{8} \\ 0 & \text{sonst} \end{cases}$$

Mittels dieser Formel kann man neue Schätzungen für die Amplitude jedes Frames erhalten:

$$A_k^{*l} = scale^l A_k^l$$

Die Fallunterscheidung bei der Berechnung von $scale^l$ bewirkt, dass Frames, die bei der Analyse fast nur aus Nullen bestanden haben müssen, mit 0 skaliert werden, also nicht zum synthetisierten Signal $s(t)$ beitragen. In diesem Fall nehmen wir an, dass die kurze Länge vorliegenden Daten vermutlich zu schlechten Analyseergebnissen geführt haben, sodass es besser ist die Ergebnisse der Analyse zu ignorieren.

Ein letztes den Attack betreffendes Detail ist, dass das Eingangssignal $x(t)$ bei der Analyse vorne mit $M/2$ Nullen ergänzt wird. Diese Nullen dienen dazu, dass Analyseframes generiert werden, die „vor“ dem eigentlichen Anfang des Signals liegen, und die über Skalierung mittels $scale^l$ Faktoren brauchbare Ausgangsdaten zur Beschreibung des Signals während der Attack Phase liefern. Bei der Synthese werden die vorangestellten $M/2$ Nullen wieder abgeschnitten, sodass das Signal nicht länger wird. Der $attack\_start\_ms$ Parameter wird so eingeschränkt, dass der Attack niemals vor den $M/2$ Nullen anfängt.

2.4.4 Test der Attack-Envelopes bei Klavieraufnahmen

Eine starke Verbesserung ergibt sich durch die Verwendung eines Attack-Envelopes vor allem bei Instrumenten, die eine kurze Einschwingphase haben. Beim Klavier wird durch die Bestimmung der Attack-Parameter eine deutlich bessere Ähnlichkeit zwischen Originalaufnahme und Resynthese erzielt. Die hörbare Verbesserung kann auch bei einer grafischen Darstellung der ersten 50 Millisekunden von Original (Abbildung 2.15), Resynthese ohne Attack-Envelope (Abbildung 2.16) und Resynthese mit Attack-Envelope (Abbildung 2.17) gesehen werden.

Die Einschwingphase bei der Resynthese mit Attack-Envelope ist zwar nicht so kurz wie im Original, jedoch gegenüber der Resynthese/Analyse ohne Attack-Envelope wird eine deutliche Verbesserung erreicht.

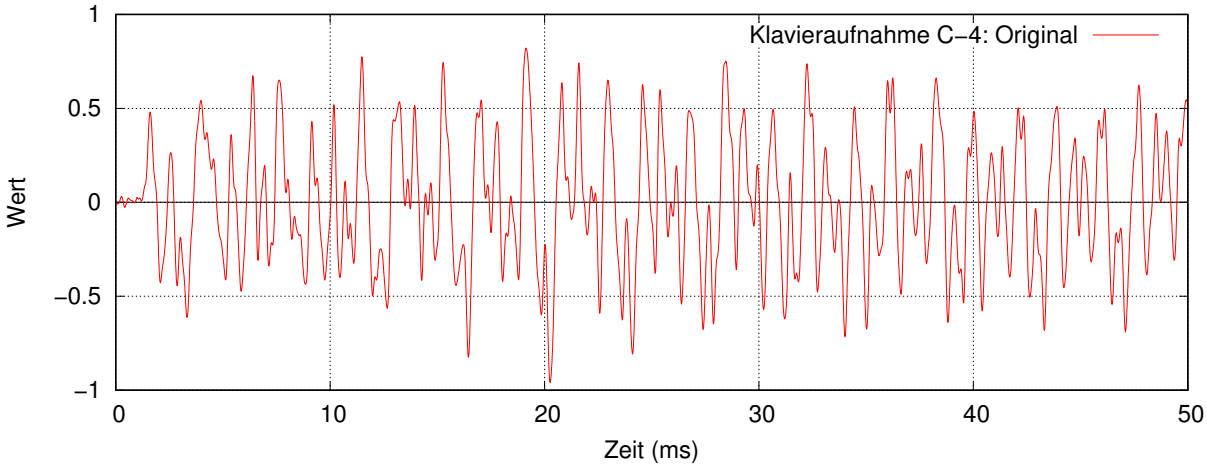


Abbildung 2.15: Originalaufnahme Klavier

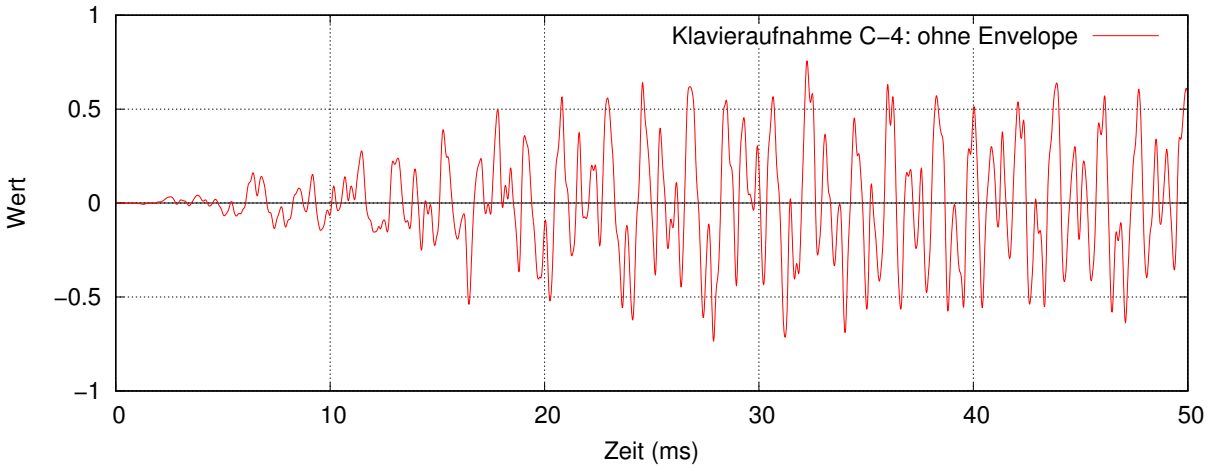


Abbildung 2.16: Analyse/Wiedergabe ohne Attack-Envelope

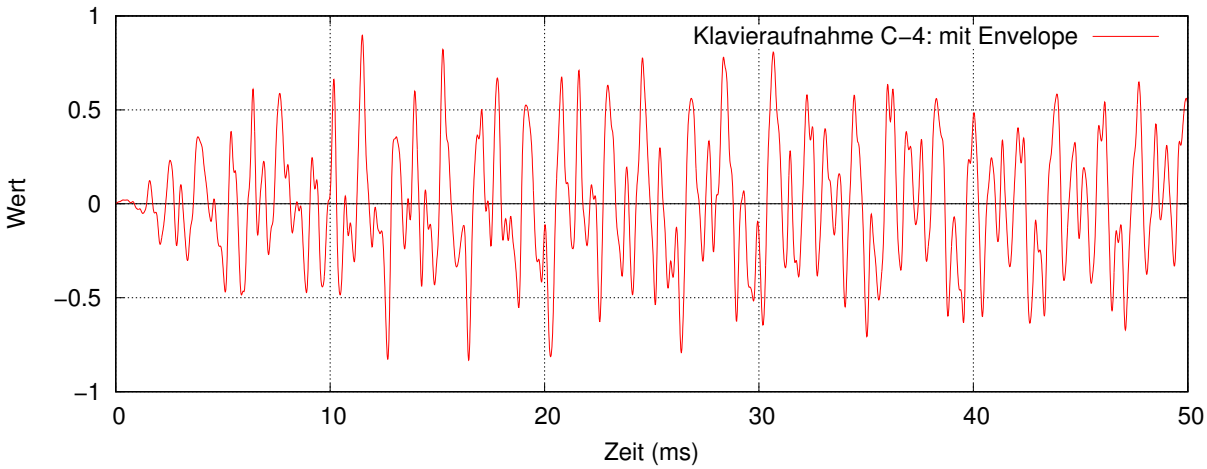


Abbildung 2.17: Analyse/Wiedergabe mit Attack Envelope

2.5 Speicherplatzsparende Repräsentation

Wir nehmen an, dass in einer späteren Anwendung viele verschiedene Instrumente verfügbar sind, zwischen denen Morphing möglich ist. Für jedes dieser Instrumente werden Aufnahmen in vielen verschiedenen Tonhöhen und möglicherweise nochmals in verschiedenen Lautstärkeebenen vorliegen. In diesem Kapitel wird beschrieben, wie die Modelldaten so gespeichert werden können, dass die einzelnen Werte für Frequenzen, Amplituden, Phasen und Noise jeweils mit 16 Bit dargestellt werden können.

Diese Repräsentation ist nicht nur zum Übertragen und Speichern auf der Festplatte nützlich; auch für das laufende Programm kann so eine RAM-Reduktion auf 50% der Größe erreicht werden, die bei der Speicherung dieser Werte als 32 Bit Floats benötigt würde. Die Umrechnung von der kompakten Darstellung in die Floating Point Darstellung ist sehr effizient möglich.

2.5.1 16 Bit Repräsentation der Frequenzen

Betrachten wir zunächst die Repräsentation von Frequenzen. Wird eine Frequenz nach der Analyse in einem 16 Bit Wert gespeichert und dieser während der Synthese oder dem Morphing wieder in einen 32 Bit Floating Point Wert umgerechnet, entsteht notwendigerweise ein Fehler. Dieser muss so klein sein, dass man den Unterschied nicht wahrnehmen kann. Hierbei ist zu erwähnen, dass das menschliche Ohr Frequenzen logarithmisch wahrnimmt, also beispielsweise der Unterschied zwischen 440 Hz und 466 Hz als gleich groß empfunden wird wie der Unterschied zwischen 880 Hz und 932 Hz: jeweils handelt es sich um einen Halbtonschritt. Unser Ziel ist daher nicht, den durch die Speicherung als 16 Bit Wert entstehenden absoluten Fehler in Hertz zu minimieren, sondern den relativen Fehler.

Die Einheit Cent ist ein Maß, dass den Frequenzunterschied eines Halbtons weiter aufteilt. Eine Oktave hat 12 Halbtöne, und jeder dieser Halbtöne hat die Größe von 100 Cent. Als Faktor ausgedrückt ergibt sich also ein Faktor von

$$p_{cent} = \sqrt[1200]{2} = 2^{\frac{1}{1200}} \approx 1{,}0005777895$$

Auf unsere beiden Frequenzen bezogen wäre dies der Unterschied zwischen 440 Hz und 440,25 Hz oder zwischen 880 Hz und 880,51 Hz. Das menschliche Ohr kann diese Töne im direkten Vergleich bereits nicht mehr unterscheiden. Durch Schwebungseffekte, die entstehen wenn mehrere Töne gleichzeitig gespielt werden wäre es denkbar, dass ein Unterschied von einem Cent noch wahrnehmbar ist. Im folgenden wird daher eine Frequenzauflösung angestrebt, die genauer als ein Cent ist, und von der anzunehmen ist, dass sie in keinem Fall mehr hörbare Fehler produziert.

Die beiden benötigten Formeln sind:

$$\begin{aligned} F_{16} &= round(6000(\ln(\Delta F) + 3)) \\ \Delta F^* &= e^{F_{16}/6000-3} \end{aligned}$$

Hierbei wird die zu speichernde Frequenz nicht als absoluter Wert angegeben, sondern zuvor durch die Grundfrequenz des Tons geteilt. Wenn die Teiltöne eines Frames eines 440 Hz Trompetentons also etwa $\{440\text{ Hz}, 880\text{ Hz}, 1320\text{ Hz}, 1760\text{ Hz}, ...\}$ sind, wird ΔF also $\{1, 2, 3, 4, ...\}$ und F_{16} damit $\{18000, 22159, 24592, 26318, ...\}$.

Wir bestimmen nun den Fehler, der durch die Rundung von F_{16} auf den nächsten 16 Bit Integer entsteht, indem wir nachvollziehen, was mit der Frequenz ΔF durch die Quantisierung passiert;

R steht für den Rundungsfehler, der im Bereich $[-0{,}5; 0{,}5]$ liegt.

$$\begin{aligned}\Delta F^* &= e^{F_{16}/6000-3} \\ &= e^{(R+6000(\ln(\Delta F)+3))/6000-3} \\ &= e^{R/6000} e^{\ln(\Delta F)} \\ &= e^{R/6000} \Delta F \\ \frac{\Delta F^*}{\Delta F} &= e^{R/6000}\end{aligned}$$

Man sieht, dass die Frequenz ΔF durch die Quantisierung um den Faktor $e^{R/6000}$ verändert wird, sodass wir die Grenzen durch Einsetzen der ungünstigsten Werte von $R = -0{,}5$ und $R = 0{,}5$ erhalten. Damit gilt:

$$e^{-1/12000} \leq \frac{\Delta F^*}{\Delta F} \leq e^{1/12000}$$

Konvertiert man diese Werte noch in Cent,

$$\log_2\left(\frac{\Delta F^*}{\Delta F}\right) 1200 \text{ Cent} = \frac{\ln(e^{R/6000})}{\ln 2} 1200 \text{ Cent} = \frac{R/6000}{\ln 2} 1200 \text{ Cent} = \frac{R}{5 \ln 2} \text{ Cent}$$

ergibt sich schließlich durch Einsetzen von $R = -0{,}5$ und $0{,}5$ für den durch die Quantisierung eingeführten Fehler ein Intervall von $[-0{,}144$ Cent$; +0{,}144$ Cent$]$.

Eine weitere Frage außer der Verfälschung der Frequenz durch die Rundung ist die Frage nach dem Bereich der Frequenzen, der abgedeckt werden kann. Ein 16 Bit Integer deckt den Bereich $[0; 65535]$ ab. Dies entspricht einem Bereich von möglichen Frequenzen von

$$\begin{aligned}e^{0/6000-3} \leq \Delta F^* &\leq e^{65535/6000-3} \\ 0{,}0498 \leq \Delta F^* &\leq 2758{,}6591\end{aligned}$$

Normalerweise treten Frequenzen unterhalb der Grundfrequenz eines Tons überhaupt nicht auf, oder nur bei der Hälfte der Grundfrequenz. Es gibt eventuell einige besondere Fälle, aber wenn man von einer Grundfrequenz von 440 Hz ausgeht, wäre die darstellbare tiefste Frequenz etwa 22 Hz. Dies ist also auf jeden Fall ausreichend.

Für die obere Grenze macht es Sinn, einen besonders tiefen Ton zu betrachten, da bei diesem die meisten Obertöne auftreten können. Nehmen wir die Hörgrenze von etwa 16 Hz, wäre der größte mögliche Oberton etwa 44139 Hz; dies reicht also auch problemlos aus.

2.5.2 16 Bit Repräsentation der Amplituden

Für die Amplituden der Teiltöne (und weiter unten der Werte des Noisespektrums) ist der Ausgangspunkt die Darstellung der Lautstärken als dB Werte. Wieder ist die Lautstärkewahrnehmung des Menschen entscheidend, und diese entspricht etwa der logarithmischen dB Skala. Um auf 16 Bit Werte zu kommen, unterteilen wir jeden dB Schritt in 64 einzelne Werte, sodass wir bei jeder Amplitude maximal einen Lautstärkefehler von $\frac{1}{128}$ dB einführen. Zusätzlich verschieben wir den 0 dB Punkt in die Mitte des Wertebereichs, und können so alle Werte von -512 dB bis 512 dB darstellen.

$$\begin{aligned}db(x) &= 20 \log_{10}(x) \\ db2factor(x) &= 10^{x/20} \\ A_{16} &= round(64(db(A) + 512)) \\ A^* &= db2factor\left(\frac{A_{16}}{64} - 512\right)\end{aligned}$$

Wenn die Gesamtlautstärke bei 0 dB liegt, können bei normalen 16 Bit Signalen nur Signale bis -96 dB vorkommen, und diese wären schon praktisch unhörbar. Der Wertebereich der möglichen Amplituden wurde sehr viel größer gewählt, um auch Algorithmen, die solche Amplitudendaten verarbeiten, einen größeren Bereich zur Verfügung zu stellen. Beispielsweise könnte ein Analyseframe einen Teilton mit der Amplitude -60 dB enthalten, und eine nachgeschaltete Lautstärkeskalierung könnte diesen Teilton um -80 dB absenken. Der resultierende Wert von -140 dB ist zwar absolut unhörbar, aber wenigstens ist das Ergebnis der Lautstärkeskalierung noch als 16 Bit Amplitudenwert darstellbar.

Die 32 Parameter des stochastischen Anteils des Signals $NOISE_0$ bis $NOISE_{31}$ aus Kapitel 2.3.7 werden für die Speicherung als 16 Bit Werte wie die Amplitudenwerte behandelt.

2.5.3 16 Bit Repräsentation der Phasen

Die Speicherung der Phasen ist sehr einfach. Die möglichen Werte für die Phase sind immer aus dem Intervall $[0; 2\pi]$, und dieses wird auf den gesamten Wertebereich skaliert.

$$\Phi_{16} = round\left(65536\frac{\Phi}{2\pi}\right)$$

$$\Phi^* = 2\pi\frac{\Phi_{16}}{65536}$$

2.5.4 Gemessene Größen der kompakten Repräsentation

Der Speicherplatz, der effektiv benötigt wird, hängt vom analysierten Signal ab und kann nicht direkt berechnet werden. Die Anzahl der zu speichernden Frames hängt von der Länge des Signals und der Framelänge ab. Die Größe jedes Frames hängt wiederum von der Anzahl der gefundenen Teiltöne ab. Um ein zwischen unterschiedlichen Aufnahmen vergleichbares Maß zu erhalten, berechnen wir die Gesamtgröße der Analysedaten in 16 Bit Form und teilen diese durch die Länge des Signals in Sekunden. Daraus ergibt sich die Datenrate, die wir dann in Kilobyte pro Sekunde angeben können, wie in Tabelle 2.4. Zum Vergleich: ein typisches Eingangssignal wäre ein Mono-Signal mit 48 kHz, 16 Bit. Dieses benötigt unkomprimiert genau 96 Kb/sec.

Instrument	**Tonhöhe (Hz)**	**Datenrate (Kb/sec)**
Trompete	196,00	76,93
	440,00	70,43
	698,46	60,91
Oboe	293,66	72,68
	440,00	71,83
	1479,98	68,20
Violine	440,00	73,89
	1318,51	84,04

Tabelle 2.4: Speicherplatzbedarf der Analysedaten verschiedener Aufnahmen

Man sieht, dass die Datenrate etwas kleiner ist, als der Platzbedarf des Eingangssamples.

2.5.5 Weiteres Einsparungspotenzial

In der Software, die zu dieser Arbeit gehört, sind nur die obengenannten Optimierungen, also nur die Konvertierung der 32 Bit-Werte in 16 Bit-Werte implementiert.

Eine weitere Einsparung wäre möglich, wenn die isolierte Speicherung der einzelnen Frames aufgegeben wird. Bisher werden alle Parameter für jeden Frame unabhängig von den anderen Frames gespeichert. Dies ist später bei der Synthese und beim Morphing praktisch.

Teiltöne entwickeln sich normalerweise von Frame zu Frame, sodass man für jeden Teilton Funktionen speichern könnte. Es ergibt sich eine Funktion für die Entwicklung der Amplitude über die Zeit und eine Funktion für die Entwicklung der Frequenz über die Zeit. Die Phaseninformation wird bei der Synthese ergänzt und daher bei normalem Morphing nicht verwendet, braucht also für sehr viele Anwendungsfälle auch nicht gespeichert zu werden.

Von Frame zu Frame ist die Amplitude und Frequenz eines Teiltons ähnlich, sodass eine Speicherung der Amplituden- und Frequenzfunktion durch eine „Line-Segment Approximation“ möglich wäre [Str80]. Statt alle einzelnen Analysewerte zu speichern, wird hierbei die Funktion durch einen Verlauf von Linienstücken dargestellt, was oftmals gleich klingen sollte, aber wesentlich weniger Speicherplatz benötigt. Experimente mit Hörern über die Auswirkung der Vereinfachung von Frequenz- und Amplitudenfunktionen durch die „Line-Segment Approximation“ finden sich in [Gre75].

2.6 Manuell ergänzte Daten

2.6.1 Grundfrequenz

Für jedes Instrument liegen normalerweise die Aufnahmen verschiedener Tonhöhen vor. Beim Abspielen kann dann die Aufnahme verwendet werden, die am nächsten an der gewünschten Höhe liegt. Da normalerweise bekannt ist, welche Aufnahme welcher Midi-Note entspricht, wird dieser Parameter bei der Analyse angegeben, und die zugehörige Grundfrequenz G bei den Analysedaten gespeichert.

2.6.2 Loopparameter

Jede Aufnahme einer bestimmten Note eines Instruments ist notwendigerweise zeitlich begrenzt. Wenn beim Abspielen eine Note länger klingen soll, als die ursprüngliche Aufnahme, müssen Teile der Originaldaten mehrmals abgespielt werden („Loop“). Folgende Loopparameter werden pro Aufnahme manuell definiert:

Parameter	**Bedeutung**
`loop-type`	Art des Loops
`loop-start`	Startpunkt des Loops
`loop-end`	Endpunkt des Loops

Tabelle 2.5: Loopparameter

Wird `loop-type` auf „Vorwärtsloop“ gesetzt, wird beim Abspielen die Aufnahme von vorne bis zu `loop-end` gespielt, und dann wieder bei `loop-start` angefangen, und wieder bis `loop-end` gespielt, und so weiter.

Wird `loop-type` auf „Ping-Pong-Loop“ gesetzt, wird beim Abspielen die Aufnahme von vorne bis zu `loop-end` gespielt, dann rückwärts bis zu `loop-start`, dann wieder vorwärts bis zu `loop-end`, und so weiter.

Schließlich kann `loop-type` auch auf „kein Loop“ gesetzt werden, um das Abspielen auf die ein-

malige Wiedergabe von vorne nach hinten zu begrenzen.

Von den beiden Looptypen „Vorwärtsloop“ und „Ping-Pong-Loop“ gibt es jeweils zwei Varianten, die eine erlaubt das Angeben der Positionen `loop-start` und `loop-end` als Zeitindex in Samples, die andere als Frameindex in Frames. In jedem Fall operiert die Wiedergabe des Loops immer auf den Frames, sodass es nicht zu Klicks beim Abspielen kommen kann. Die Positionsangabe in Frames ist der Normalfall, das Festlegen der Looppositionen als Zeitindex wurde implementiert um Daten importieren zu können, für die bereits solche Looppositionen vorhanden sind.

2.7 Weitere Bearbeitung der Analysedaten

In den nachfolgenden Abschnitten werden einige Algorithmen beschrieben, die die Analysedaten weiter verändern. Alle beschriebenen Verfahren sind optional und können je nach Einzelfall pro Instrument angewendet werden.

2.7.1 Automatisches Stimmen der Aufnahmen

Für den Fall, dass die Aufnahmen eines Instrumentes nicht exakt der jeweiligen Grundfrequenz G entsprachen, weil das Instrument verstimmt war, haben wir zwei Algorithmen implementiert um dies zu korrigieren.

Der `auto-tune` Algorithmus verwendet als Basis für die Schätzung der tatsächlichen Grundfrequenz G' die bei der Analyse ermittelten Frameparameter (A_p^l, F_p^l). Es werden nur die Frames in der Mitte der Aufnahme verwendet, deren Frameindex zwischen 40% und 60% der Gesamtzeit liegt. Für jeden dieser Frames wird die Amplitude und Frequenz der im Frame enthaltenen Grundfrequenz (a_t, f_t) ermittelt. Dazu werden die Teiltöne im Bereich

$$0{,}8G \leq F_p^l \leq 1{,}25G$$

betrachtet. Der Teilton p mit der größten Amplitude A_p^l wird ausgewählt, und so das Paar $(a_t, f_t) = (A_p^l, F_p^l)$ bestimmt. Es kann passieren, dass ein Frame l gar keinen Teilton im geforderten Bereich enthält. In diesem Fall wird der Frame nicht verwendet. Es ergeben sich damit T Amplituden $a_1, a_2, ..., a_t$ und Frequenzen $f_1, f_2, ..., f_t$. Aus diesen Werten schätzen wir die (verstimmte) Grundfrequenz G' als gewichtete Summe der Frequenzen.

$$G' = \frac{\sum_{t=1}^{T} a_t f_t}{\sum_{t=1}^{T} a_t}$$

Schließlich korrigieren wir in allen Frames die Frequenzen.

$$F_p'^l = \frac{G}{G'} F_p^l$$

Durch diese Anpassung werden die Analysedaten so verändert, dass sie tatsächlich die Grundfrequenz G erhalten. Allerdings passen nach der Veränderung die Phasen Φ_p^l nicht mehr zu den anderen Parametern. Da weder bei der Synthese noch beim Morphing die Phasen verwendet werden, ist dies normalerweise kein Problem.

Alternativ dazu kann der `tune-all-frames` Algorithmus verwendet werden. Während der `auto-tune` Algorithmus aus vielen Frames eine Gesamtgrundfrequenz schätzt, und alle Frequenzen aller Frames mit dem selben Korrekturfaktor G/G' anpasst, wird beim `tune-all-frames` Algorithmus die

Grundfrequenz jedes Frames individuell ermittelt, und jeder Frame so korrigiert, dass er die identische Grundfrequenz erhält. Dies eignet sich nicht für jedes Instrument, da beispielsweise ein in der Originalaufnahme enthaltenes Vibrato durch diese Korrektur entfernt wird.

Implementiert ist dies so, dass für jeden Frame zunächst wieder die Teiltöne im Bereich

$$0{,}8G \leq F_p^l \leq 1{,}25G$$

betrachtet werden. Es ergibt sich die Grundfrequenz des Frames f_l als die Frequenz des Teiltons p mit der größten Amplitude. Falls der Frame l gar keinen Teilton im geforderten Bereich enthält, wird dieser Frame gar nicht korrigiert. Ansonsten werden alle Frequenzen des Frames angepasst, und zwar als

$$F_p'^l = \frac{G}{f_l} F_p^l$$

Auch bei diesem Verfahren passen nach der Veränderung die Phasen Φ_p^l nicht mehr zu den anderen Parametern, was wie gesagt normalerweise kein Problem ist.

Zusammengefasst wird also für jede Aufnahme die Midi-Note und damit die Grundfrequenz G angegeben. Für jedes Instrument (welches aus mehreren Aufnahmen besteht) kann zusätzlich einer der Algorithmen `auto-tune` oder `tune-all-frames` für das automatische Stimmen der Aufnahmen verwendet werden, sofern dies gewünscht ist.

Eine Alternative zu unserem Ansatz beim `auto-tune` Algorithmus erst G anzugeben und dann die Analysedaten zu verwenden um die tatsächliche Grundfrequenz G' zu bestimmen, wäre diese beiden Schritte zu ersetzen und direkt die Grundfrequenz der Aufnahme automatisch zu ermitteln. Ein Überblick von Algorithmen, die die Grundfrequenz automatisch aus den Daten bestimmen, findet sich zum Beispiel in [Ger03].

2.7.2 Automatisches Setzen der Loopparameter

Bei manchen Instrumenten kann der Loop sehr einfach definiert werden, indem der `auto-loop` Algorithmus verwendet wird. Dieser erhält als Parameter eine Prozentzahl P, also beispielsweise `auto-loop 50`. Nun berechnet der Algorithmus für jede Aufnahme einen einzelnen Frameindex l aus dieser Zahl. Dann wird der `loop-type` auf „Frame Vorwärtsloop“ gesetzt, `loop-start` und `loop-end` auf l, sodass der l-te Frame immer wieder und wieder abgespielt wird. Dieses Verfahren erspart die manuelle Definition der Loopparameter, eignet sich aber nur für Klänge die sehr gleichförmig sind. Zwei Instrumente bei denen wir mit diesem Algorithmus sinnvolle Ergebnisse erhalten haben waren „Trompete“ und „Altsaxophon“.

2.7.3 Lautstärkenormalisierung

Um beim Abspielen zu erreichen, dass alle Aufnahmen aller Instrumente etwa die gleiche Lautstärke haben, gibt es in unserer Implementation zwei Algorithmen, die zur Lautstärkenormalisierung eingesetzt werden können. Beide operieren auf den durch die Synthese entstehenden Ausgabewerten, die wir hier mit $x(n)$ bezeichnen.

Der `auto_volume` Algorithmus erhält als Parameter eine Prozentzahl P, und verwendet die Ausgabewerte der Synthese - ohne dass der Loop abgespielt wird - als Basis für die Schätzung der Signalenergie E. Die beiden Grenzen L und R sind dabei relativ zur Gesamtlänge der Ausgabe N

definiert, und liegen um die Prozentzahl P im Bereich $[P-5\%, P+5\%]$:

$$L = \frac{P-5}{100}N$$
$$R = \frac{P+5}{100}N$$

Die Signalenergie im Bereich $[P-5\%, P+5\%]$ ist damit

$$E = \frac{\sum_{n=L}^{R} x(n)^2}{R-L+1}$$

Bei der Normierung wird der Zielwert für die Energie E^* von 0,05 angestrebt, dies ist empirisch als günstig ermittelt worden. Die normierten Amplituden und Noisebandparameter ergeben sich damit als

$$E^* = 0{,}05$$
$$A_p'^l = A_p^l \sqrt{\frac{E^*}{E}}$$
$$NOISE_b'^l = NOISE_b^l \sqrt{\frac{E^*}{E}}$$

Der `auto_volume_from_loop` Algorithmus verwendet zur Berechnung der Energie die durch die Synthese entstandenen Ausgabewerte - allerdings mit Wiedergabe des Loops. Als linke und rechte Grenze ergeben sich

$$L = COUNT$$
$$R = max(2 \cdot COUNT, MIX\_FREQ)$$

wobei $COUNT$ die Anzahl der Samples in der Originalaufnahme ist, und MIX_FREQ die Abtastrate. Durch das Maximum $max()$ wird sichergestellt, dass genügend Daten zur Schätzung der Energie vorhanden sind. Durch Festlegung der linken Grenze L auf $COUNT$ wird erreicht dass alle Werte die zur Schätzung verwendet werden nach dem Ende der Originalaufnahme liegen, und damit notwendigerweise in dem durch den Loop erzeugten Bereich der Ausgabe.

Die weiteren Schritte, die Berechnung der Signalenergie E, und die Neuberechnung der Amplituden $A_p'^l$ sowie der Noisebandwerte $NOISE_b'^l$ sind identisch mit den Schritten des `auto_volume` Algorithmus.

3 Synthese der Ausgabe

3.1 Überblick über das Syntheseverfahren

3.1.1 Ziele der Synthese

Die im Kapitel 2 beschriebene Analyse wird für jede Aufnahme nur einmal als Vorbereitung durchgeführt und darf daher nahezu beliebig viel Rechenzeit benötigen. Im Gegensatz ist unsere Anforderung an die Synthese, dass die Klänge in Echtzeit produziert werden können, sodass beispielsweise ein Musiker an einem MIDI-Keyboard spielt, und die Software live den entsprechenden Klang berechnet. Da auch mehrere Noten gleichzeitig gespielt werden können, muss die Software so performant sein, dass eine akzeptable Polyphonie erreicht werden kann.

Ziel dieses Kapitels ist daher nicht nur zu beschreiben, wie man prinzipiell aus den Analysedaten wieder Ausgabesamples berechnen kann, wir werden darüber hinaus an verschiedenen Stellen darauf eingehen wie dies effizient zu erreichen ist. Außerdem werden wir einige Benchmarkergebnisse anführen, an den Stellen, an denen es sinnvoll ist.

Ein weiteres Ziel ist es, zu ermöglichen dass nicht nur die unveränderten Analysedaten abgespielt werden können, sondern auch veränderte Versionen davon, die typischerweise von dem im Kapitel 4 beschriebenen Morphing stammen. Normalerweise liefert die Analyse für die Amplituden, Frequenzen und Phasen eines Frames die Parameter

$$(A_p, F_p, \Phi_p)$$

sodass sich der deterministische Anteil eines Frames einfach als

$$d(t) = \sum_{p=1}^{P} A_p cos\left(2\pi \frac{F_p}{F_s} + \Phi_p\right)$$

berechnen lässt. Wenn nun aber F_p durch das Morphing als gewichtete Kombination von zwei Frequenzen zweier Ausgangssignale entsteht, entwickelt sich die Phase anders als in den Originalaufnahmen. Schon die Anwendung des `auto-tune` Algorithmus aus Kapitel 2.7.1 hat diesen Effekt.

Daraus entsteht die Anforderung an den Synthesealgorithmus, dass nur die Amplituden und Frequenzen A_p und F_p verwendet werden, und die passende Phase automatisch errechnet wird. Die ausschließliche Rekonstruktion des Signals aus Amplitude und Frequenz (ohne Phase) wird in beispielsweise in [MQ84] und [Ser89] beschrieben.

3.1.2 Aufbau der Synthese

Ähnlich wie bei der Analyse findet die Rekonstruktion des Signals durch die Berechnung einzelner Syntheseframes statt. Als Länge der Syntheseframes wird immer eine Zweierpotenz verwendet, um die FFT performant berechnen zu können.

Die Frames werden nach dem Overlap-Add-Verfahren (Überlappen und Addieren) mit einem Hann-Synthesewindow $w(t)$ zusammengesetzt. Aus Effizienzgründen ist es sinnvoll, die Syntheseframes so zu überlappen, dass immer die Hälfte der Framelänge als Schrittweite verwendet wird. Dann addieren sich die Fensterfunktionen der Frames auch exakt zu eins. Ein Beispiel mit einer Framelänge N von 1024 und einer zugehörigen Schrittweite von 512 findet sich in Abbildung 3.1.

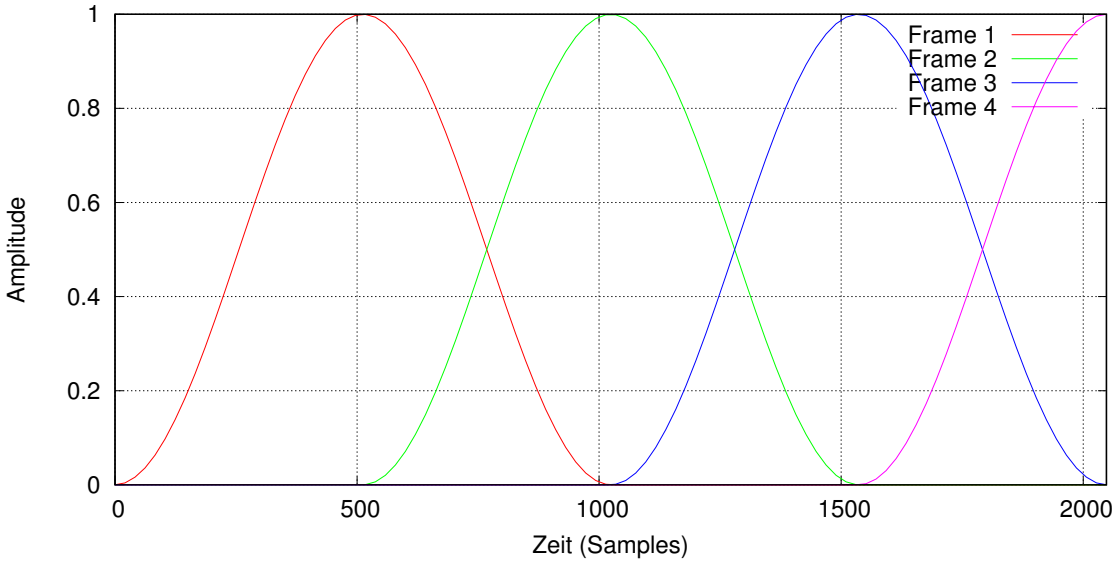


Abbildung 3.1: Synthesefenster $w(t)$ der Frames

Wir berechnen sowohl den deterministischen Anteil eines Frames als auch den stochastischen Anteil eines Frames im Frequenzbereich. Das Spektrum des Frames wird zunächst mit Nullen initialisiert. Dann wird für jeden Sinusanteil ein Peak im Spektrum eingetragen. Wir folgen hierbei der in [RD92] beschriebenen Technik. Danach wird der stochastische Anteil auf das Spektrum aufaddiert. Schließlich transformieren wir den Frame mit einer einzelnen inversen FFT zurück in den Zeitbereich und vervollständigen die Ausgabe durch Overlap-Add und die Berechnung des Envelopes.

3.2 Konvertierung der 16 Bit Repräsentation

Um in den Dateien und im Speicher Platz zu sparen, hatten wir in Kapitel 2.5 beschrieben, wie wir die einzelnen Werte für Frequenzen, Amplituden, Phasen und Noise jeweils mit 16 Bit dargestellt werden können. Die Phasen werden von der Synthese nicht verwendet. Die verbleibenden Werte müssen aber während der Synthese wieder in 32 Bit Floats konvertiert werden. Wir beschreiben im Folgenden wie dies effizient möglich ist.

Später geben wir unsere Synthesealgorithmen relativ zu den bereits konvertierten Werten an.

3.2.1 Konvertierung der Frequenzen

Zuvor wurde die Umrechnung von einer 16 Bit Frequenz F_{16} als

$$\Delta F = e^{F_{16}/6000-3}$$

angegeben, wobei ΔF immer ein Faktor relativ zur Grundfrequenz des Tons ist. Im Hinblick auf die Effizienz ist vor allem die Berechnung der Exponentialfunktion problematisch. Unsere Berechnung basiert auf der Regel:

$$e^{a+b} = e^a \cdot e^b$$

Wir zerlegen unseren 16 Bit Wert F_{16} in zwei 8 Bit Werte (an dieser Stelle ist Integerdivision ohne Nachkommastellen gemeint):

$$H = F_{16}/256$$
$$L = F_{16} \bmod 256$$

Es gilt nun:

$$\begin{aligned}\Delta F &= e^{F_{16}/6000-3} \\ &= e^{(256\cdot H+L)/6000-3} \\ &= e^{256\cdot H/6000-3+L/6000} \\ &= e^{256\cdot H/6000-3} \cdot e^{L/6000}\end{aligned}$$

Mit zwei 256-elementigen Tabellen

$$\begin{aligned}delta\_f\_high[x] &= e^{256\cdot x/6000-3} \\ delta\_f\_low[x] &= e^{x/6000}\end{aligned}$$

ergibt sich schließlich die schnellere Berechnung

$$\Delta F = delta\_f\_high[H] \cdot delta\_f\_low[L]$$

3.2.2 Konvertierung der Amplituden

Für die Umrechnung einer 16 Bit Amplitude gilt:

$$\begin{aligned}db2factor(x) &= 10^{x/20} \\ A &= db2factor\left(\frac{A_{16}}{64} - 512\right)\end{aligned}$$

Wie bei den Frequenzen können wir zwei 256-elementige Tabellen verwenden

$$\begin{aligned}a\_high[x] &= db2factor\left(\frac{x\cdot 256}{64} - 512\right) \\ a\_low[x] &= db2factor\left(\frac{x}{64}\right)\end{aligned}$$

und damit durch Zerlegung von A_{16} in die zwei 8 Bit Werte H und L die effizientere Berechnung benutzen:

$$A = a\_high[H] \cdot a\_low[L]$$

Da wir die Noisewerte auch auf die gleiche Weise - als Amplituden - gespeichert haben, sind damit alle für die Synthese notwendigen Konvertierungen erklärt.

3.3 Synthese des deterministischen Anteils

3.3.1 Ausgangsmaterial und Antialias Filter

Als Ausgangsmaterial verwenden wir die in der Analyse ermitteln Frequenzen und Amplituden. Wenn ein Ton mit einer festgelegten Zielfrequenz Z gespielt werden soll, suchen wir im Ausgangsmaterial nach einer Aufnahme mit dieser Grundfrequenz. Es kann vorkommen, dass es keine solche Aufnahme gibt. In diesem Fall verwenden wir die Aufnahme, deren Grundfrequenz G am nächsten an der Zielfrequenz liegt, und skalieren alle Frequenzen. Die Amplituden und Noisebandwerte bleiben gleich.

$$\begin{aligned}F_p &= \frac{Z}{G}F_p^* && \text{für } 1 \le p \le P \\ A_p &= A_p^* && \text{für } 1 \le p \le P\end{aligned}$$

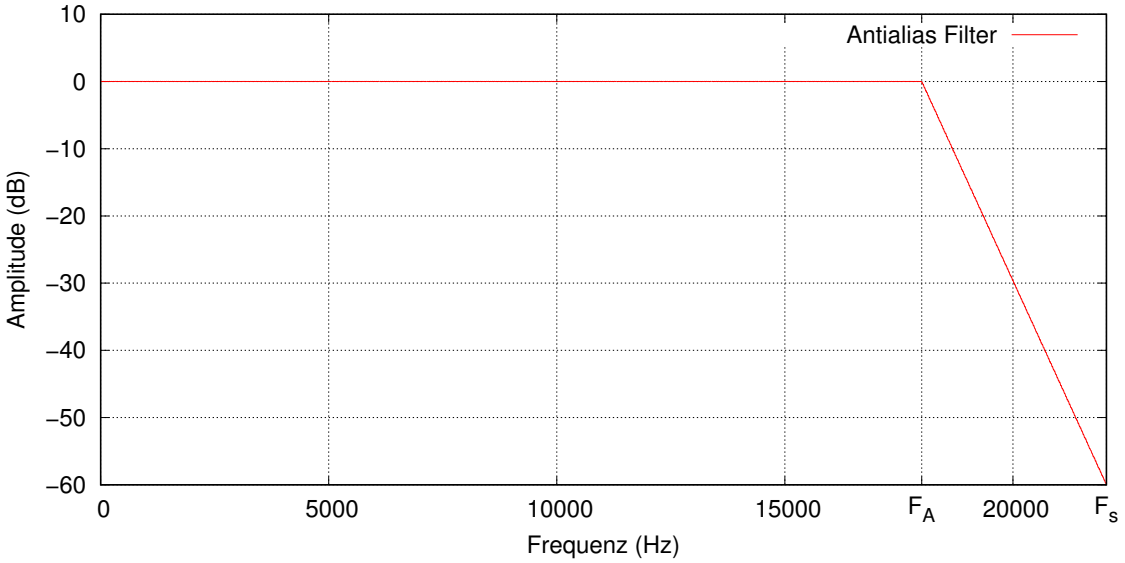


Abbildung 3.2: Antialias-Filter für hohe Frequenzen

Die Berechnung der Phasen wird später in Kapitel 3.3.4 behandelt.

Es gibt zwei Gründe, weshalb die Frequenz F_p oberhalb der Nyquist-Frequenz

$$F_N = \frac{F_s}{2}$$

liegen kann. Der eine ist, dass durch die oben genannte Frequenzskalierung alle Frequenzen mit dem Faktor Z/G skaliert werden. Wenn beispielsweise $Z = 4400$ Hz und $G = 440$ Hz ist, werden alle Frequenzen mit dem Faktor 10 skaliert. Ist in der Aufnahme eine Frequenz von $F_p^* = 3000$ Hz enthalten, würde sich für die Synthese ein Wert von $F_p = 30000$ Hz ergeben.

Auch wenn das Originalsignal mit einer höheren Abtastrate aufgenommen wurde als die Wiedergaberate, können Frequenzen mit $F_p \geq F_N$ auftreten.

In beiden Fällen werden Frequenzen mit $F_p \geq F_N$ entfernt. Damit dies nicht hörbar ist, werden die Amplituden ab er Frequenz von

$$F_A = \frac{18000}{44100} F_s$$

mit einem frequenzabhängigen Faktor von 0 dB bei F_A auf -60 dB bei F_s abgesenkt. Ist die Abtastrate $F_s = 44100$ Hz, so ist $F_A = 18000$ Hz, sodass von der Filterung nur Frequenzen betroffen sind, die man normalerweise ohnehin nicht hört. Bei höheren Abtastraten verschiebt sich F_A entsprechend nach oben. Der Frequenzgang des Filters ist in Abbildung 3.2 dargestellt.

3.3.2 Berechnung des Spektrums der Teiltöne

Wir verwenden aus Effizienzgründen für die Berechnung des deterministischen Anteils die in [RD92] beschriebenen Technik. Dabei wird das Spektrum eines Frames aus den einzelnen Peaks der Teiltöne zusammengesetzt. Danach werden die Werte der Sinusanteile im Zeitbereich durch eine inverse FFT bestimmt.

In Kapitel 2.2.6 haben wir bereits gezeigt, dass die Transformierte einer einzelnen komplexen Sinusschwingung

$$x(n) = Ce^{j\omega_x n}$$

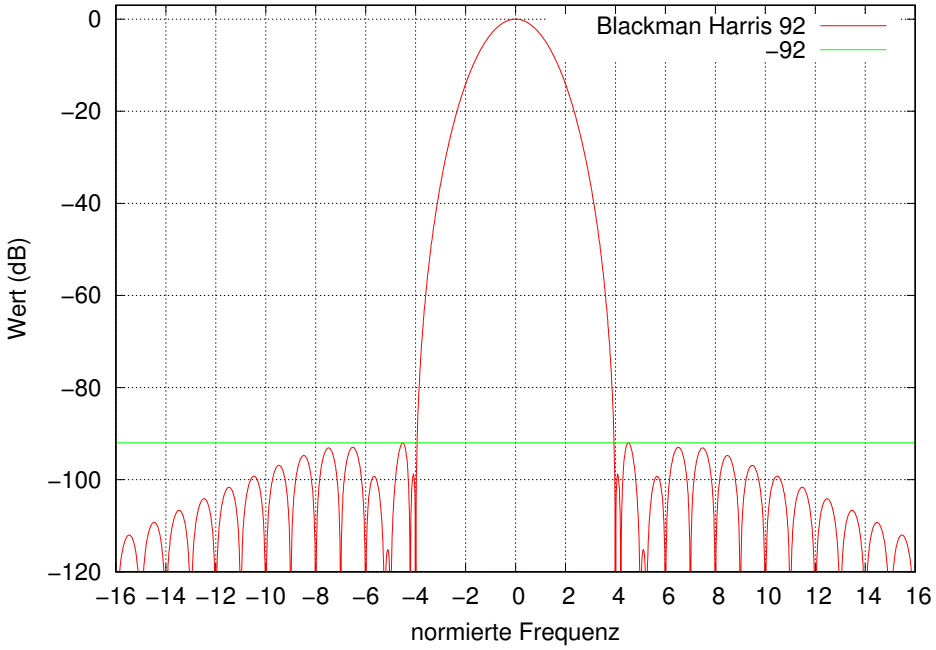


Abbildung 3.3: Transformierte des Blackman-Harris-92 Fensters

mit einer Fensterfunktion $w(n)$ folgendes Spektrum ist:

$$X(\omega) = CW(\omega - \omega_x)$$

In dieser Darstellung ist C komplex, und bestimmt so sowohl die Amplitude A als auch die Phase Φ der Sinusschwingung.

W ist die Transformierte der Fensterfunktion, wobei diese bei einer Fensterfunktion, die um ihre Mitte symmetrisch ist, reell ist.

Wählen wir als Fensterfunktion das Blackman-Harris-92-Fenster, zu finden in [Har78], konzentriert sich die Energie jedes Sinusanteils bereits in insgesamt 9 Werten für jeden Peak. In Abbildung 3.3 ist die Transformierte des Blackman-Harris-92 Fensters inklusive einer Linie in Höhe von -92 dB dargestellt. Man sieht dass die links und rechts von -4 bzw. 4 liegenden Anteile sehr klein sind, sodass sie bei der Synthese nicht berücksichtigt werden müssen.

Für die eigentliche Implementation verwenden wir für die Werte von W eine Tabelle, die wir aufbauen indem wir die FFT des Blackman-Harris-92 Fensters berechnen welches wir mit einem Zeropadding von 256 aufgefüllt haben, um auch die in der Berechnung von $W(\omega - \omega_x)$ auftretenden Zwischenwerte berechnen zu können. Durch die Tabellierung wird damit keine exakte Lösung mehr erreicht, die Frequenzen werden auf die durch diese Vereinfachung möglichen Werte quantisiert. Dabei liegt der maximale Fehler bei dem von uns gewählten Zeropadding von 256 bei

$$quantize\_freq\_delta = \frac{1}{2} \cdot \frac{mix\_freq}{256 \cdot block\_size}$$

In einem typischen Fall, bei der Abtastrate und Blockgröße

$$mix\_freq = 48000$$
$$block\_size = 1024$$

liegt der durch die Frequenzquantisierung verursachte Fehler damit im Bereich von

$$-0{,}0916 \text{ Hz} \leq freq\_err \leq 0{,}0916 \text{ Hz}$$

Die Berechnung ermittelt für jeden Sinusanteil zunächst aus der Amplitude A und der Phase Φ einmal die komplexe Zahl C:

$$C_{re} = A \cdot cos(\Phi)$$
$$C_{im} = A \cdot sin(\Phi)$$

Durch die Verwendung einer Sinustabelle können $cos(\Phi)$ und $sin(\Phi)$ jeweils der Tabelle entnommen werden. In unserer Implementation hat diese Tabelle 4096 Werte. Da die Tabelle der Werte von W reell ist, muss diese komplexe Zahl C nur mit den passenden 9 Werten für die entsprechenden Einträge von W multipliziert werden. Die Ergebnisse werden in das Spektrum aufaddiert.

Als weitere Optimierung speichern wir die Tabelle der Werte von W so, dass die für einen Peak benötigten Werte jeweils nebeneinander liegen.

3.3.3 Rücktransformation und Skalierung

Die Rücktransformation des Spektrums mittels einer inversen FFT ergibt die Summe aller Sinusanteile im Zeitbereich. Da wir die Tabelle W aus dem Blackman-Harris-92-Fenster berechnet haben, erhalten wir im Zeitbereich das Produkt aus dem Sinusanteil und dieser Fensterfunktion.

Um dies zu korrigieren, verwenden wir eine Tabelle $win\_scale[n]$. Diese wird berechnet, indem die gewünschte Synthesefensterfunktion $w[n]$ durch den entsprechenden Wert der Blackman-Harris-92 Fensterfunktion geteilt wird.

$$win\_scale[n] = \frac{w[n]}{bh\_92[n]} \text{ mit } 0 \leq n < N$$

Wir multiplizieren die Ausgabe der Synthese mit dieser Tabelle, und erhalten dadurch die Sinusanteile mit der gewünschten Fensterfunktion, in unserem Fall ist $w[n]$ wie oben gesagt das Hanning Fenster.

3.3.4 Bestimmung der Phasen für die Synthese

Es gibt zwei Gründe aus denen wir die Phasen für die Synthese neu berechnen müssen, und nicht die Phasen der Analyse übernehmen können. Einerseits haben unsere Syntheseframes im Allgemeinen eine andere Länge als die Analyseframes. Daher können die Phasen nicht einfach übernommen werden. Andererseits findet im Normalfall eine Kombination mehrerer Instrumente mittels dem später beschriebenen Morphing statt. In diesem Fall gibt es keine direkte Entsprechung der in der Synthese benötigten Phase zu den Phasen des Ausgangsmaterials.

Insofern bestimmen wir immer während der Synthese für jeden Sinusanteil die Phase, vergleichbar mit dem in [MQ84] beschriebenen Verfahren. Unser Algorithmus setzt voraus, dass die P Sinusanteile des aktuellen Syntheseframes

$$(A_p, F_p) \text{ für } 1 \leq p \leq P$$

und die Q Anteile des vorherigen Syntheseframes

$$(A_q^*, F_q^*, \Phi_q^*) \text{ für } 1 \leq q \leq Q$$

in sortierter Form vorliegen. Dabei haben die Parameter des vorherigen Syntheseframes bereits eine Phase (die im letzten Schritt bestimmt wurde), die Phasen Φ_p sind zu berechnen. Beim ersten Syntheseframe ist der vorige Frame leer, also in diesem Fall gilt $Q = 0$.

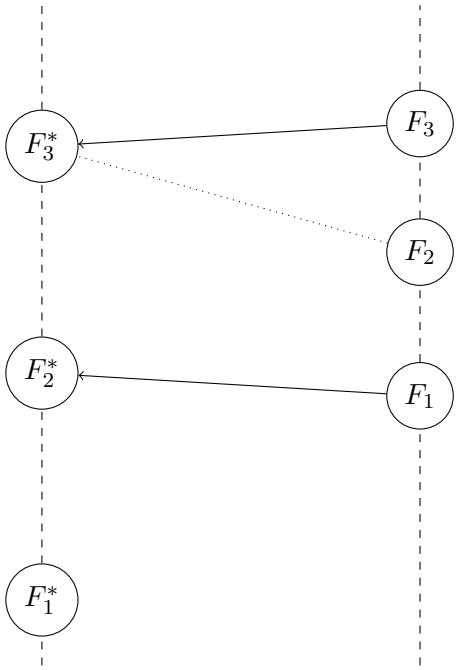


Abbildung 3.4: Algorithmus zur Bestimmung der Phasen $\Phi_1, ..., \Phi_P$

Da beide Parametersätze sortiert sind, können wir die Amplituden A_p und Frequenzen F_p einen Eintrag nach dem anderen durchgehen, und gleichzeitig immer die nächstgelegene Frequenz F_q^* finden, die Berechnungskomplexität bleib dabei linear: $O(\max(P, Q))$.

Sei F_q^* die nächstgelegene Frequenz zu F_p. Liegt F_p nahe an F_q^*, genauer gesagt, wenn

$$0{,}95 \cdot F_q^* \leq F_p \leq 1{,}05 \cdot F_q^*$$

dann gehen wir davon aus, dass F_p den Sinusanteil des letzten Frames mit der Frequenz F_q^* fortsetzt. Dann berechnet sich die Phase Φ_p am Anfang des aktuellen Syntheseframes, indem die Phase Φ_q um $\frac{N}{2}$ Samples mit der Frequenz F_q erhöht wird:

$$\Phi_p = normalize\_phase\left(\Phi_q + \frac{N}{2}\frac{2\pi F_q}{F_s}\right)$$

Gibt es keine entsprechende Frequenz F_q zu F_p setzen wir

$$\Phi_p = 0$$

Die Funktion $normalize\_phase(x)$ ist hierbei eine Funktion, die die Phase in den Bereich $[0, 2\pi]$ bringt.

In Abbildung 3.4 ist ein Beispiel für die Berechnung der Phasen dargestellt. Hierbei ist die nächstgelegene Frequenz zu F_1 aus dem vorigen Frame F_2^*. Die beiden Frequenzen liegen so nah zusammen, dass die Phase Φ_1 aus Φ_2^* berechnet wird. Für F_2 ist die nächstgelegene Frequenz F_3^*. Jedoch ist der Abstand beider Frequenzen so groß, dass $\Phi_2 = 0$ gesetzt wird. Für F_3 ist die ähnlichste Frequenz ebenfalls F_3^*. Die beiden Frequenzen sind so ähnlich, dass Φ_3 wiederum aus Φ_3^* errechnet wird.

3.4 Synthese des stochastischen Anteils

3.4.1 Effiziente Generierung der Zufallszahlen

Für die Erzeugung des Spektrums des stochastischen Anteils werden komplexe Werte mit einem festen Betrag, aber mit zufälliger Phase benötigt. Da dafür relativ viele Zufallszahlen benötigt werden, betrachten wir in diesem Kapitel gesondert die effiziente Generierung dieser Werte. Eine große Vereinfachung ergibt sich dadurch, dass es ausreicht, für jede Phase einen zufälligen 8 Bit Wert zu verwenden. Gesucht sind also $R_1, R_2, R_3, ...R_N$ mit $R_n \in [0{,}255]$.

In Tabelle 3.1 werden die gemessenen Geschwindigkeiten verschiedener Implementationen von Zufallszahlengeneratoren auf einem 64-bit Linux System (AMD Phenom(tm) 9850 Quad-Core Processor) aufgelistet.

Bibliothek	**Generator**	**Ausgabe (Bit)**	**Geschwindigkeit**
libc	`rand`	31	74,18
	`rand_r`	31	108,93
	`random`	31	84,51
	`random_r`	31	190,35
glib	`g_random_int`	32	39,38
	`g_rand_int`	32	116,02
boost	`mt19937`	32	169,42
	`rand48`	31	357,91
	`minstd_rand`	31	131,87
	`taus88`	32	278,39
rapicorn	`pcg32`	32	417,16

Tabelle 3.1: Geschwindigkeiten verschiedener Zufallszahlengeneratoren

In der ersten Spalte ist jeweils die Bibliothek angegeben, in der sich die Implementation des Zufallsgenerators findet. Hierbei ist mit *libc* die Standard C Bibliothek gemeint, die also immer verfügbar ist. Die Bibliotheken *glib*, *boost* und *rapicorn* werden in unserer Software auch aus anderen Gründen benutzt, sind also auch ohne Mehraufwand verfügbar. Jede dieser Bibliotheken stellt unterschiedliche Methoden zur Generierung von Zufallszahlen zur Verfügung, deren Namen in der zweiten Spalte zu finden ist. In der dritten Spalte findet sich die Länge der Ausgabe der erzeugten Zufallszahlen in Bit.

In der letzten Spalte wird angegeben, wieviele Millionen Werte pro Sekunde der Generator erzeugen kann. Je höher dieser Wert ist, desto besser für die Gesamtperformance der Synthese des stochastischen Anteils.

Man kann die Implementationen grob in zwei Kategorien unterteilen. Die Generatoren `rand`, `random` und `g_random_int` operieren auf einem globalen Zustand, der für alle Threads des gesamten Programmes gilt. Dadurch ist es notwendig, für jede generierte Zahl einen Lock zu akquirieren und wieder freizugeben. Dadurch sind diese Generatoren im Verhältnis langsamer. Die Varianten `rand_r`, `random_r` und `g_rand_int` korrespondieren zu den erstgenannten, verwenden aber für jeden Generator einen eigenen Zustand, der Lock entfällt, und man sieht, dass die Geschwindigkeit

jeweils der Variante mit globalem Zustand überlegen ist.

Alle anderen Implementationen operieren ebenfalls mit einem eignen Zustand für jeden Generator, benötigen also keinen Lock.

Eine weitere Unterteilung ergibt sich dadurch, dass die in der *libc* und in *glib* implementierten Generatoren als Funktion implementiert wurden, sodass jede generierte Zufallszahl einen Funktionsaufruf bedeutet. Die in der C++-Template-Bibliothek *boost* enthaltenen Generatoren werden jedoch vollständig in Headerdateien definiert, sodass der Compiler die gesamte Berechnung inlinen kann. Dadurch ist zu erwarten dass diese - sofern die Komplexität der Algorithmen vergleichbar sind - zu den schnellsten Verfahren zählen werden. Auch der `pcg32` Generator der *rapicorn* Bibliothek wird in C++-Headern definiert, genießt also auch diesen Vorteil.

Um das Problem der Generierung der 8 Bit Zufallszahlen $R_1, R_2, R_3, ...R_N$ zu lösen betrachten wir die schnellsten drei Implementationen, `pcg32`, `rand48` und `taus88`. Erwartungsgemäß sind diese inline-fähig und ohne Lock implementiert. Die beiden Generatoren `pcg32` und `taus88` haben den Vorteil, dass mit einem Aufruf jeweils 32 Bit erzeugt werden. Da wir 8 Bit Zahlen benötigen, können mit einem Aufruf immer vier 8 Bit Werte erzeugt werden. Mit dem `rand48` Verfahren ist dies nicht so einfach möglich, da hierbei immer nur 31 Bit generiert werden. Man könnte nur entweder jeweils drei 8 Bit Werte erzeugen oder mehrere Ausgabewerte geeignet kombinieren. Dies würde aber zusätzliche Kosten verursachen. Insofern verwerfen wir den `rand48` Generator als Möglichkeit.

Ob man den `taus88` Generator oder den `pcg32` Generator als Basis für die Generierung der Zufallszahlen verwendet, ist eine Abwägung. Der Vorteil des `taus88` Generators ist, dass er in einer etablierten C++ Standardbibliothek zu finden ist, der *boost* Bibliothek. Damit ist die langfristige Verfügbarkeit und die Korrektheit der Implementation relativ sicher. Allerdings ist der `pcg32` Generator deutlich schneller, daher benutzen wir diesen für die Erzeugung der Zufallszahlen.

Um die gewünschten N 8 Bit Werte zu berechnen, füllen wir ein Array mit

$$M = \left\lfloor \frac{N+3}{4} \right\rfloor$$

zufälligen 32 Bit Werten $R_1^*, R_2^*, ...R_M^*$. Dieses kann dann Byte für Byte ausgelesen werden und so ergeben sich $R_1, R_2, R_3, ...R_N$.

Die mathematischen Grundlagen des `taus88` Generators finden sich in [Ecu96]. Der `pcg32` Generators wird in [Nei15] beschrieben.

3.4.2 Erzeugung des Spektrums

In Kapitel 2.3.7 haben wir den stochastischen Anteil eines Frames als 32 Koeffizienten $NOISE_0$ bis $NOISE_{31}$ gespeichert, die angeben, wie stark der stochastische Anteil in den jeweiligen Frequenzbändern war. Bei der Synthese erzeugen wir ein Spektrum, welches dazu passt. Zu generieren sind also die k komplexen Zahlen $E(k)$ des Spektrums, wobei der Betrag $|E(k)|$ des jeweiligen Bandes fest ist. Wir nennen ihn A_b. Die Phase wird zufällig gewählt, die Generierung der Zufallszahlen wurde im vorigen Kapitel besprochen. Wir bestimmen also zunächst den normierten Betrag:

$$A_b = norm \cdot NOISE_b$$

$$norm = \frac{1}{2} \cdot \sqrt{\frac{mix\_freq}{0{,}375 \cdot N}}$$

Die Normierung $norm$ enthält den Faktor $1/2$. Dies liegt daran, dass wir die reelle Variante der FFT verwenden, wodurch wir nur die eine Hälfte des Spektrums erzeugen, die andere Hälfte ergibt

sich durch die in Kapitel 2.2.2 erwähnte Symmetrie. Die Energie des Noisebands wird also auf die FFT Bins $E(k)$ die wir generieren und auf die durch Symmetrie festgelegten $E(N-k)$ verteilt, sodass wir nur die jeweils die Hälfte der Energie auf die Eingabe der FFT verteilen.

Die Abtastrate mix_freq wird bei der Normierung einbezogen, da bei der Synthese möglicherweise eine andere Abtastrate als bei der Analyse verwendet wird.

Bei der Synthese wird als Fensterfunktion $w(n)$ immer das Hann-Fenster verwendet. Daher kann man den Einfluss des Fensters direkt als Faktor angeben. Wie bei der Analyse lautet die Frage, wie die Energie des n-ten Ausgabewertes $x(n)$ durch die Multiplikation mit dem Synthesefenster $w(n)$ verändert wird. Statt $x(n)^2$ ist die Energie $w(n)^2 \cdot x(n)^2$. Im Mittel wird $x(n)^2$ also mit

$$wnorm^* = \frac{1}{N} \sum_{n=0}^{N-1} w(n)^2$$

multipliziert. Es ergibt sich als Spezialfall für Hann-Fenster etwa

$$wnorm^* \approx 0{,}375$$

Die von uns verwendete fftw-Bibliothek normiert weder die FFT noch die hier verwendete inverse Transformation, wie in Kapitel 2.2.4 beschrieben ist. Um zu erreichen, dass die Energie des Eingabespektrums und die Energie der Ausgabe übereinstimmen, verwenden wir zur Normierung den Faktor

$$fnorm = \frac{1}{\sqrt{N}}$$

Damit funktioniert die Synthese unabhängig von der Anzahl der gewünschten Ausgabesamples N. Nehmen wir an N verdoppelt sich (N ist immer eine Zweierpotenz). Dann gibt es (etwa) doppelt so viele Werte für das Spektrum $E(k)$ in jedem Frequenzband. Die Energie der Eingabe wird also doppelt so groß. Auf der anderen Seite produzieren wir doppelt so viele Samples, sodass auch die Energie der Ausgabe doppelt so groß sein muss.

Nachdem die Normierung erklärt wurde, nun zur Bestimmung der eigentlichen Werte eines Frequenzbandes. Wieder geben wir mit low_b und $high_b$ die Grenzen der 32 Bänder an ($b \in [0{,}31]$). Wir verwenden eine zufällige Phase

$$\Phi_k = random(0, 2\pi) \text{ das heißt } 0 \leq \Phi_k < 2\pi$$

Als Optimierung verwenden wir die in Kapitel 3.4.1 vorbereiteten 8 Bit Zufallszahlen.

$$\Phi_k = \frac{2\pi \cdot R_k}{256}$$

Wir setzen alle $E(k)$ des b-ten Frequenzbandes, das heißt $low_b \leq k < high_b$, aus dem Realteil und Imaginärteil zusammen.

$$E_{re}(k) = A_b \cdot cos(\Phi_k)$$
$$E_{im}(k) = A_b \cdot sin(\Phi_k)$$

Damit ist

$$E(k) = A_b(cos(\Phi_k) + j \cdot sin(\Phi_k))$$

und hat wie gewünscht die Phase Φ_k und den Betrag A_b. Gehört $E(k)$ zu keinem Band ($k < low_0$ oder $k > high_{31}$), dann setzen wir $E(k) = 0$.

Da für die Phase ohnehin nur 256 verschiedene Werte vorkommen, können wir als weitere Optimierung eine Tabelle aller 256 möglichen Sinuswerte anlegen

$$sin\_table[n] = sin\left(\frac{2\pi \cdot n}{256}\right) \text{ für } 0 \leq n < 256$$

Damit ergibt sich die effizientere Berechnung

$$E_{re}(k) = A_b \cdot sin\_table[(R_k + 64) \bmod 256]$$
$$E_{im}(k) = A_b \cdot sin\_table[R_k]$$

wobei die Modulo 256 Operation nicht teuer ist, da das Ergebnis der Addition einfach als Byte gespeichert werden kann.

3.4.3 Anwendung der Fensterfunktion im Zeitbereich

An diesem Punkt haben wir das Spektrum des gewünschten stochastischen Signals ermittelt. Ein Beispiel findet sich in Abbildung 3.5. Man sieht die Frequenzbänder, die breiter werden je höher die Frequenz ist. Das zugehörige Analysesignal $x(t)$ war in diesem Fall weißes Rauschen, sodass die Beträge der 32 Frequenzbänder A_b ähnlich sind. Die zufälligen Phasen Φ_k sieht man nicht, da nur die Beträge $|E(k)|$ dargestellt sind.

Wenn man dieses Spektrum mit der inversen FFT zurücktransformiert, erhält man das in Abbildung 3.6 dargestellte Zeitsignal $e(t)$. Die Multiplikation mit dem Hann-Synthesefenster $w(t)$ ergibt schließlich das in Abbildung 3.7 dargestellte Signal, was im Overlap-Add-Verfahren (Überlappen und Addieren) Frame für Frame zur gewünschten Ausgabe zusammengesetzt werden kann.

Es gibt allerdings einen Nachteil, wenn die Schritte in dieser Reihenfolge ausgeführt werden: man benötigt eine inverse FFT zur Synthese des deterministischen Anteils (der Sinusschwingungen), und eine weitere inverse FFT zur Synthese des stochastischen Anteils, der in diesem Kapitel behandelt wird.

Dass man beide Spektren nicht einfach addieren kann liegt daran, dass die Sinusschwingungen bereits im Spektrum in einer Form vorliegen, die bei der das für die Sinussynthese typische Blackman-Harris-92-Fenster angewendet wurde. Nach der Rücktransformation erhalten wir daher bereits ein mit einer Fensterfunktion multipliziertes Signal, welches wir dann noch weiter anpassen. Wie in Abbildung 3.6 zu sehen ist, hat das stochastische Signal jedoch nach der Rücktransformation keine Fensterfunktion, die Amplitude am Anfang, in der Mitte und am Ende bleibt gleich. In den nächsten beiden Kapiteln zeigen wir, wie das Noisespektrum $|E(k)|$ so weiterbehandelt werden kann, dass es zum Fenster des Sinusanteils passt, sodass wir beide, den deterministischen und stochastischen Anteil, mit einer einzigen inversen FFT berechnen können.

3.4.4 Anwendung der Fensterfunktion im Frequenzbereich

Die Multiplikation des durch inverse FFT aus $E(k)$ berechneten Zeitsignals $e(t)$ mit einer Fensterfunktion $w(t)$ lässt sich im Frequenzbereich als zyklische Faltung implementieren. Dazu wird die Transformierte der Fensterfunktion $W(m)$ benötigt.

$$E^*(k) = \sum_{m=-M}^{M} W(m)E((k-m) \bmod N)$$

Dabei ist die Berechnungskomplexität $O(N \cdot M)$ wobei N die Länge des Spektrums und M die Länge der Transformierten der Fensterfunktion ist. Ist M so groß wie N wird die Berechnungskomplexität $O(N^2)$. In diesem Fall lohnt sich die Berechnung der Fensterfunktion im Frequenzbereich nicht, da sie zu lange dauert.

In unserem Fall wollen wir das Blackman-Harris-92 Fenster benutzen, dabei ist M mit $M = 3$ relativ klein, alle $W(m)$ für $m \in [-3;3]$ sind reell. Die Berechnung der Modulo-Operation lässt

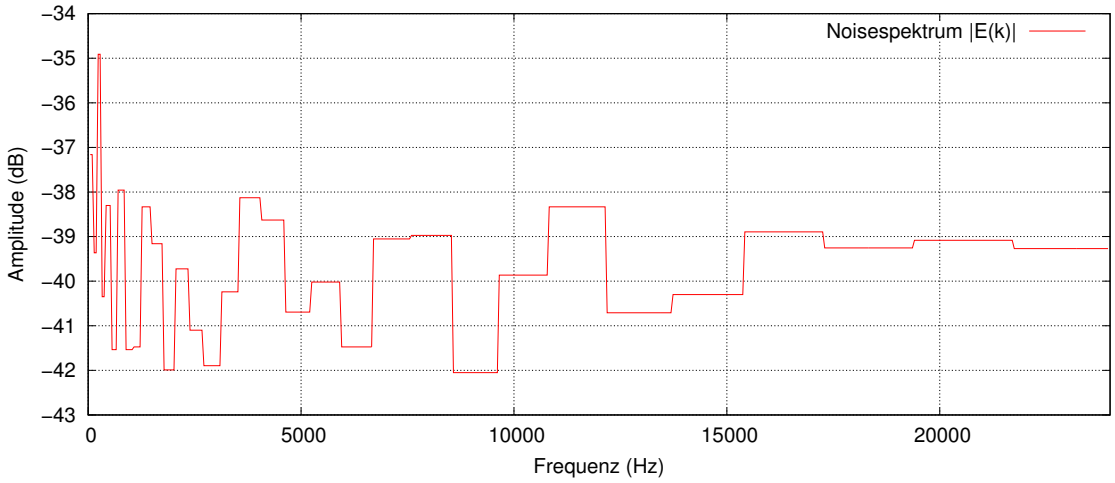


Abbildung 3.5: Noisespektrum $|E(k)|$

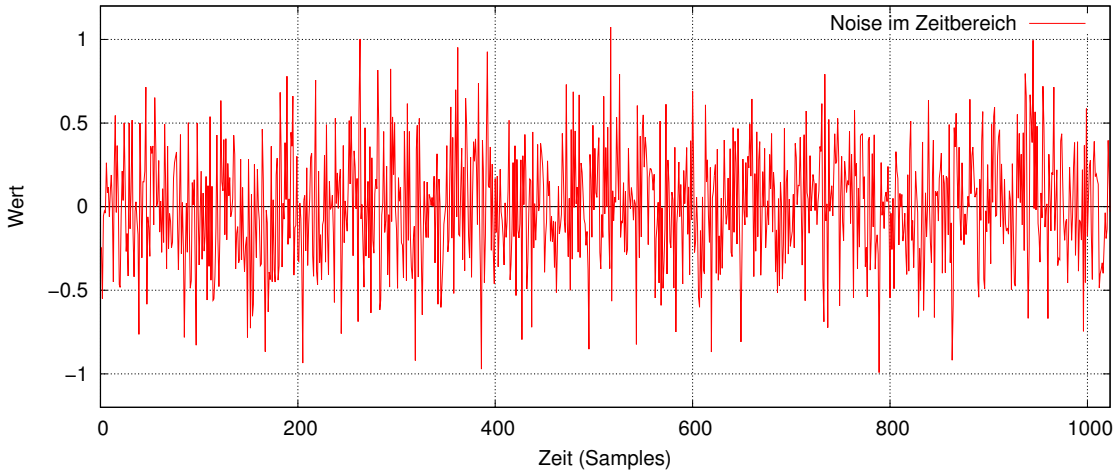


Abbildung 3.6: Noisesignal $e(t)$

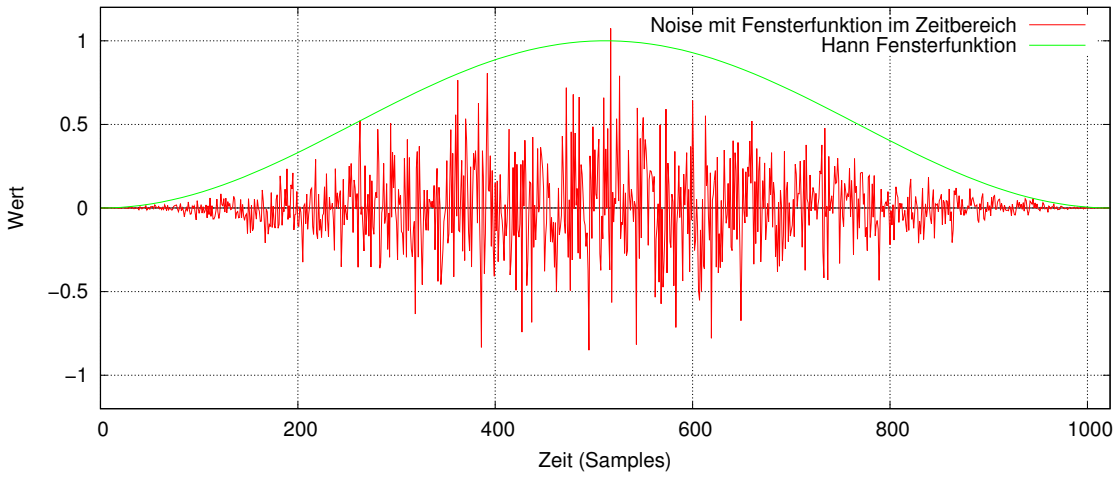


Abbildung 3.7: Noisesignal $e(t)$ mit Synthesefenster $w(t)$

Linker Rand: Werte für $k < 0$	**Rechter Rand: Werte für $k > \frac{N}{2}$**
$\tilde{E}(-1) = E(N-1) = \overline{E(1)}$	$\tilde{E}\left(\frac{N}{2}+1\right) = E\left(\frac{N}{2}+1\right) = \overline{E\left(\frac{N}{2}-1\right)}$
$\tilde{E}(-2) = E(N-2) = \overline{E(2)}$	$\tilde{E}\left(\frac{N}{2}+2\right) = E\left(\frac{N}{2}+2\right) = \overline{E\left(\frac{N}{2}-2\right)}$
$\tilde{E}(-3) = E(N-3) = \overline{E(3)}$	$\tilde{E}\left(\frac{N}{2}+3\right) = E\left(\frac{N}{2}+3\right) = \overline{E\left(\frac{N}{2}-3\right)}$

Tabelle 3.2: Behandlung der Ränder für die Funktion $\tilde{E}(k)$

sich vermeiden, indem stattdessen ein größeres Eingabespektrum $\tilde{E}(k)$ verwendet wird. Dieses entspricht $E(k)$, wobei an den Rändern die in Tabelle 3.2 angegebenen Werte aufgefüllt werden.

Da wir tatsächlich nur die Werte von $E(k)$ für $k \in \left[0; \frac{N}{2}\right]$ vorliegen haben, verwenden wir die in Kapitel 2.2.2 erwähnte Symmetrie für reelle FFTs und berechnen die gewünschten Werte über die konjugiert komplexen Werte.

Damit lässt sich die Berechnung der zyklische Faltung vereinfachen

$$E^*(k) = \sum_{m=-M}^{M} W(m)\tilde{E}(k-m)$$

Um die $E^*(k)$ zu berechnen werden die Werte $W(m)$ für $m \in [-3;3]$ benötigt, diese sind in Tabelle 3.3 angegeben. Sie lassen sich aus den in [Har78] angegebenen Koeffizienten a_0, a_1, a_2, a_3 des Fensters ermitteln.

Position	**Wert**
$W(0)$	$a_0 = 0{,}358\,75$
$W(-1) = W(1)$	$a_1/2 = 0{,}244\,145$
$W(-2) = W(2)$	$a_2/2 = 0{,}070\,64$
$W(-3) = W(3)$	$a_3/2 = 0{,}005\,84$

Tabelle 3.3: Transformierte $W(m)$ des Blackman-Harris-92 Fensters

3.4.5 Optimierte Faltung mit SSE Instruktionen

Alle modernen Intel und AMD Prozessoren, unabhängig davon ob es sich um 32-Bit oder 64-Bit CPUs handelt, unterstützen spezielle Erweiterungen, die sich gut für digitale Signalverarbeitung eignen. Sie funktionieren nach dem SIMD - also Single Instruction Multiple Data - Prinzip, bei dem mit einer einzigen Instruktion mehrere Werte berechnet werden. In diesem Kapitel beschreiben wir, wie wir mit den SSE Instruktionen (Streaming SIMD Extensions), die jeweils vier 32-Bit Floating Point Werte auf einmal berechnen können, eine schnelle Version der in Kapitel 3.4.4 beschriebenen Anwendung des Blackman-Harris-92-Fensters im Frequenzbereich erstellen können.

Wir benötigen drei SSE-Instruktionen. Zunächst die Addition, bei der die beiden Variablen a und b, die jeweils aus vier einzelnen 32-Bit Werten bestehen, komponentenweise addiert werden. Wir schreiben dafür

$$c = a \oplus b = [a^0 + b^0, a^1 + b^1, a^2 + b^2, a^3 + b^3]$$

wobei die Zahlen als Superskript bedeuten, dass ein einzelner Wert einer SSE-Variablen gemeint ist, sodass beispielsweise die SSE-Variable a aus den vier Werten a^0, a^1, a^2, a^3 besteht.

Analog zur Addition gibt es die komponentenweise Multiplikation

$$c = a \odot b = [a^0 \cdot b^0, a^1 \cdot b^1, a^2 \cdot b^2, a^3 \cdot b^3]$$

Schließlich gibt es die Shuffle Operation. Diese hat als Ausgangspunkt ebenfalls zwei SSE-Variablen a und b, und die ersten zwei Werte des Ergebnisses sind Komponenten von a, die anderen beiden Werte sind Komponenten von b. Wir schreiben eine Shuffle Operation beispielsweise als

$$c = S(a^2, a^3, b^0, b^1) = [a^2, a^3, b^0, b^1]$$

Damit die Shuffle Operation tatsächlich als SSE Befehl implementiert werden kann, muss immer gewährleistet sein, dass die Komponenten beider Ausgangsvariablen in genau dieser Reihenfolge auftauchen, im allgemeinen Fall hat die Shuffle Operation die Form

$$c = S(a^i, a^j, b^k, b^l) = [a^i, a^j, b^k, b^l]$$

Zunächst definieren wir die Konstanten $w_0, w_1, ..., w_7$, die die Werte der Transformierten der Fensterfunktion $W(m)$ in der für die spätere Berechnung passenden Reihenfolge enthalten. Sie sind in Tabelle 3.4 aufgelistet. Jeder dieser Werte besteht aus vier 32-Bit Floating Point Komponenten. Die Konstanten werden nur einmal definiert und in der späteren Berechnung verwendet, aber nie verändert.

Konstante	**Wert**
w_0	$[\ 0, \quad 0, \quad W(3), \quad W(3)\ ]$
w_2	$[\ W(2), \quad W(2), \quad W(1), \quad W(1)\ ]$
w_4	$[\ W(0), \quad W(0), \quad W(-1), \quad W(-1)\ ]$
w_6	$[\ W(-2), \quad W(-2), \quad W(-3), \quad W(-3)\ ]$
w_1	$[\ W(3), \quad W(3), \quad W(2), \quad W(2)\ ]$
w_3	$[\ W(1), \quad W(1), \quad W(0), \quad W(0)\ ]$
w_5	$[\ W(-1), \quad W(-1), \quad W(-2), \quad W(-2)\ ]$
w_7	$[\ W(-3), \quad W(-3), \quad 0, \quad 0\ ]$

Tabelle 3.4: Windowabhängige SSE Konstanten

Variable	**Wert**
in_0	$[\ \tilde{E}_{re}(k-4), \quad \tilde{E}_{im}(k-4), \quad \tilde{E}_{re}(k-3), \quad \tilde{E}_{im}(k-3)\ ]$
in_1	$[\ \tilde{E}_{re}(k-2), \quad \tilde{E}_{im}(k-2), \quad \tilde{E}_{re}(k-1), \quad \tilde{E}_{im}(k-1)\ ]$
in_2	$[\ \tilde{E}_{re}(k), \quad \tilde{E}_{im}(k), \quad \tilde{E}_{re}(k+1), \quad \tilde{E}_{im}(k+1)\ ]$
in_3	$[\ \tilde{E}_{re}(k+2), \quad \tilde{E}_{im}(k+2), \quad \tilde{E}_{re}(k+3), \quad \tilde{E}_{im}(k+3)\ ]$
in_4	$[\ \tilde{E}_{re}(k+4), \quad \tilde{E}_{im}(k+4), \quad \tilde{E}_{re}(k+5), \quad \tilde{E}_{im}(k+5)\ ]$

Tabelle 3.5: Eingabewerte als SSE Variablen

Um nun vier Ausgabewerte der Faltung, nämlich

$$out = [E^*_{re}(k), E^*_{im}(k), E^*_{re}(k+1), E^*_{im}(k+1)]$$

in einem Schritt zu berechnen, benötigen wir zunächst die Eingabewerte $in_0, in_1, ..., in_4$, die aus den passenden Werten von $\tilde{E}(k)$ bestehen. Diese sind in Tabelle 3.5 aufgelistet. Da die Werte von $\tilde{E}(k)$ genau in dieser Reihenfolge im Speicher stehen, nämlich abwechselnd jeweils der Realteil und der Imaginärteil, müssen die in-Variablen einfach nur aus dem Speicher gelesen werden.

Die eigentliche Berechnung erfolgt nun in zwei Schritten. Zunächst werden die Variablen f und s (abgekürzt aus „first“ und „second“) aus den in-Variablen und w-Konstanten ermittelt. Danach wird das von uns gewünschte Ergebnis out mittels Shuffle Operationen und Addition zusammengesetzt.

$$\begin{aligned} f &= (in_0 \odot w_0) \oplus (in_1 \odot w_2) \oplus (in_2 \odot w_4) \oplus (in_3 \odot w_6) \\ s &= (in_1 \odot w_1) \oplus (in_2 \odot w_3) \oplus (in_3 \odot w_5) \oplus (in_4 \odot w_7) \\ out &= S(f^0, f^1, s^0, s^1) \oplus S(f^2, f^3, s^2, s^3) \end{aligned}$$

Am Ende dieser Berechnung enthält out die gewünschten Ausgabewerte der Faltung:

$$out = [E^*_{re}(k), E^*_{im}(k), E^*_{re}(k+1), E^*_{im}(k+1)]$$

Um zu sehen, dass dies so ist, zeigen wir wie sich der Wert der out-Komponente an der Position 0 ergibt (also out^0). Dieser hängt nur von f (nicht von s) ab, und zwar von den beiden Bestandteilen f^0 und f^2. Diese können wie folgt berechnet werden:

$$\begin{aligned} f^0 &= in_0^0 \cdot w_0^0 + in_1^0 \cdot w_2^0 + in_2^0 \cdot w_4^0 + in_3^0 \cdot w_6^0 \\ &= \tilde{E}_{re}(k-4) \cdot 0 + \tilde{E}_{re}(k-2) \cdot W(2) + \tilde{E}_{re}(k) \cdot W(0) + \tilde{E}_{re}(k+2) \cdot W(-2) \\ f^2 &= in_0^2 \cdot w_0^2 + in_1^2 \cdot w_2^2 + in_2^2 \cdot w_4^2 + in_3^2 \cdot w_6^2 \\ &= \tilde{E}_{re}(k-3) \cdot W(3) + \tilde{E}_{re}(k-1) \cdot W(1) + \tilde{E}_{re}(k+1) \cdot W(-1) + \tilde{E}_{re}(k+3) \cdot W(-3) \end{aligned}$$

Diese beiden Werte werden bei der Berechnung von out^0 addiert. Man muss die Bestandteile von f^0 und f^2 nur noch geeignet umsortieren, damit die für den Realteil von $E^*(k)$ typische Summe der Faltung zu erkennen ist.

$$out^0 = f^0 + f^2 = \sum_{m=-3}^{3} W(m)\tilde{E}_{re}(k-m) = E^*_{re}(k)$$

Die anderen Komponenten out^1, out^2 und out^3 lassen sich genauso überprüfen.

3.4.6 Performance der einzelnen Schritte

Um zu beurteilen, wie sich die benötigte Rechenzeit für die Synthese des stochastischen Anteils auf die in den vorigen Kapiteln besprochenen Schritte verteilt, haben wir in Tabelle 3.6 und Abbildung 3.8 gemessene Geschwindigkeiten auf einem 64-bit Linux System (AMD Phenom(tm) 9850 Quad-Core Processor) dargestellt.

Dabei sind die Zeiten in Nanosekunden pro Ausgabesample angegeben. Da pro Ausgabesample dank des Overlap-Add-Verfahrens zwei Frames berechnet werden, sind die Zeiten daher mit einem Faktor von zwei skaliert. Es wurde eine Abtastrate von 48000 Hz angenommen - diese beeinflusst die Framegröße in Samples, und damit die gemessene Performance.

Schritt [1] ist die Berechnung von $E(k)$. Dies umfasst die in Kapitel 3.4.1 beschriebene Generierung der Zufallszahlen und die in Kapitel 3.4.2 beschriebene Erzeugung des Spektrums $E(k)$ anhand der Frameparameter $NOISE_b$.

Schritt [2] und [3] sind die unterschiedlichen Alternativen für die Anwendung der Fensterfunktion im Frequenzbereich, einmal die in Kapitel 3.4.4 beschriebene einfache Faltung, und die in

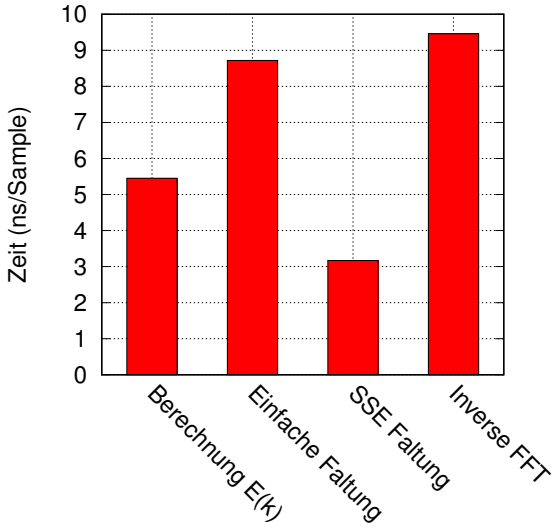


Abbildung 3.8: Zeitbedarf der einzelnen Schritte

Schritt	Zeit (ns/Sample)
[1] Berechnung $E(k)$	5,45
[2] Einfache Faltung	8,72
[3] SSE Faltung	3,17
[4] Inverse FFT	9,46

Tabelle 3.6: Zeitbedarf der einzelnen Schritte

Kapitel 3.4.5 dargestellte Faltung mittels SSE Instruktionen. Es muss nur einer dieser Schritte ausgeführt werden, typischerweise wird die SSE Variante verwendet.

Schritt [4] zeigt schließlich die Kosten der inversen FFT, die ausgeführt wird, nachdem das Spektrum $E^*(k)$ über die Faltung berechnet wurde.

An dieser Stelle haben wir die nötigen Daten um zu bewerten ob wir die Fensterfunktion im Zeitbereich anwenden sollten, wie in Kapitel 3.4.3 beschrieben, oder im Frequenzbereich. Wenn wir die Fensterfunktion im Zeitbereich anwenden, benötigen wir die als Schritt [4] angegebene inverse FFT, sodass wir mindestens die Zeit

$$t_{time} = 5{,}45\,\text{ns/sample} + 9{,}46\,\text{ns/sample} = 14{,}91\,\text{ns/sample}$$

benötigen. Die eigentliche Anwendung des Fensters im Zeitbereich wurde hier nicht gemessen. Diese ist zwar sehr einfach, erhöht den Wert aber noch etwas.

Wenn wir die Fensterfunktion im Frequenzbereich anwenden, wird typischerweise die SSE Faltung verwendet. Die Kosten für die inverse FFT entfallen, da die Synthese des deterministischen und des stochastischen Anteils kombiniert werden können. Als Kosten entstehen in diesem Fall also

$$t_{freq} = 5{,}45\,\text{ns/sample} + 3{,}17\,\text{ns/sample} = 8{,}62\,\text{ns/sample}$$

Man sieht dass die Anwendung der Fensterfunktion im Frequenzbereich mittels der SSE Faltung deutliche Effizienzvorteile mit sich bringt. Wenn keine SSE Instruktionen vorhanden sind, ist dieser Vorteil sehr viel geringer.

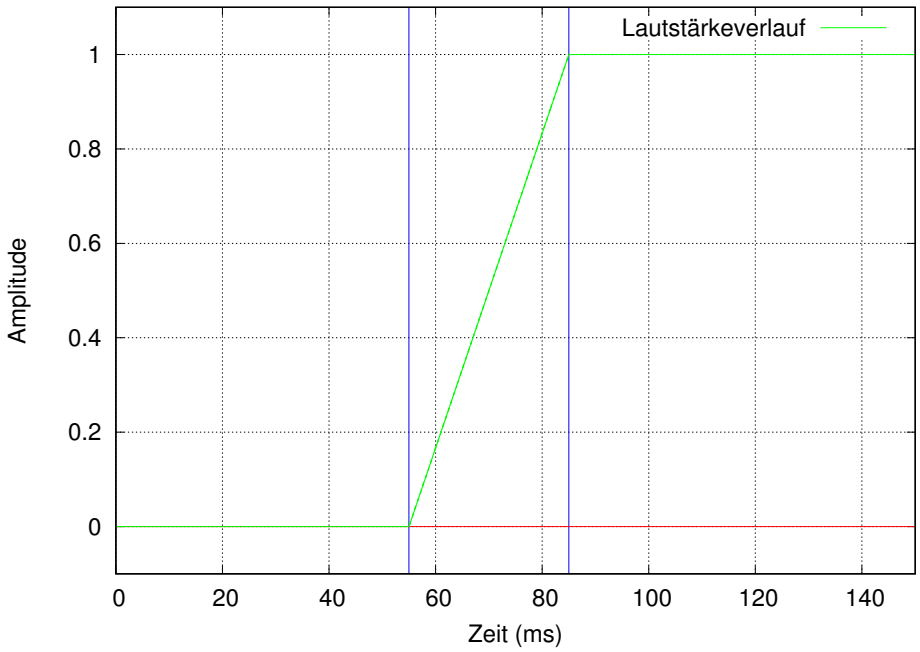


Abbildung 3.9: Attack-Envelope aus der Analyse

3.5 Synthese des Envelopes

Als letzter Schritt der Synthese ist es nötig, den bei der Analyse bestimmten Attack-Envelope mit der Summe des deterministischen und stochastischen Anteils zu kombinieren.

Wie in Kapitel 2.4.3 beschrieben wurde, wurde das Signal $x(t)$ vorne mit Nullen aufgefüllt, um bessere Ergebnisse für den Attack-Envelope zu erhalten. Diese werden bei der Synthese wieder abgeschnitten, wobei berücksichtigt wird, dass die Analyseabtastrate nicht unbedingt gleich der Syntheseabtastrate ist (die Anzahl der abzuschneidenden Samples wird also entsprechend skaliert).

Die Envelopefunktion - dargestellt in Abbildung 3.9 - hatten wir zuvor definiert als:

$$env(t,\alpha,\beta) = \begin{cases} 0 & \text{für} \quad ms(t) < \alpha \\ \frac{ms(t)-\alpha}{\beta-\alpha} & \text{für } \alpha \leq \quad ms(t) < \beta \\ 1 & \text{für } \beta \leq \quad ms(t) \end{cases}$$

$$ms(t) = 1000\frac{t}{F_s}$$

Diese wird im Zeitbereich berechnet, und auf das Syntheseergebnis aufmultipliziert. Da die Werte für den Start und für das Ende des Attacks (in der Formel α und β) in Millisekunden gespeichert wurden, können sie mittels der Syntheseabtastrate in Samples umgerechnet werden. Da der Envelope immer bei der Amplitude eins endet und dann konstant bleibt, kann die Anwendung des Envelopes mittels Multiplikation auf den Anfang der Aufnahme begrenzt werden, ab dem Ende des Attacks ist keine Envelopesynthese mehr notwendig.

Als Bemerkung zur Performance dieses Schritts sei angemerkt, dass in unserer Implementation unnötigerweise am Anfang des Samples bei der Synthese des deterministischen und stochastischen Anteils Werte berechnet werden, die dann ohnehin bei der Envelopesynthese entweder abgeschnitten oder mit Null multipliziert werden. Man könnte am Anfang jeder Note die Berechnung also

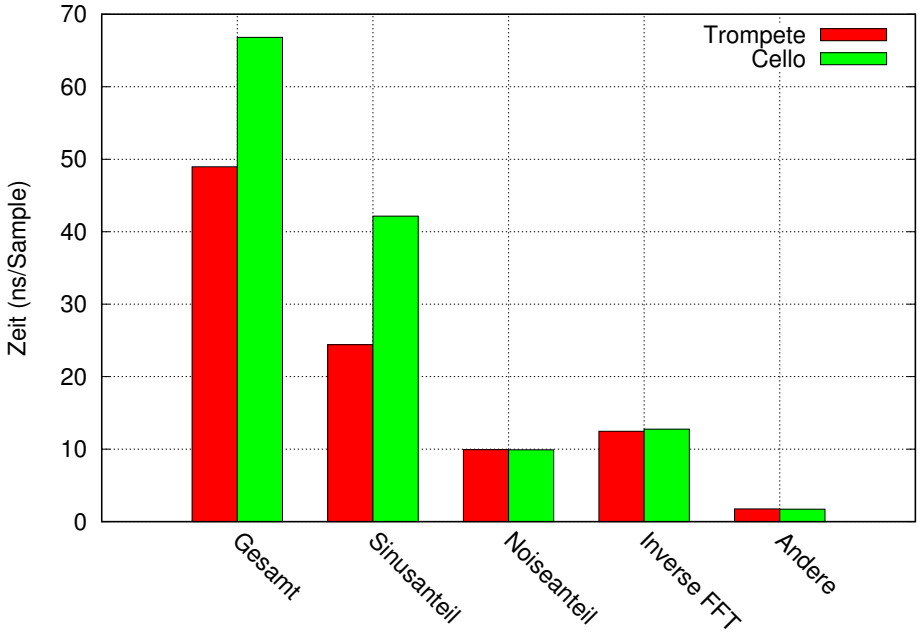


Abbildung 3.10: Zeitbedarf der Synthese

etwas optimieren.

3.6 Gesamtperformance der Synthese

Um abschließend die Performance der Synthese zu beurteilen, berechnen wir bei einer Abtastrate von 48 000 Hz die Samples für die Ausgabe von einer Sekunde und ermitteln dabei wieviel Rechenzeit auf welche Schritte der Gesamtsynthese entfallen. Normiert werden die Ergebnisse so, dass jeweils die Dauer in Nanosekunden pro Ausgabesample angegeben werden. Die Messungen wurden auf einem 64-bit Linux System (AMD Phenom(tm) 9850 Quad-Core Processor) durchgeführt.

Der Aufwand der Berechnung hängt vom Instrument und der gewählten Note ab, da jeweils unterschiedlich viele Teiltöne vorkommen. Daher können wir, anders als bei der Synthese des stochastischen Anteils, kein allgemeines Ergebnis angeben. Um dennoch eine ungefähre Einschätzung geben zu können, messen wir zumindest zwei verschiedene Fälle, die Synthese eines Trompetentons und eines tiefen Cellotons.

Instrument	**Tonhöhe (Hz)**	**Durchschnitt Teiltöne/Frame**
Trompete	440,0	107,3
Cello	65,4	196,3

Tabelle 3.7: Gemessene Instrumente

Man sieht bereits in Tabelle 3.7, dass bei dem tiefen Celloton im Durchschnitt für die Synthese des deterministischen Anteils wesentlich mehr Teiltöne pro Frame berechnet werden müssen, als für den Trompetenton. Die genauen Ergebnisse unseres Benchmarks finden sich in Tabelle 3.8 und in Abbildung 3.10.

Die Gesamtkosten der Synthese belaufen sich für die Trompete auf

$$t_{trumpet} = 49{,}0\,\text{ns/sample}$$

Schritt	Trompete (ns/Sample)	Cello (ns/Sample)
Gesamt	49,0	66,8
Sinusanteil	24,4	42,2
Noiseanteil	9,9	9,9
Inverse FFT	12,5	12,7
Andere	1,8	1,7

Tabelle 3.8: Zeitbedarf der Syntheseschritte

und für das Cello auf

$$t_{cello} = 66{,}8\,\text{ns/sample}$$

Diese sind dann jeweils weiter unterteilt, in die Kosten für die Berechnung des Sinusanteils, die Kosten für die Berechnung des Noiseanteils, die inverse FFT und Kosten, die sich nicht diesen Blöcken zuordnen ließen. Erwartungsgemäß unterscheiden sich die Ergebnisse der Instrumente nur bei der Synthese des Sinusanteils. Da das Cello durchschnittlich wesentlich mehr Teiltöne pro Frame enthält, ist dieser Schritt hier deutlich teurer. Es wird, wie in Kapitel 3.4.6 besprochen, nur eine gemeinsame inverse FFT für beide, Sinus- und Noiseanteil, durchgeführt.

Teilt man die Kosten des Sinusanteils durch die durchschnittliche Anzahl der Teiltöne pro Frame, erhält man in etwa den gleichen Wert:

$$s_{trumpet} = \frac{24{,}4\,\text{ns/sample}}{107{,}3} = 0{,}227\,\text{ns/sample}$$
$$s_{cello} = \frac{42{,}2\,\text{ns/sample}}{196{,}3} = 0{,}214\,\text{ns/sample}$$

Da die Synthese des Sinusanteils eines Frames im wesentlichen aus der in Kapitel 3.3.4 beschriebenen Bestimmung der Phase sowie der in Kapitel 3.3.2 erläuterten Berechnung von jeweils einem Peak im Spektrum pro Teilton besteht, ist dies auch zu erwarten. Diese Komponente ist also von der Komplexität $O(P)$ wobei P die Anzahl der Teiltöne des aktuellen Syntheseframes ist. Alle anderen Kosten sind für jeden Syntheseframe konstant.

Um die Ergebnisse der Performancetests noch in eine intuitivere Form zu bringen, betrachten wir die theoretisch maximale Polyphonie. In einer fertigen Anwendung muss in der Regel nicht nur die Synthese einer einzigen Stimme stattfinden. Es soll normalerweise die Synthese mehrerer Stimmen gleichzeitig erfolgen. Daher stellt sich die Frage, wie viele Stimmen die Software in Echtzeit produzieren kann. Dies ist die maximal erreichbare Polyphonie. Diese hängt wie oben von der Anzahl der Teiltöne pro Stimme ab.

Aus den Benchmarkergebnissen können wir die theoretisch maximale Polyphonie schätzen. Wir wissen für die Trompete und für das Cello die Zeit, die es kostet, ein Sample zu synthetisieren. Für eine Stimme benötigen wir 48000 Samples pro Sekunde. Also können wir ausrechnen, wieviele Stimmen theoretisch pro Sekunde berecht werden können.

$$p_{trumpet} = \frac{1}{48000 \cdot 49 \cdot 10^{-9}} = 425{,}2$$
$$p_{cello} = \frac{1}{48000 \cdot 66{,}8 \cdot 10^{-9}} = 312{,}3$$

Bei 425,2 Trompetenstimmen bzw. bei 312,3 Cellostimmen ist die Implementation als effizient genug für anspruchsvolle Echtzeitanwendungen anzusehen - allerdings sind dies theoretische Werte, die in der Praxis wahrscheinlich nicht erreicht werden können. Vor allem der Zugriff auf die Analysedaten aus dem Cache verlangsamt sich wenn mehrere unterschiedliche Noten parallel gespielt

werden. In dem vorliegenden Benchmark wird außerdem der Aufwand für das Morphing nicht berücksichtigt. Die Performance der vorliegenden Implementation der Synthese stellt aber angesichts dieser Werte kein Problem für das Gesamtsystem dar.

4 Morphing zwischen Klängen

4.1 Der Morph-Plan

4.1.1 Aufbau des Plans

Um durch Morphing neue Klänge aus bestehenden Instrumenten kombinieren zu können, haben wir ein modulares Design entwickelt, welches dem Musiker erlaubt flexibel zu definieren, wie die Klänge kombiniert werden sollen. Dabei gibt der Morph-Plan die Struktur vor, anhand der die Berechnung erfolgt. Er besteht aus miteinander verbundenen Operatoren.

Ein Beispiel für einen Morph-Plan findet sich in Abbildung 4.1. Hier ist zum einen die Darstellung an der GUI zu sehen, zum anderen den Graph, der durch diesen Morph-Plan definiert wird, und die Grundlage der eigentlichen Berechnung ist. Insgesamt werden vier Operatoren verwendet.

Die beiden Operatoren `Source #1` und `Source #2` stellen die Ausgangsdaten bereit. An der grafischen Oberfläche kann hier jeweils ein Instrument selektiert werden. Der `Linear #1` Operator wird von den Source-Operatoren gespeist, und kombiniert beide mittels Morphing. Der `Output #1` Operator führt schließlich die Umwandlung des vom linearen Morphing erzeugten Spektrums in den Zeitbereich durch, es findet dort also die in Kapitel 3 beschriebene Synthese statt.

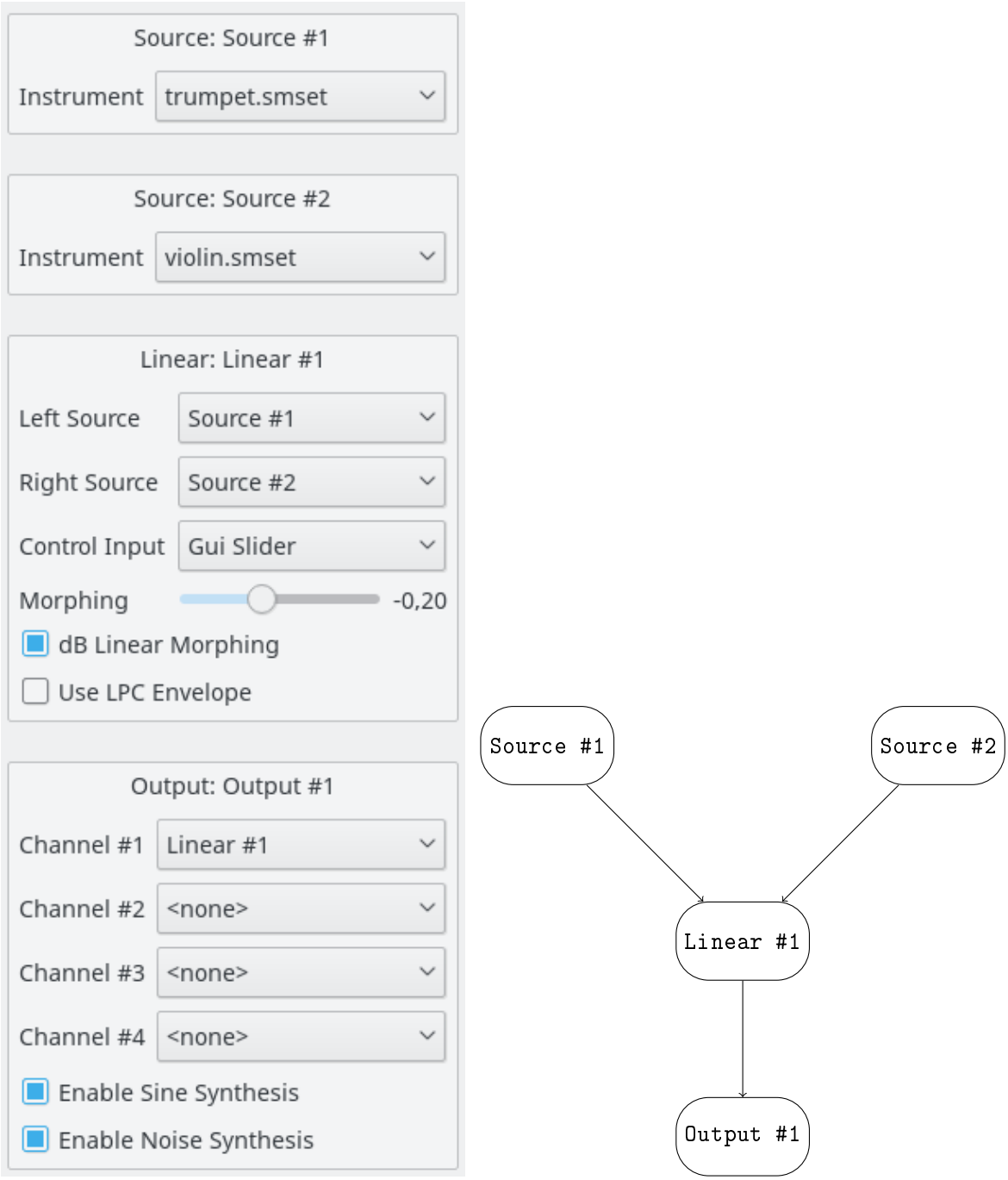
Durch den Morph-Plan kann der Musiker festlegen, wie die einzelnen Instrumente kombiniert werden. Dabei wird eine beliebige Schachtelung der Operatoren unterstützt, es kann beispielsweise das Ergebnis eines linearen Morph Operators wiederum als Eingabe für einen weiteren linearen Morph Operator verwendet werden. Zusätzlich erlaubt das Design durch die modulare Struktur das Hinzufügen von neuen Operatoren, die andere Möglichkeiten eröffnen.

In den nächsten Abschnitten stellen wir die verfügbaren Operatoren mit ihren Eigenschaften vor, wobei wir zunächst nur die Oberfläche beschreiben. Später in Kapitel 4.2 gehen wir auf die Implementation der Morphing Operatoren ein. Die `Source` und `Output` Operatoren werden nicht weiter beschrieben, da sie keine eigene Berechnung bereitstellen. Sie fungieren nur als Eingabe und Ausgabe.

4.1.2 Lineares Morphing

Der `Linear`-Operator stellt die einfachste Form des Morphings bereit. Es werden zwei Klänge als Eingabe angegeben, die kombiniert werden. Dabei wird ein `Morphing`-Wert von -1 bis 1 benötigt, der angibt wie stark die Klänge die Ausgabe beeinflussen. Wenn der Wert -1 ist, ist nur die angegebene „linke Quelle“ zu hören. Ist der Wert 1, ist nur die angegebene „rechte Quelle“ zu hören. Ein Wert von 0 entspricht einer 50%/50% Mischung, und so weiter.

Im einfachsten Fall wird dieser `Morphing`-Wert direkt an der GUI festgelegt und kann dort durch einen Slider verändert werden (auch während die Synthese läuft). Eine weitere Möglichkeit zur Definition des `Morphing`-Wertes ist die Steuerung von außen. Damit ist gemeint, dass ein Sequencer, der ggf. auch die zu spielenden Noten festlegt, den `Morphing` Wert beim Abspielen steuert, wodurch in einem Musikstück festgelegt werden kann, wie sich die Klangzusammensetzung über die Zeit verändert. Ein LFO kann auch zur Steuerung des Morphings benutzt werden, dies wird in Kapitel 4.1.4 beschrieben. Dabei steht LFO für „Low Frequency Oscillator“, also übersetzt etwa niederfrequenter Oszillator. Damit kann eine periodische langsame Veränderung der Klangfarbe erzielt werden.



(a) An der GUI (b) Als Graph

Abbildung 4.1: Einfacher Morph-Plan

Der lineare Morph Operator hat zudem die beiden Eigenschaften `db Linear Morphing` und `Use LPC Envelope`, die verschiedene Morphingalgorithmen aktivieren.

4.1.3 Gitterbasiertes Morphing

In Abbildung 4.2 ist der `Grid` Morph Operator zu sehen. Dabei werden Instrumente auf einem $W \times H$ Gitter angeordnet, also jeweils W Instrumente nebeneinander und H Instrumente übereinander. Als Beispiel verwenden wir hier ein 2×2 Gitter. Im Prinzip lassen sich die Klänge (wie beim linearen Morphing) außerhalb definieren, etwa durch `Source` Operatoren. Im vorliegenden Fall würden aber vier weitere Operatoren außerhalb des `Grid` Operators hinzukommen. Um die Bedienung zu

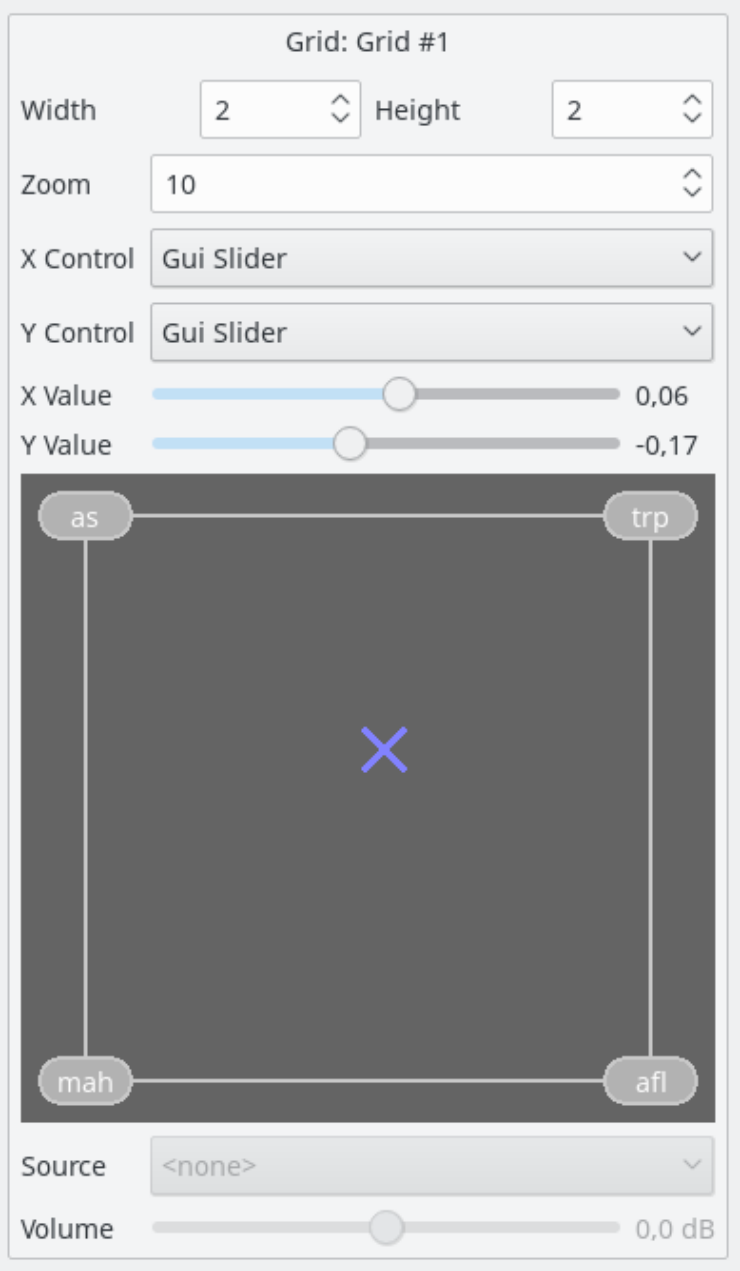


Abbildung 4.2: Grid Morph Operator

vereinfachen lassen sich die Knoten des Gitters direkt mit Instrumenten belegen, wie es hier der Fall ist.

Das blaue Kreuz definiert an der GUI die beiden Steuerungswerte `X Control` und `Y Control`. Liegt es auf einer der Ecken, ist nur der dort befindliche Klang zu hören. Befindet es sich auf den Außenkanten, so wird ein lineares Morphing der beiden Instrumente auf den Kanten vorgenommen. Liegt es schließlich in der Mitte, werden alle vier Instrumente kombiniert, wobei die Stärke des Einflusses durch die `X/Y` Position vorgegeben wird.

Man kann das gitterbasiertes Morphing auch mit anderen Größen verwenden, ein 3×1 Gitter beschreibt zum Beispiel drei nebeneinander liegende Instrumente, wobei immer lineares Morphing von zwei Instrumenten erfolgt. Man könnte damit eine Veränderung von Trompete über Geige zu Oboe definieren, wobei immer nur zwei Instrumente zur Zeit zu hören sind.

Ein 3×3 Gitter enthält insgesamt 9 Klänge, in diesem Fall befindet sich die `X/Y` Position jedoch immer in einem der vier Teilquadrate, sodass nur die Instrumente an den vier Ecken des Quadrats,

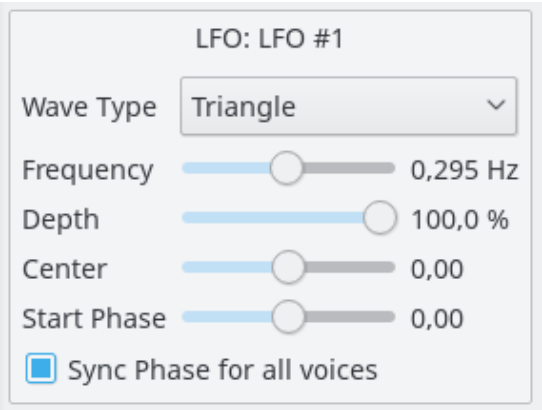


Abbildung 4.3: LFO Operator

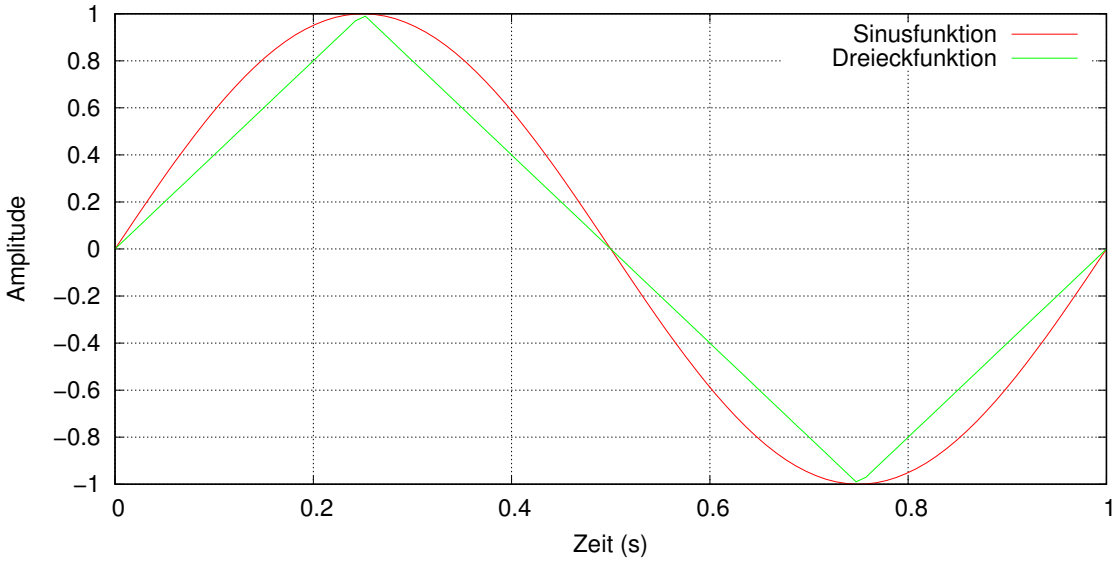


Abbildung 4.4: LFO Steuerfunktion

in dem die `X/Y` Position ist, für das Morphing relevant sind.

Schließlich kann die Steuerung des Morphings wieder direkt an der GUI erfolgen, wobei dann das Kreuz auf der Fläche bewegt werden kann, um das Morphing zu steuern. Es ist analog dazu möglich, das Morphing von außen zu steuern, wobei dann ein Sequencer zwei Parameter `X` und `Y` steuert. Als Alternative kann man auch für beide Parameter jeweils einen LFO verwenden. Dieser wird im nächsten Abschnitt vorgestellt.

4.1.4 Der LFO Operator

In Abbildung 4.3 ist der `LFO` Operator zu sehen („Low Frequency Oscillator“). Die Aufgabe dieses Operators ist es, Steuerungseingaben zu berechnen, die für das Morphing verwendet werden. Dabei wird der zu steuernde Parameter automatisch durch eine sich langsam ändernde Funktion festgelegt. Es ist also so, als ob der Benutzer den Parameter durch eine langsame Bewegung immer von -1 nach 1 und wieder nach -1 und so weiter steuern würde. Die berechnete Steuerfunktion ist, für eine Beispielfrequenz von 1 Hz, in Abbildung 4.4 zu sehen.

Dabei können sowohl der `Morphing` Parameter des linearen Morphings als auch die `X` und `Y` Parameter des gitterbasierten Morphings gesteuert werden. Die Ausgabe des Operators ist je nach `Wave Type` Parameter entweder eine Sinus- oder Dreieckfunktion mit der Frequenz des `Frequency`

Parameters. Die beiden Parameter `Center` und `Depth` verschieben den Mittelwert und bestimmen die Amplitude.

Die Phase LFO startet bei `Start Phase`. Es ist möglich, den LFO so einzustellen, dass die Phase bei mehrstimmiger Synthese für alle Stimmen gleich ist.

4.2 Implementation des Linearen Morphings

4.2.1 Eingabeparameter

Für die Implementation des linearen Morphings muss aus dem Klang von zwei Instrumenten ein resultierender Klang berechnet werden. Dabei werden für jeden Ausgabeframe die Parameter von zwei Eingabeframes, jeweils einer von jedem Ausgangsinstrument, geeignet kombiniert. Wir nennen die Eingabe im folgenden den „linken“ Frame (mit dem Superskript l und P Sinusanteilen) und den „rechten“ Frame (mit dem Superskript r und Q Sinusanteilen).

Die Eingabeparameter sind in Tabelle 4.1 zu sehen. Die Phasen werden bei der Synthese ergänzt, müssen also nicht gesondert behandelt werden.

Die Steuerung des Morphings erfolgt durch den `Morphing` Parameter aus Kapitel 4.1.2. Dieser liegt im Bereich [−1,1], und gibt an wie stark der linke bzw. rechte Frame im Ausgabesignal sein soll. Um später die Formeln klarer beschreiben zu können, rechnen wir diesen in den Parameter $\lambda \in [0,1]$ um:

$$\lambda = \frac{\texttt{Morphing} + 1}{2}$$

Hierbei bedeutet ein Wert von $\lambda = 0$, dass nur das linke Instrument zu hören ist, $\lambda = 1$, dass nur das rechte Instrument zu hören ist, $\lambda = 0{,}5$ entspricht einer 50%/50% Mischung, und so weiter.

Parameter	**linker Frame**	**rechter Frame**
Frequenzen	$F^l_1, \ldots, F^l_P$	$F^r_1, \ldots, F^r_Q$
Amplituden	$A^l_1, \ldots, A^l_P$	$A^r_1, \ldots, A^r_Q$
Noiseanteil	$NOISE^l_0, \ldots, NOISE^l_{31}$	$NOISE^r_0, \ldots, NOISE^r_{31}$

Tabelle 4.1: Eingabeparameter für das lineare Morphing

4.2.2 Berechnung des stochastische Anteils

Da die Berechnung des stochastischen Anteils besonders einfach ist, geben wir diese zu erst an. Die 32 Noisebänder der Ausgabe erhält man als

$$NOISE_b = (1 - \lambda) \cdot NOISE^l_b + \lambda \cdot NOISE^r_b, \text{ für } b \in [0,31]$$

Ist $\lambda = 0$, wird nur der Noiseanteil des linken Eingabeframes abgespielt, ist $\lambda = 1$, wird nur der Noiseanteil des rechten Eingabeframes abgespielt, ist λ dazwischen wird die Amplitude der korrespondierenden Noisebänder linear interpoliert.

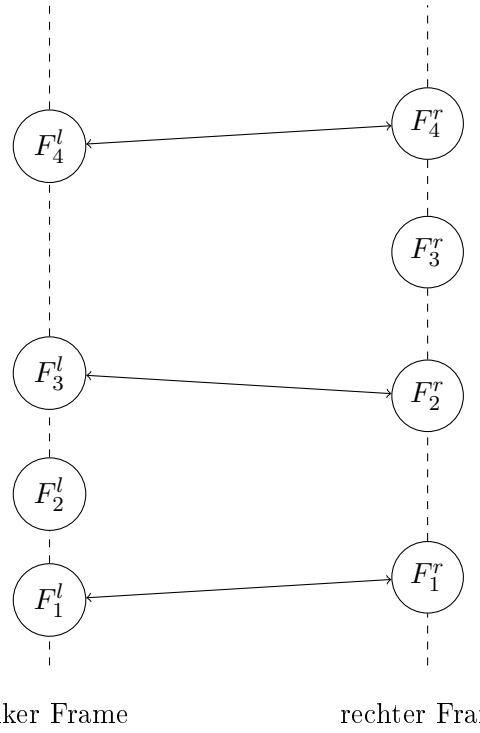


Abbildung 4.5: Zuordnung der Teiltöne des linken und rechten Frames

4.2.3 Zuordnung der Teiltöne

In Abbildung 4.5 ist ein Beispiel für die Position der Teiltöne des linken und rechten Frames dargestellt. Der Morphing Algorithmus erstellt zunächst eine Zuordnung der Frequenzen des linken Frames und der des rechten Frames. Bei Übereinstimmungen wird jeder Eintrag höchstens einmal zugeordnet, kein Teilton wird mehrmals verwendet.

Dabei treten zwei Fälle auf. Entweder gibt es eine Übereinstimmung, es passt also eine Frequenz aus dem linken Frame gut zu einer Frequenz aus dem rechten Frame. Dies ist hier bei F_1^l und F_1^r, F_3^l und F_2^r sowie F_4^l und F_4^r der Fall. Dann wird aus beiden Parametern dieser Teiltöne ein neuer Teilton für den Ausgabeframe erzeugt. Oder es bleiben Teiltöne des linken oder rechten Frames übrig, hier F_2^l oder F_3^r, dann wird für diese jeweils ein eigener Eintrag im Ausgabeframe berechnet. Im Beispiel hätten dieser also fünf Einträge.

Der erste Schritt des Morphings ist die Bestimmung der Teiltöne beider Frames, die einander zugeordnet werden können. Dabei berücksichtigen wir nicht nur die Frequenzen der Frames, sondern auch deren Amplituden. Laute Anteile der Eingangssignale sind normalerweise auch im Ausgangssignal laut, und besser wahrzunehmen als leise Anteile. Daher ist die Zuordnung der lauten Teiltöne für die Qualität entscheidend.

Um die Daten nach Amplitude sortiert behandeln zu können, füllen wir ein Array mit jeweils drei Parametern pro Teilton. Zunächst die Amplitude, dann die Nummer des Teiltons und schließlich ein Flag (`LEFT` oder `RIGHT`), ob er zum linken oder rechten Frame gehört. Insgesamt enthält das Array also $P + Q$ Einträge, und zwar

$$[(A_1^l, 1, \texttt{LEFT}),\ (A_2^l, 2, \texttt{LEFT}),\ \ldots, (A_p^l, p, \texttt{LEFT}),\ \ldots, (A_P^l, P, \texttt{LEFT}),$$
$$(A_1^r, 1, \texttt{RIGHT}), (A_2^r, 2, \texttt{RIGHT}), \ldots, (A_q^r, q, \texttt{RIGHT}), \ldots, (A_Q^r, Q, \texttt{RIGHT})]$$

Dieses wird nach Amplituden absteigend sortiert, und dann Eintrag für Eintrag abgearbeitet. Für die Beschreibung nehmen wir der Einfachheit halber an, dass der aktuelle Eintrag aus dem linken Frame stammte (das Flag also `LEFT` ist) - Teiltöne aus dem rechten Frame werden analog dazu

behandelt.

Der Teilton hat also die Position p, die Amplitude A_p^l und die Frequenz F_p^l. Wurde er zuvor schon zugeordnet, sind wir mit diesem Eintrag fertig. Andernfalls suchen wir einen Teilton q aus dem rechten Frame, der dazu passt. Dies bedeutet, dass die Frequenz F_q^r in der Nähe der Frequenz F_p^l liegen soll, genauer gesagt basiert unsere Entscheidung auf einem Kriterium, welches die Grundfrequenz G berücksichtigt:

$$\delta = |F_q^r - F_p^l| \leq \frac{G}{2}$$

Weiterhin soll der Eintrag noch nicht schon zugeordnet sein. Gibt es mehrere Kandidaten nehmen wir den Kandidaten q für den δ minimal ist. Die Wahl des Abstandskriteriums folgt der Annahme, dass die Teiltöne normalerweise im Abstand der Grundfrequenz auftreten, womit bei der Zuordnung ein Abstand von maximal $^{G}/_{2}$ vorhanden sein darf, wenn man den Kandidaten nicht fälschlicherweise dem vorherigen oder nächsten Teilton zuordnen will.

4.2.4 Bestimmung der Amplitude

Wenn eine Zuordnung des p-ten Teiltons aus dem linken Eingabeframes und des q-ten Teiltons aus dem rechten Eingabeframes erfolgt ist, wird aus den beiden Parametersätzen ein Teilton für die Ausgabe bestimmt. Dabei wird die Ausgabeamplitude - analog zu den Noiseanteilen - berechnet als:

$$A = (1 - \lambda) \cdot A_p^l + \lambda \cdot A_q^r$$

Es gibt zwei weitere Fälle zu berücksichtigen, einerseits berechnen wir die Amplitude falls der p-te Teilton nicht zugeordnet werden konnte als

$$A = (1 - \lambda) \cdot A_p^l$$

und für den Fall, dass der q-te Teilton nicht zugeordnet werden konnte als

$$A = \lambda \cdot A_q^r$$

Dies entspricht dem Einsetzen von 0 für die jeweils andere Amplitude.

4.2.5 Bestimmung der Frequenz

Liegt eine Zuordnung vom p-ten Teilton des linken Frames und dem q-ten Teilton des rechten Frames vor, erscheint es zunächst logisch, die Ausgabefrequenz auf die gleiche Weise zu berechnen wie die Ausgabeamplitude

Wenn die Amplituden A_p^l und A_q^r etwa gleich groß sind, würde dies auch funktionieren. In den Analysedaten kommen aber immer wieder sehr schwache Teiltöne vor, die sich manchmal nicht von Frame zu Frame fortsetzen, sondern erscheinen und verschwinden. Diese haben auch nicht immer die typischen Frequenzen von G, $2G$, ... sondern liegen oft irgendwo zwischen zwei ganzzahligen Vielfachen der Grundfrequenz.

Bei der Resynthese ohne Morphing sind diese Teiltöne nicht zu hören, sodass normalerweise kein Qualitätsverlust entsteht. Würde man die Ausgabefrequenz F nun linear aus einem gut zu hörenden Teilton und einem ggf. instabilen, nicht zu hörenden Teilton bestimmen, würde die bei der Analyse sehr gut geschätzte Frequenz des lauten Teiltons und die möglicherweise durch Analysefehler entstandene Frequenz des leisen Teiltons linear kombiniert. Dadurch könnte der resultierende

Teilton eine stark abweichende Frequenz F erhalten. Und da die Amplitude des lauten Teiltons bei der Bestimmung der Ausgabeamplitude A verwendet wird, würde der normalerweise unhörbare Analysefehler durch das Morphing gut hörbar, in Form einer falschen Frequenz.

Um diesen Effekt zu verhindern, beziehen wir beide Amplituden in die Frequenzberechnung mit ein. Sei der Teilton des linken Frames lauter als der Teilton des rechten Frames (im anderen Fall erfolgt die Berechnung analog), also $A_l^p \geq A_q^r$. Dann verwenden wir als Frequenz

$$F = F_p^l + m\lambda(F_q^r - F_p^l)$$

wobei m ein von beiden Amplituden abhängiger Korrekturfaktor ist:

$$m = \frac{A_q^r}{A_p^l}$$

Im einen Grenzfall ist sind die Amplituden gleich, dann ergibt sich für $m = 1$:

$$F = F_p^l + 1\lambda(F_q^r - F_p^l) = (1-\lambda)F_p^l + \lambda F_q^r$$

In dem anderen Grenzfall ist eine Amplitude sehr viel lauter als die andere, dann ergibt sich für $m \approx 0$:

$$F \approx F_p^l + 0\lambda(F_q^r - F_p^l) = F_p^l$$

Dieser Fall tritt auf, wenn ein lauter Teilton, dessen Frequenz bei der Analyse gut geschätzt werden konnte, mit einem unstabilen leisen Teilton kombiniert wird. Durch den Korrekturfaktor m wird die Ausgabefrequenz dann (fast) nur vom stabilen lauten Teilton bestimmt. Bei Tests hat sich diese Art F zu berechnen als robust und zuverlässig erwiesen.

Es bleibt noch die Berechnung der Frequenz F, wenn ein Teilton nicht zugeordnet werden konnte. Konnte der p-te Teilton des linken Frames nicht zugeordnet werden, verwenden wir

$$F = F_p^l$$

Konnte der q-te Teilton des rechten Frames nicht zugeordnet werden, verwenden wir

$$F = F_q^r$$

4.2.6 Behandlung unterschiedlicher Grundfrequenzen

In der bisherigen Beschreibung wird angenommen, dass die Grundfrequenz des linken Frames, des rechten Frames und des Ausgabeframes gleich sind. Dann funktioniert die Zuordnung der Teiltöne so wie erwartet, da in beiden Eingabeframes jeweils die typischen Vielfachen der Grundfrequenz G, $2G$, ... vorkommen. Diese sich entsprechenden Frequenzen werden einander zugeordnet.

Es kann aber vorkommen, dass die Eingabeframes voneinander abweichende Grundfrequenzen haben. Im allgemeinen Fall ist die gewünschte Grundfrequenz der Ausgabe des Morphings G, die Grundfrequenz des linken Frames ist G^l und die Grundfrequenz des rechten Frames ist G^r. Damit das Morphing dann richtig funktioniert, skalieren wir in diesem Fall alle Frequenzen des linken Frames mit dem Faktor G/G^l und die Frequenzen des rechten Frames mit dem Faktor G/G^r. So passen die Teiltöne für die Zuordnung wie gewünscht zusammen, und die Frequenzen der Ausgabe haben die erwartete Grundfrequenz G.

4.2.7 Bestimmung der Amplitude aus dB-Werten

Eine alternative Implementation zu der oben angegebenen Berechnung der Amplituden basiert auf der Annahme, dass die Umrechnung der Amplituden in dB eine bessere Grundlage zur Bestimmung

der Lautstärken ist, denn ein gleichgroß empfundener Lautstärkeunterschied entspricht auf der dB Skala immer dem gleichen Abstand.

Es ist z.B. eine Amplitude von 1 doppelt so laut wie eine Amplitude von 0,5, diese wiederum doppelt so laut wie eine Amplitude von 0,25. Auf dieser Skala sind die Werte unterschiedlich weit entfernt (obwohl der wahrgenommene Lautstärkeunterschied jeweils gleichgroß ist). Auf der dB-Skala entsprechen diesen Amplituden 0 dB, etwa -6 dB und etwa -12 dB. Man sieht, dass auf dieser Skala die Amplituden den gleichen Abstand haben (etwa 6 dB). Daher macht es Sinn, die Amplituden in dB umzurechnen, die resultierende Amplitude als dB Wert zu berechnen, und wieder zurückzukonvertieren.

Wir verwenden für die Konvertierung die Formeln:

$$db(x) = \max(20 \log_{10}(x), -96)$$
$$db2factor(x) = 10^{x/20}$$

Hierbei wird die Umrechnung in dB nach unten begrenzt, sodass die kleinstmögliche Ergebnis -96 ist. Sonst könnten durch den Logarithmus Werte bis $-\infty$ entstehen, weil x beliebig nah an 0 kommen kann.

Sei nun also der p-te Teilton aus dem linken Eingabeframe dem q-ten Teilton aus dem rechten Eingabeframe zugeordnet, dann ergibt sich die neue Berechnung

$$A = db2factor((1-\lambda) \cdot db(A_p^l) + \lambda \cdot db(A_q^r))$$

Falls der p-te Teilton nicht zugeordnet werden konnte, berechnen wir die Amplitude als

$$A = db2factor((1-\lambda) \cdot db(A_p^l) + \lambda \cdot (-96))$$

und für den Fall, dass der q-te Teilton nicht zugeordnet werden konnte als als

$$A = db2factor((1-\lambda) \cdot (-96) + \lambda \cdot db(A_q^r))$$

Man sieht hier, dass wir die dB-Skala nach unten begrenzt haben, sodass das Einsetzen von 0 für die jeweils andere Amplituden eine Konstante von -96 für die Berechnung des resultierenden dB-Wertes ergibt.

4.2.8 Das Problem des Attack-Envelopes

Bei der Analyse hatten wir für jede Aufnahme einen optimalen Attack-Envelope bestimmt. Im Morphing liegen uns also zwei verschiedene Envelopes für die beiden Eingangsinstrumente vor. In unserer aktuellen Implementation verwenden wir diese Envelopes nicht für das Morphing. Es produziert trotzdem in den Tests gute Ergebnisse, da die Instrumente die sich gut für das Morphing eignen, also etwa Trompete, Geige und Altsaxophon ohnehin relativ langsame Einschwingphasen haben. Der Attack-Envelope erbringt hier keine relevante Verbesserung.

Das Klavier, welches signifikant besser klingt, wenn es mit Attack-Envelope synthetisiert wird, eignet sich nicht gut für unser Morphing. Wir brauchen dafür ein Instrument welches nach der Einschwingphase einen Zustand erreicht ab dem die Amplitude etwa konstant bleibt (und der geloopt werden kann). Dies ist bei den obengenannten Instrumenten so, das Klavier klingt jedoch langsam aus.

Wir werden trotzdem hier zwei Ideen zum Envelopemorphing vorstellen. Ohne sie jedoch implementiert und auf einer großen Menge von Fällen getestet zu haben, können wir keine definitive Bewertung abgeben.

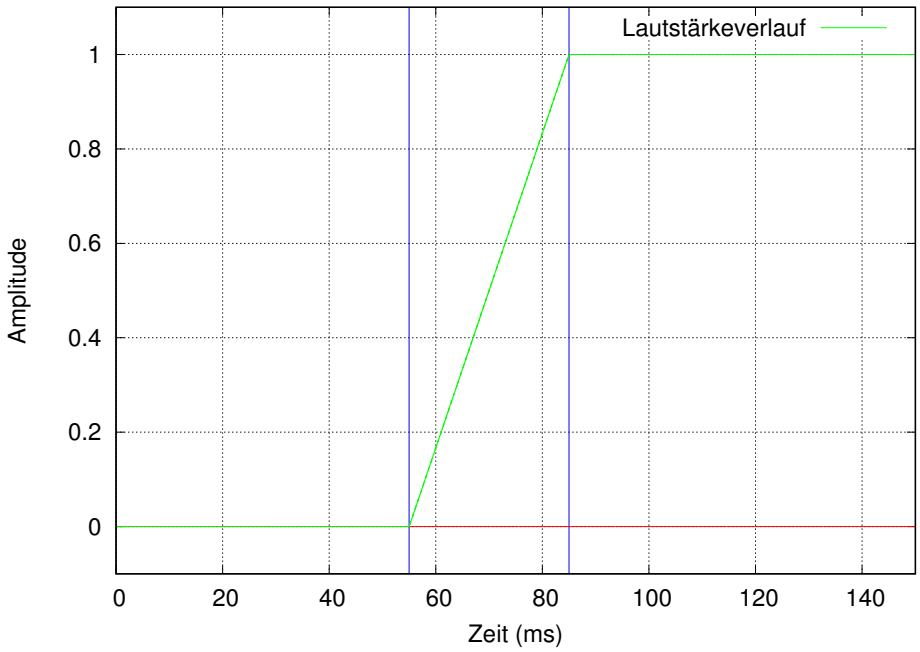


Abbildung 4.6: Attack-Envelope aus der Analyse

Die Ausgangsparameter für die Envelopes der beiden Instrumente sind in Tabelle 4.2 gelistet. Dabei ist die Envelopefunktion aus Kapitel 2.4 wie folgt definiert:

$$env(t, \alpha, \beta) = \begin{cases} 0 & \text{für} \quad ms(t) < \alpha \\ \frac{ms(t)-\alpha}{\beta-\alpha} & \text{für } \alpha \leq ms(t) < \beta \\ 1 & \text{für } \beta \leq ms(t) \end{cases}$$

$$ms(t) = 1000\frac{t}{F_s}$$

Parameter	**linkes Instrument**	**rechtes Instrument**
Attack-Start	α^l	α^r
Attack-Ende	β^l	β^r

Tabelle 4.2: Envelopeparameter für das lineare Morphing

Gesucht ist eine durch Morphing ermittelte Funktion $env^*(t)$, die die beiden Eingabeenvelopes $env(t, \alpha^l, \beta^l)$ und $env(t, \alpha^r, \beta^r)$ geeignet kombiniert.

Die eine Idee wäre, den Envelope für das linke Instrument zu berechnen, den Envelope für das rechte Instrument zu berechnen, und die gewünschte Envelopefunktion als gewichtete Summe der Funktionen zu berechnen, also

$$env^*(t) = (1-\lambda) \cdot env(t, \alpha^l, \beta^l) + \lambda \cdot env(t, \alpha^r, \beta^r)$$

Dieser Ansatz hat allerdings den Nachteil, dass die vom Morphing erzeugte Envelopefunktion aus mehr Teilstücken besteht, als der Eingabeenvelope. Eine Verschachtelung der Operatoren, wie es der Morph-Plan erlaubt, würde also immer komplexere Envelopefunktionen erzeugen.

Eine andere Idee wäre, die Envelopeparameter selbst zu kombinieren, also

$$env^*(t) = env(t, (1-\lambda) \cdot \alpha^l + \lambda \cdot \alpha^r, (1-\lambda) \cdot \beta^l + \lambda \cdot \beta^r)$$

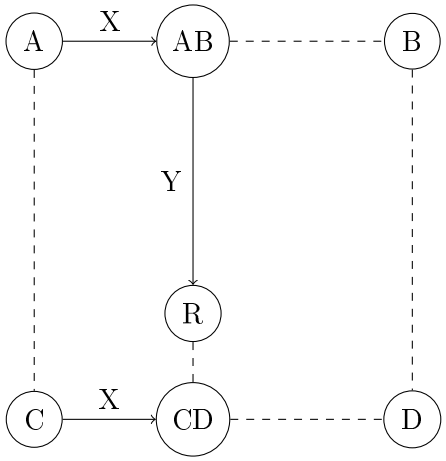


Abbildung 4.7: Gitterbasierte Morphing von vier Instrumenten

Dies würde sich besser für die in Kapitel 4.1 vorgestellte modulare Architektur eignen, da Eingabeenvelopes und Ausgabeenvelope hier durch jeweils zwei Parameter (α und β) beschrieben würden. Man könnte also lineare Morph-Operatoren weiterhin beliebig verschachteln.

Insgesamt haben wir allerdings die Vermutung, dass beide Ansätze nicht in allen Fällen die Ergebnisse verbessern, und daher das Morphing ohne Envelope robuster ist, als das Morphing mit Envelope.

4.3 Implementation des gitterbasierten Morphings

Beim gitterbasierten Morphing - wir hatten es in Kapitel 4.1.3 vorgestellt - werden Instrumente auf einem $W \times H$ Gitter angeordnet, also jeweils W Instrumente nebeneinander und H Instrumente übereinander. Es gibt den einfachen Fall, in dem entweder $W = 1$ oder $H = 1$ ist. Dann sind die Instrumente auf einer Gerade angeordnet, sodass in jedem Fall nur zwei Instrumente durch Morphing kombiniert werden müssen. Dies entspricht dann genau den Berechnungen des linearen Morphings.

Im Normalfall werden jeweils vier Instrumente kombiniert. In Abbildung 4.7 ist die Situation grafisch dargestellt. Die Ausgangsinstrumente A, B, C und D befinden sich an den Ecken. Gesucht ist der resultierende Klang R, der dessen X-Position durch den Steuerwert X und dessen Y-Position durch den Steuerwert Y festgelegt ist. Diese können vom Benutzer beliebig gewählt werden.

Um R zu berechnen gehen wir wie folgt vor: zunächst berechnen wir aus A und B mit dem Steuerwert X mittels linearem Morphing den zusammengesetzten Klang AB. Dann berechnen wir aus C und D erneut mit dem Steuerwert X mittels linearem Morphing den zusammengesetzten Klang CD. Schließlich kombinieren wir wieder mit linearem Morphing AB und CD mit dem Steuerwert Y zu R.

Insgesamt bedeutet dies, dass wir R berechnen können, indem wir dreimal das bereits beschriebene lineare Morphing anwenden.

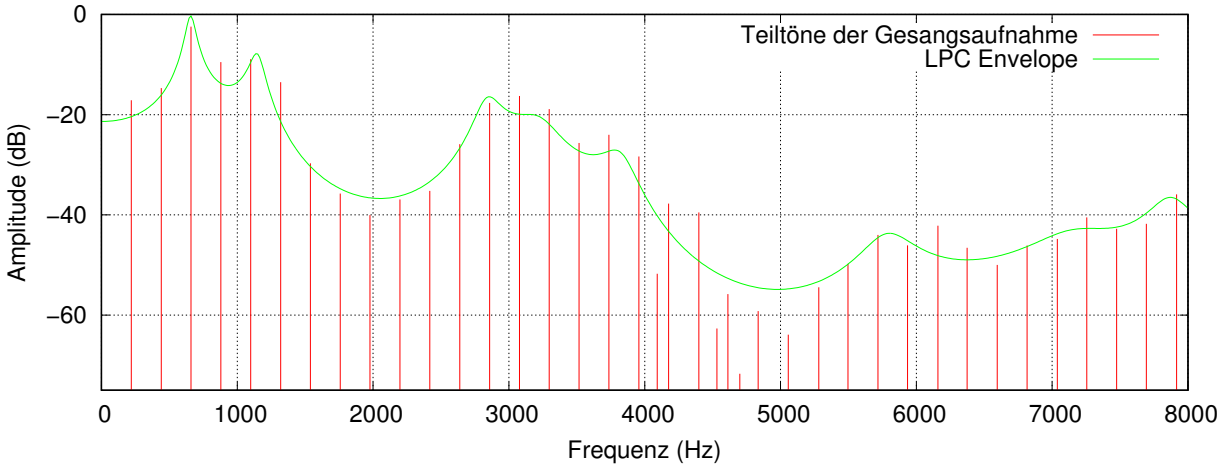


Abbildung 4.8: Analyse Gesang mit LPC-Envelope ($K = 50$)

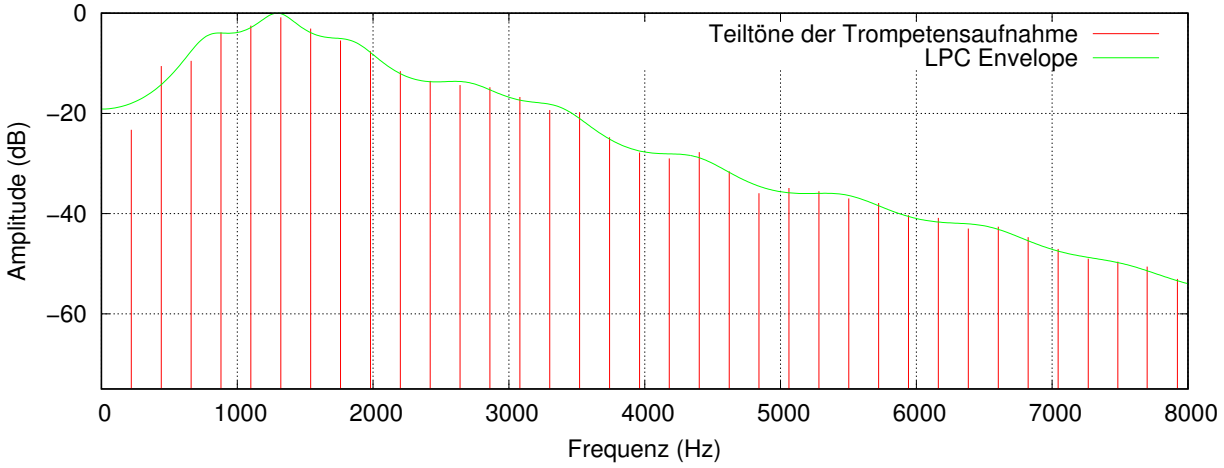


Abbildung 4.9: Analyse der Trompete mit LPC-Envelope ($K = 50$)

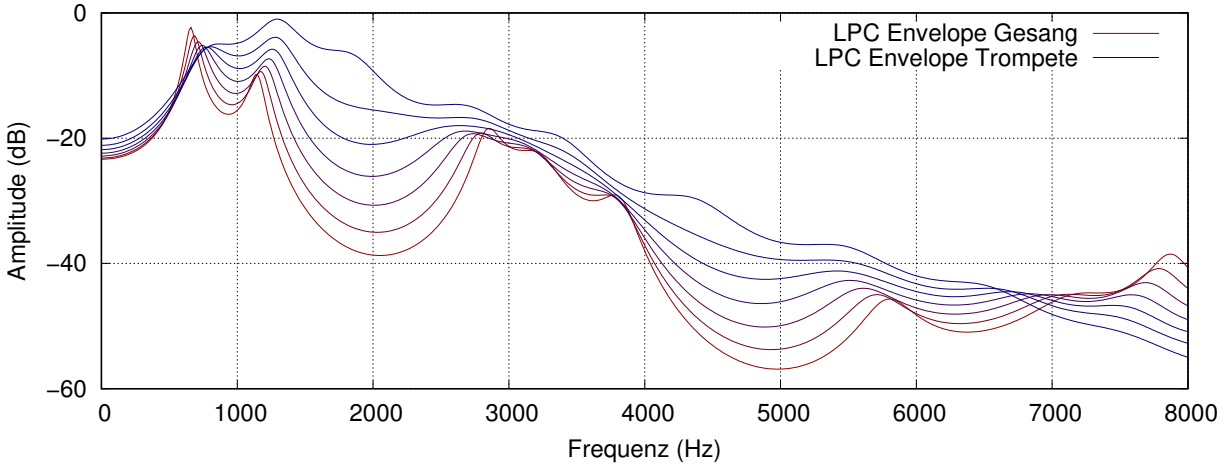


Abbildung 4.10: Morphing des LPC-Envelopes ($K = 50$), Gesang - Trompete

4.4 Experimentelles Morphing mittels LPC

4.4.1 Motivation und Funktionsweise

Die bisher beschriebenen Morphing Algorithmen basieren direkt auf der Zerlegung des Signals in einzelne Teiltöne. Dabei findet immer eine 1:1 Zuordnung statt, es wird also jeweils ein Teilton des einen Eingabesignals mit einem Teilton des anderen Eingabesignals verrechnet.

Der hier beschriebene experimentelle LPC Morphing Algorithmus wählt einen anderen Zugang: statt die Teiltöne einzeln zu betrachten, berechnen wir zunächst für jedes Signal eine spektrale Hüllkurve („Spectral Envelope"). Das Morphing basiert auf der Transformation dieser Hüllkurven.

Die unterschiedliche Betrachtung des Spektrums der Eingabe für das Morphing sind für eine Gesangsaufnahme in Abbildung 4.8 und für eine Trompetenaufnahme Abbildung 4.9 zu sehen. Die roten Linien entsprechen hierbei den Teiltönen, die grüne Linie entspricht der spektralen Hüllkurve[1]. In Abbildung 4.10 sieht man schließlich den Übergang der spektralen Hüllkurve des Gesangsspektrums (rot) in die spektrale Hüllkurve des Trompetenspektrums (blau). Die Grafiken wurden mit $K = 50$ erstellt, da so die Hüllkurven sehr genau sind, wir beschreiben später in Kapitel 4.4.5 warum wir diese Ordnung nicht für den finalen Test wählen.

Die Hoffnung bei der Implementation dieses Verfahrens war, dass dieser alternative Ansatz qualitativ bessere Übergänge zwischen Tönen liefern würde. Um es vorweg zu nehmen, zumindest in ersten Experimenten konnte keine Verbesserung der Klangqualität beobachtet werden, wohl aber in einigen Fällen zusätzliche Störgeräusche (vor allem bei sehr hohen Werten von K), die das zuvor beschriebene (dB)-lineare Morphing nicht aufweist. Wir beschreiben das Verfahren dennoch, und überlassen ein endgültige Beurteilung dem in Kapitel 5 durchgeführten Hörversuch mit Testpersonen.

4.4.2 Bestimmung der LPC Koeffizienten

Für diesen Morphing Algorithmus benötigen wir einige zusätzliche Berechnungen, die zur Bestimmung der spektralen Hüllkurve dienen. Dieser zusätzliche Analyseschritt muss ausgeführt werden, nachdem die Instrumentdaten fertig sind, insbesondere auch nach der der Anwendung der automatischen Stimmalgorithmen aus Kapitel 2.7.1, da dort die Frequenzen der Teiltöne nochmals modifiziert werden.

Wir betrachten als Ausgangspunkt zunächst einen linearen Prädiktor, der das aktuelle Sample aus den vergangenen K Samples vorhersagt.

$$s^*(n) = -\sum_{k=1}^{K} a_k s(n-k)$$

Die Berechnung der Koeffizienten a_k, für die der Fehler minimal wird, ist ein Standardproblem. Es gibt dafür mehrere Verfahren, wir verwenden die in [PK95] beschriebene Kovarianzmethode. Für jeden Analyseframe mit den zugehörigen Amplituden, Frequenzen und Phasen (A_p, F_p, Φ_p) berechnen wir zunächst das Eingabesignal für den linearen Prädiktor (ohne Synthesewindow) als Summe aller Teiltöne. Der Noiseanteil wird nicht berücksichtigt. Die Phasen können ebenfalls

[1] Da die LPC Berechnung die Lautstärke des Signals nicht berücksichtigt, wurden die Hüllkurven der Grafiken manuell so verschoben, dass sie zu den Amplituden der Teiltöne passen.

ignoriert werden, da sie für die spektrale Hüllkurve keine Rolle spielen.

$$s(n) = \sum_{p=1}^{P} A_p cos\left(2\pi \frac{F_p}{S} n\right)$$

Die Länge dieses Signals entspricht der bei der Analyse verwendeten Framelänge. Aus $s(n)$ bestimmen wir die K LPC Koeffizienten des linearen Prädiktors

$$a_1, a_2, \ldots, a_K$$

Damit die LPC Koeffizienten verschiedener Eingabesignale vergleichbar und später per Morphing kombinierbar sind, verwenden wir für $s(n)$ nicht die Abtastrate F_s des Eingabesignals. Stattdessen wird $s(n)$ immer mit der einheitlichen Abtastrate

$$S = 44100$$

berechnet. Diese ist so gewählt, dass Teiltöne im hörbaren Spektrum darstellbar sind, falls Teiltöne mit $F_p \geq S/2$ auftreten werden diese nicht berücksichtigt.

Um sicherzustellen, dass die LPC Koeffizienten einen stabilen Filter beschreiben berechnen wir an dieser Stelle die Wurzeln des charakteristischen Polynoms und verschieben die Wurzeln in den Einheitskreis wenn nötig. Details zu diesem Schritt finden sich in [PTVF07, Chapter 13.6.2.].

Zu den so gewonnenen Koeffizienten gehört der inverse Filter

$$A(z) = 1 + \sum_{k=1}^{K} a_k z^{-k}$$

und die Übertragungsfunktion

$$H(z) = \frac{1}{A(z)}$$

Die durch den linearen Prädiktor definierte spektrale Hüllkurve $h(\omega)$ erhält man, in dem man $|H(z)|$ auf dem Einheitskreis evaluiert:

$$h(\omega) = |H(e^{j\omega})| \text{ für } \omega \in [0, \pi]$$

4.4.3 Berechnung der LSF Koeffizienten

Zwei LPC Koeffizientensätze von zwei Eingabeframes können nicht direkt interpoliert werden, eignen sich also nicht direkt für den Morphingalgorithmus. Wir verwenden daher die als „Line Spectral Frequencies“ bekannte Transformation der Parameter, um spektrale Hüllkurven interpolieren zu können. Wir orientieren uns in der mathematischen Darstellung an [ZSL+98]. Dieses Verfahren ist auch als „Line Spectral Pairs“ bekannt (da die Wurzeln der hier definierten Polynome $P(z)$ und $Q(z)$ paarweise um den Einheitskreis auftreten).

Wir gehen für die folgenden Schritte davon aus, dass die Anzahl der LPC Koeffizienten K gerade ist. Am Anfang der Bestimmung zerlegen wir den inversen Filter $A(z)$ in zwei Polynome, die jeweils den Grad $K + 1$ haben, $P(z)$ und $Q(z)$, sodass

$$A(z) = \frac{P(z) + Q(z)}{2}$$

und zwar

$$\begin{aligned} P(z) &= A(z) + z^{-(K+1)}A(z^{-1}) \\ &= 1 + \sum_{k=1}^{K}(a_k + a_{K+1-k})z^{-k} + z^{-(K+1)} \\ Q(z) &= A(z) - z^{-(K+1)}A(z^{-1}) \\ &= 1 + \sum_{k=1}^{K}(a_k - a_{K+1-k})z^{-k} - z^{-(K+1)} \end{aligned}$$

Aufgrund der Konstruktion liegen deren Wurzeln auf dem Einheitskreis, sodass nur die zu den Wurzeln gehörigen Frequenzen gespeichert werden müssen. Da die Polynome reelle Koeffizienten haben, reicht der Bereich $\omega \in [0, \pi]$ aus, die anderen Wurzeln liegen symmetrisch dazu.

Ein Algorithmus zur Bestimmung der Wurzeln von $P(z)$ und $Q(z)$ findet sich in [KR86], wir verwenden eine einfachere Implementation: indem wir $|P(z)|$ (bzw. $|Q(z)|$) auf einem Gitter um den Einheitskreis evaluieren, bilden sich an den Gitterpunkten nahe an den Wurzeln lokale Minima. So lassen sich die gewünschten Frequenzen ermitteln.

Dabei hat $P(z)$ immer eine Wurzel bei $\omega = \pi$, wir bezeichnen die Frequenzen der anderen Wurzeln mit

$$\omega_1, \omega_2, \ldots, \omega_{K/2}$$

$Q(z)$ hat immer eine Wurzel bei $\theta = 0$, die Frequenzen der anderen Wurzeln nennen wir

$$\theta_1, \theta_2, \ldots, \theta_{K/2}$$

Die Gesamtzahl der Parameter beträgt also in beiden Darstellungen K. Die Polynome $P(z)$ und $Q(z)$ lassen sich damit faktorisieren:

$$\begin{aligned} P(z) &= (1 + z^{-1}) \prod_{k=1}^{K/2}(1 - z^{-1}e^{j\omega_k})(1 - z^{-1}e^{-j\omega_k}) \\ &= (1 + z^{-1}) \prod_{k=1}^{K/2}(1 - 2\cos\omega_k z^{-1} + z^{-2}) \\ Q(z) &= (1 - z^{-1}) \prod_{k=1}^{K/2}(1 - z^{-1}e^{j\theta_k})(1 - z^{-1}e^{-j\theta_k}) \\ &= (1 - z^{-1}) \prod_{k=1}^{K/2}(1 - 2\cos\theta_k z^{-1} + z^{-2}) \end{aligned}$$

Durch diese faktorisierte Darstellung von $P(z)$ und $Q(z)$, die nun ausschließlich von den LSF Koeffizienten ω_k und θ_k ($k \in [1, K/2]$) abhängt, lässt sich die Übertragungsfunktion berechnen als:

$$H(z) = \frac{1}{A(z)} = \frac{2}{P(z) + Q(z)}$$

Und schließlich ergibt sich eine neue Berechnung der spektralen Hüllkurve $h(\omega)$:

$$h(\omega) = |H(j\omega)| = \left| \frac{2}{P(e^{j\omega}) + Q(e^{j\omega})} \right| \text{ für } \omega \in [0, \pi]$$

4.4.4 Implementation des LPC Morphings

Da wir an dieser Stelle für jeden Frame zusätzlich die LSF Koeffizienten berechnet haben, können wir hiermit ein neues Morphingverfahren implementieren. Es reicht wiederum aus, zu beschreiben

wie zwei Eingabeframes, der „linken“ Frame und der „rechten“ Frame geeignet kombiniert werden können. Neu ist hier, dass dazu die spektrale Hüllkurve, die durch die LSF Parameter definiert ist, verwendet wird um die Amplituden der Ausgabe zu bestimmen. Wie oben wird der Parameter $\lambda \in [0,1]$ als Steuerung verwendet, um die Gewichtung des linken und rechten Frames zu bestimmen.

Die Eingabeparameter sind in Tabelle 4.3 zu sehen. Die Frequenzen, Amplituden und der Noiseanteil entsprechen den in Kapitel 4.2 verwendeten Eingabeparametern. Neu sind die insgesamt jeweils K LSF Parameter, die die Wurzeln von $P(z)$ und $Q(z)$ repräsentieren.

Parameter	**linker Frame**	**rechter Frame**
Frequenzen	$F_1^l, \ldots, F_P^l$	$F_1^r, \ldots, F_Q^r$
Amplituden	$A_1^l, \ldots, A_P^l$	$A_1^r, \ldots, A_Q^r$
Noiseanteil	$NOISE_0^l, \ldots, NOISE_{31}^l$	$NOISE_0^r, \ldots, NOISE_{31}^r$
Wurzeln $P(z)$	$\omega_1^l, \ldots, \omega_{K/2}^l$	$\omega_1^r, \ldots, \omega_{K/2}^r$
Wurzeln $Q(z)$	$\theta_1^l, \ldots, \theta_{K/2}^l$	$\theta_1^r, \ldots, \theta_{K/2}^r$

Tabelle 4.3: Eingabeparameter für das lineare Morphing

Aus diesen lassen sich die spektrale Hüllkurve des linken Frames $h^l(\omega)$ und die spektrale Hüllkurve des rechten Frames $h^r(\omega)$ ermitteln. Für das LPC Morphing entscheidend ist jedoch, dass sich auch die gewünschte spektrale Hüllkurve des Ausgabeframes berechnen lässt, da die LSF Parameter direkt interpoliert werden können. Wenn wir die Wurzeln des linken und rechten Frames mit dem Steuerparameter λ kombinieren, erhalten wir

$$\omega_k^i = (1-\lambda) \cdot \omega_k^l + \lambda \cdot \omega_k^r \text{ für } k \in [1, K/2]$$
$$\theta_k^i = (1-\lambda) \cdot \theta_k^l + \lambda \cdot \theta_k^r \text{ für } k \in [1, K/2]$$

Damit wird die interpolierte spektrale Hüllkurve $h^i(\omega)$ definiert, diese entspricht der gewünschten Ausgabehüllkurve. Ein Beispiel für die so durch λ festgelegtes Zwischenschritte zwischen $h^l(\omega)$ und $h^r(\omega)$, die wir eben als $h^i(\omega)$ bezeichnet hatten, haben wir zuvor bereits in Abbildung 4.10 grafisch dargestellt.

Da das LPC Morphing lediglich die Ausagabeamplituden mittels der spektralen Hüllkurven $h^l(\omega)$, $h^r(\omega)$ und $h^i(\omega)$ anders behandelt, sind praktisch alle Berechnungsschritte mit dem in Kapitel 4.2 beschriebenen linearen Morphing identisch. Das bedeutet, dass die Berechnung des stochastischen Anteils $NOISE_b$, die Zuordnung der Teiltöne über die Sortierung der Teiltöne nach der Amplitude, und die Bestimmung der Frequenzen der Ausgabe 1:1 übernommen werden.

Die für das LPC Morphing verwendete Amplitudenberechnung funktioniert für nicht zugeordnete Teiltöne wie in Kapitel 4.2.7. Die spektralen Hüllkurven werden ausschließlich verwendet, wenn wenn eine Zuordnung des p-ten Teilton aus dem linken Eingabeframe zum q-ten Teilton des rechten Eingabeframes erfolgt ist. In diesem Fall muss aus A_p^l und A_q^r die Lautstärke des resultierenden Teiltons A ermittelt werden.

Es gibt nun drei Werte der spektralen Hüllkurven, die eine Rolle spielen:

$$H_l = h^l\left(\frac{2\pi \cdot F_p^l}{S}\right)$$
$$H_r = h^r\left(\frac{2\pi \cdot F_q^r}{S}\right)$$
$$H_i = h^i\left(\frac{2\pi \cdot F}{S}\right)$$

Diese geben den Wert der spektralen Hüllkurve $h^l(\omega)$ des linken Frames bei der Frequenz F_p^l des p-ten Teiltons, den Wert der spektralen Hüllkurve $h^r(\omega)$ bei der Frequenz F_q^r des q-ten Teiltons und den Wert der spektralen Hüllkurve $h^i(\omega)$, die für die Ausgabe durch Interpolation der Wurzeln von $P(z)$ und $Q(z)$ gewonnen wurde, bei der Frequenz F.

Mit diesen Werten berechnen wir die Amplitude des resultierenden Teiltons als

$$A = db2factor((1-\lambda)\cdot(db(A_p^l) - db(H_l)) + \lambda\cdot(db(A_q^r) - db(H_r)) + db(H_i))$$

mit den Konvertierungsformeln:

$$db(x) = \max(20\log_{10}(x), -96)$$
$$db2factor(x) = 10^{x/20}$$

Diese Amplitudenberechnung ermittelt zunächst für den linken Frame die Differenz der spektralen Hüllkurve und des tatsächlichen Werts A_p^l. Es wird also ermittelt, wieviel A_p^l unter oder über der spektralen Hüllkurve liegt. An dieser Stelle sei daran erinnert, dass die Hüllkurve generell nicht die Lautstärke des Signals enthält, sodass diese Differenz sowohl die relative Amplitude des Teiltons als auch die Amplitudendifferenz zwischen $h^l(\omega)$ und der Lautstärke des Frames abbildet.

Für den rechten Frame wird analog dazu die relative Amplitude zu $h^r(\omega)$ berechnet. Aus beiden Spektren wurden also die spektrale Hüllkurve durch Subtraktion entfernt. Grob kann man sagen, dass diese damit flach werden, da die charakteristische Form der Hüllkurve - und damit der charakteristische Klang - nun nicht mehr enthalten sind.

Als letzten Schritt addieren wir nach der Interpolation mit λ die gewünschte Form der Ausgabehüllkurve $h^i(\omega)$ wieder. Insgesamt haben wir durch dieses Verfahren die relative Lautstärke der Teiltöne bezüglich ihrer jeweiligen spektralen Hüllkurve ermittelt, gemorpht, und wieder in eine absolute Lautstärke durch Addition der Ausgabehüllkurve umgewandelt.

4.4.5 Wahl der LPC Ordnung

Bisher haben wir das LPC Morphing unabhängig von der Anzahl der LPC Koeffizienten, der LPC Ordnung K beschrieben. Die einzige Bedingung ist, dass K gerade ist. Der Klang wird durch die LPC Ordnung beeinflusst. Um zu entscheiden, welcher Wert günstig ist, haben wir daher den Klang des Morphings mit

$$K = 10, 18, 26, 34, 42, 50$$

verglichen. Dabei wurden die gleichen Beispiele verwendet, die wir im Hörversuch im Kapitel 5 verwenden werden, um die Klangqualität des LPC Morphings zu bewerten. Generell würden wir davon gerne eine möglichst hohe Ordnung wählen, da der lineare Prädiktor mit mehr Koeffizienten bessere Voraussagen liefert, und damit die spektrale Hüllkurve besser zu den Eingabedaten passt, je höher K ist.

Bei Ordnung $K = 50$ gibt es jedoch in einigen Beispielen deutlich hörbare störende Resonanzgeräusche, die bei normalem dB-linearen Morphing nicht auftreten. Die Intensität der Störungen nehmen mit der Ordnung ab, treten aber auch noch bei $K = 42$ und $K = 34$ auf. Tendenziell waren eher hohe Töne betroffen. Leider konnte nicht abschließend geklärt werden, wie genau diese Resonanzgeräusche entstehen. Denkbar wäre ein Fehler in der Implementation oder wahrscheinlicher, dass die hier gegebene Spezifikation in diesen Fällen ungünstige Ergebnisse mit den Analysedaten liefert, wenn K hoch ist.

Abbildung 4.11 und Abbildung 4.12 zeigen ein prinzipielles Problem[2], welches hohe Ordnungen betrifft, ob dies unsere Resonanzgeräusche verursacht, ist nicht klar. Bei der tieferen Aufnahme,

[2]Um die LPC Kurven klarer unterscheidbar zu machen, wurden sie auseinander geschoben.

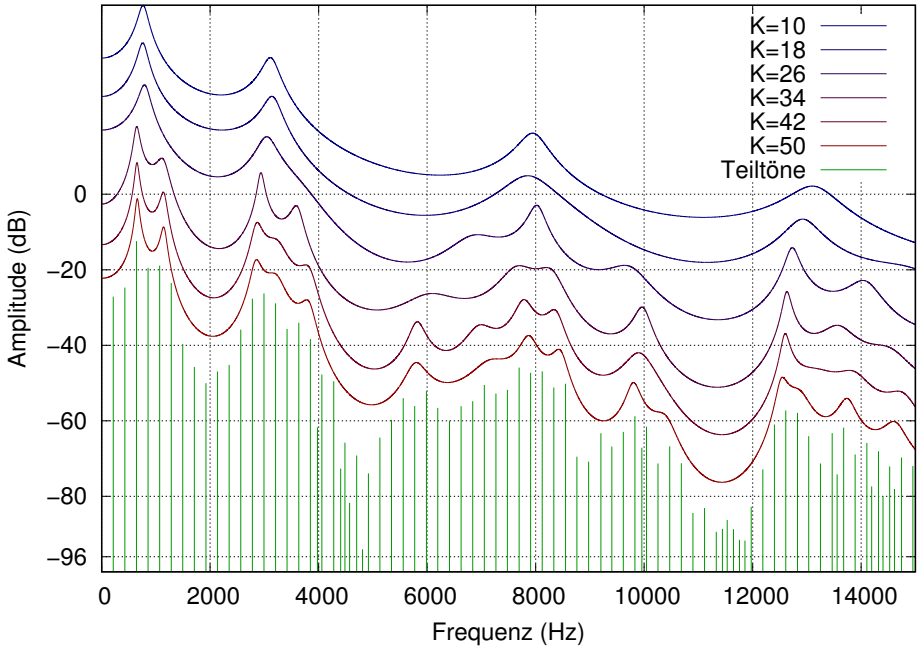


Abbildung 4.11: Gesangsaufnahme 220 Hz, Einfluss der LPC Ordnung K

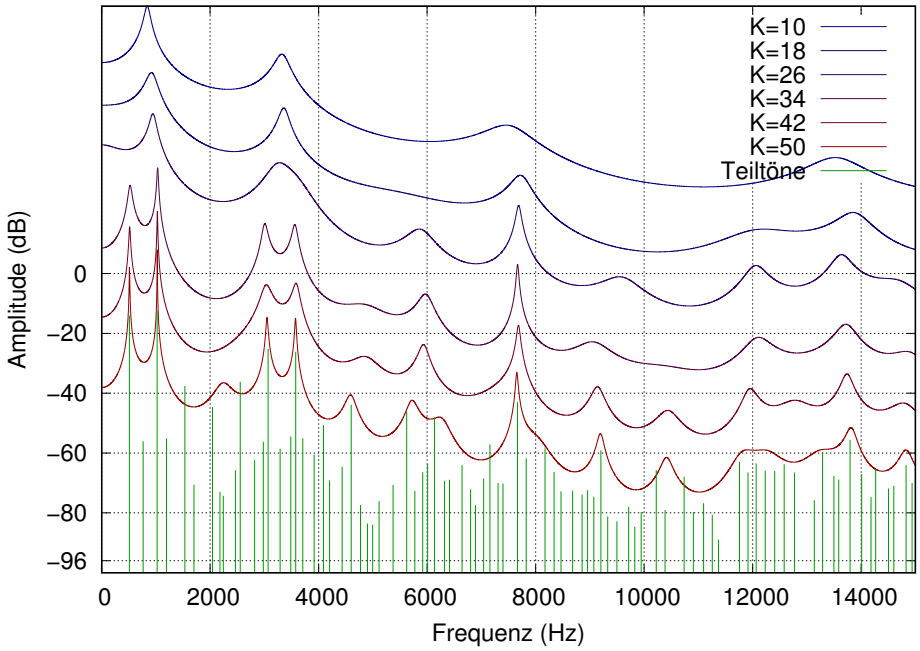


Abbildung 4.12: Gesangsaufnahme 523 Hz, Einfluss der LPC Ordnung K

die die Grundfrequenz 220 Hz hat, wirkt sich eine höhere LPC Ordnung durchweg positiv aus. Hier liefert $K = 50$ die besten Ergebnisse, das Spektrum wird umso besser approximiert, je höher die Ordnung ist.

Bei der anderen Aufnahme mit der Grundfrequenz 523 Hz liegen die Teiltöne des Spektrums weiter auseinander. Dies führt dazu, dass es nicht so viele Stützstellen gibt, die die spektrale Hüllkurve festlegen. Bei höheren Ordnungen $K = 34, 42, 50$ bilden sich zunehmend stärkere Resonanzen um die lauten Teiltöne. Die Hüllkurve bricht zwischen einzelnen Teiltönen ein, wodurch nicht mehr die beabsichtigte einhüllende Funktion des Spektrums entsteht, was für das Morphing unerwünscht ist. Lediglich bei niedrigen Ordnungen hat der LPC Envelope eine für das Morphing erwünschte Form.

Zusammenfassend wählen wie LPC Ordnung so, dass bei dem Testmaterial keine Resonanzgeräusche auftreten, andererseits die spektralen Hüllen möglichst genau werden, also $K = 26$.

4.5 Gesamtperformance des Morphings

Zum Abschluss der Beschreibung der Morphingverfahren gehen wir an dieser Stelle noch auf die Performance des Morphings ein. Dabei hängt die erreichbare Geschwindigkeit zum einen, wie bei der Synthese in Kapitel 3.6, von der Anzahl der Teiltöne ab. Daher verwenden wir zum Testen wie zuvor jeweils eine eher hohe Grundfrequenz von 440 Hz mit einer geringeren Anzahl von Teiltönen und eine eher niedrige Grundfrequenz von 65,4 Hz mit einer höheren Anzahl von Teiltönen. Zum anderen wird die Geschwindigkeit vom Morphing Verfahren beeinflusst. Wir testen hier die drei Varianten des Morphing zwischen zwei Instrumenten, lineares Morphing, dB-lineares Morphing und LPC Morphing mit $K = 26$. Als viertes Verfahren testen wir das gitterbasierte Morphing, bei dem vier Instrumente kombiniert werden.

Normiert werden die Ergebnisse so, dass jeweils die Dauer in Nanosekunden pro Ausgabesample angegeben werden. Die Messungen wurden auf einem 64-bit Linux System (AMD Phenom(tm) 9850 Quad-Core Processor) durchgeführt. Als Abtastrate für den Test verwenden wir 48 000 Hz. Für die Morphingverfahren, die zwei Instrumente kombinieren, verwenden wir Trompete und Geige für die hohe Grundfrequenz, Cello und Fagott für die tiefe Grundfrequenz. Beim gitterbasierten Morphing verwenden wir Trompete, Geige, Oboe und Altsaxophon bei 440 Hz. Da wir nur drei Instrumente zur Verfügung haben, die bei 65,4 Hz spielbar sind, nämlich Cello, Fagott und Bass-Posaune, wurden außer diesen für den gitterbasierten Test das Cello doppelt verwendet (um den Einfluss auf den Benchmark minimal zu halten, wurde das Cello an den gegenüberliegenden Ecken links/oben und rechts/unten platziert).

Grundfrequenz Hz	**Verfahren**	**Synthese** ns/Sample	**Morphing** ns/Sample	**Gesamt** ns/Sample	**Polyphonie** Stimmen
440,0	Linear Morph	52,81	115,30	168,11	123,93
	dB Linear Morph	53,20	161,99	215,19	96,81
	LPC Morph, K=26	57,39	348,16	405,55	51,37
	Grid Morph	58,25	241,90	300,15	69,41
65,4	Linear Morph	89,54	232,63	322,17	64,67
	dB Linear Morph	89,78	292,02	381,80	54,57
	LPC Morph, K=26	94,47	590,96	685,43	30,39
	Grid Morph	85,18	469,60	554,78	37,55

Tabelle 4.4: Gemessene Performance unterschiedlicher Morphing Verfahren

In Tabelle 4.4 und Abbildung 4.13 sind die Ergebnisse dargestellt. Dabei werden die Gesamtkosten auf die Kosten für die Synthese und die Kosten für das Morphing aufgeteilt. Man sieht, dass in allen Fällen das Morphing den grösseren Teil der Zeit benötigt. Die relative Geschwindigkeit der Verfahren ist bei beiden Grundfrequenzen gleich. Das schnellste Verfahren ist das lineare Morphing, das zwei Instrumente kombiniert und die Amplituden dabei linear bestimmt. Eine zusätzliche Umrechnung nach und von den dB-Amplituden im dB linearen Morphing macht dieses etwas langsamer, und damit zum zweitschnellsten Verfahren.

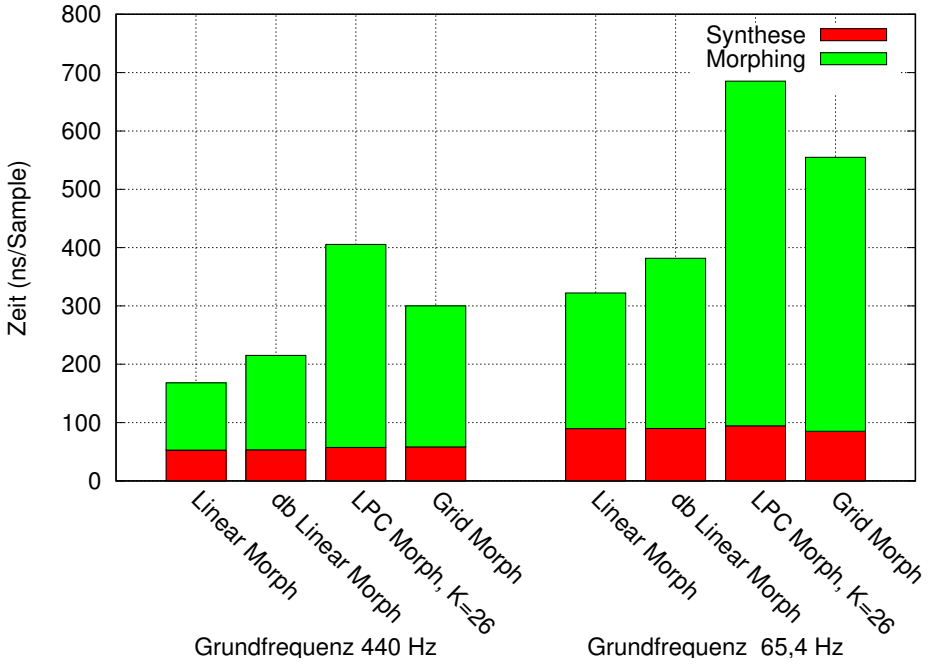


Abbildung 4.13: Performance der Morphingverfahren

Obwohl das LPC Morphing ebenfalls nur zwei Instrumente kombiniert, ist es das langsamste Verfahren im Test. Dies liegt hauptsächlich daran, dass hierbei für jeden zugeordneten Teilton die spektralen Hüllkurven $h^l(\omega)$, $h^r(\omega)$ und $h^i(\omega)$ berechnet werden, wobei der komplexe Wert der jeweils zugehörigen Polynome $P(z)$ und $Q(z)$ ermittelt werden muss. Wir haben hier die LPC Ordnung $K = 26$ verwendet, bei höheren Ordnungen wäre der Aufwand noch grösser.

Schließlich ist das gitterbasierte Morphing damit das drittschnellste Verfahren, obwohl hierbei für jeden Frame drei einzelne lineare Morphingschritte notwendig sind um die vier Instrumente zu kombinieren. Diese entsprechen dem dB linearen Morphing, da wir beim gitterbasierten Morphing Amplituden immer als dB Werte behandeln, es gibt hier keine Einstellmöglichkeit. An dieser Stelle kann noch angemerkt werden, dass das gitterbasierte Morphing (welches ja drei interne lineare Morphingschritte enthält) nicht ganz dreimal so langsam wie die anderen beiden Verfahren ist, da der interne Morphingschritt etwas effizienter implementiert ist als in den anderen Verfahren.

Als letztes gehen wir auf die Frage ein: *ist das Morphing schnell genug*? Dabei betrachten wir nur die beiden Verfahren von denen wir annehmen, dass sie in Anwendungen am nützlichsten sind, dB lineares Morphing zur Kombination von zwei Instrumenten und gitterbasiertes Morphing zur Kombination von vier Instrumenten. Als Bewertungskriterium dient uns die theoretische Polyphonie, also die maximale Anzahl der Stimmen die gleichzeitig in Echtzeit berechnet werden kann, geschätzt aus der Dauer der Berechnung eines Samples. In Kapitel 3.6 hatten wir die Synthese auf die gleiche Weise bewertet.

Die theoretische Polyphonie ist in der letzten Spalte in Tabelle 4.4 angegeben. Für das dB lineare Morphing liegt diese bei 96,81 Stimmen bzw. 54,57 Stimmen, je nach Tonhöhe. Für das gitterbasierte Morphing sind die Werte 69,41 Stimmen bzw. 37,55 Stimmen. Wir bewerten dies als schnell genug, auch für anspruchsvollere Echtzeitanwendungen.

Inwiefern die theoretischen hochgerechneten Werte auch der Performance in der Praxis entsprechen, ist eine andere Frage. Wir nehmen an, dass bei mehreren parallel gespielten Noten der Zugriff auf die benötigten Analysedaten eventuell langsamer wird, da diese nicht mehr so gut im CPU Cache gehalten werden können. Weiterhin haben wir hier eine lange Note gemessen, bei sehr kurzen Noten könnte die Initialisierungszeit wichtig für die Gesamtperformance sein.

5 Evaluation

5.1 Durchführung eines Hörversuchs

5.1.1 Überblick

In den letzten Kapiteln haben wir die Analyse, die Synthese und das Morphing von Klängen beschrieben. Dabei wurden die Algorithmen und die zugehörigen Parameter (wie z.B. die Fensterfunktion, die Länge der Frames, der Grad K des LPC Polynoms) angegeben. In diesem Kapitel bewerten wir die subjektive Qualität, die dadurch insgesamt erreicht wird, in vier verschiedenen Kategorien.

Wir orientieren uns bei unserem Hörversuch grob an zwei Empfehlungen der International Telecommunication Union, Radiocommunication Sector: ITU-R. Die eine Empfehlung [IR15a] beschreibt den ABC/HR Test, bei dem kleine Qualitätsunterschiede zwischen einem Originalsignal und einem leicht verändertem Signal gemessen werden. Diese Vorgehensweise verwenden wir bei der Bewertung der Analyse/Synthese Qualität. Die andere Empfehlung [IR15b] beschreibt den MUSHRA Test, mit dem sich zum Beispiel die subjektive Qualität mehrerer Audio Codecs vergleichen lässt.

Der wichtigste Unterschied zwischen unserem Hörversuch den von der ITU-R beschriebenen Verfahren ist, dass sowohl bei einem ABC/HR als auch bei einem MUSHRA-Test davon ausgegangen wird, dass es immer ein Original (eine Referenz) gibt, sodass man die Frage immer so stellen kann: wie viel schlechter ist ein von einer (Codec)-Software produziertes Testbeispiel verglichen mit dem „perfekten“ Original. In unserem Fall geht es aber darum, neue Klänge zu bewerten, beispielsweise den Übergang von einem Geigenton in einen Oboenton. Dafür gibt es kein perfektes Original.

Wir haben daher das MUSHRA Verfahren modifiziert, und stellen in diesem Fall dem Hörer zwei Referenzen zur Verfügung, den Geigenton und den Oboenton, und erfragen die Bewertung des Übergangs relativ zu diesen Referenzsignalen. Der genaue Ablauf wird im folgenden Kapitel beschrieben.

5.1.2 Ablauf der Bewertungsphase

Der Hörversuch läuft immer beginnend von Kategorie 1 bis zu Kategorie 4 ab. Am Anfang jeder Kategorie wird den Versuchspersonen beschrieben, was genau bewertet werden soll. Innerhalb jeder Kategorie gibt es eine Anzahl von Tests, beim Morphing von Einzelnoten (Kategorie 2) wären dies zum Beispiel Übergänge verschiedener Instrumentkombinationen. Die Reihenfolge der Tests innerhalb einer Kategorie wird zufällig gewählt, sodass jede Versuchsperson eine andere Reihenfolge erhält. Damit sollen durch die Reihenfolge der Präsentation entstehende Bewertungsunterschiede vermieden werden.

Ein typischer Screenshot der Bewertungssoftware findet sich in Abbildung 5.1. An der linken Seite des Bildschirms ist die Bewertungsskala angegeben, die der Skala des MUSHRA-Tests entspricht. Hierbei ist die beste Bewertung 100, die schlechteste 0, wobei ein Wert zwischen 100 und 80 als ausgezeichnet gilt, zwischen 80 und 60 als gut, zwischen 60 und 40 als ordentlich, zwischen 40 und 20 als mäßig und zwischen 20 und 0 als mangelhaft.

Am unteren Rand befinden sich Knöpfe um das Audiomaterial abzuspielen. Dabei gibt es ein oder mehrere Referenzen (Originale). Diese werden zum Vergleich genutzt, aber nicht bewertet. In unserem Beispielscreenshot sind die Referenzen Oboe und Trompete. Die Referenzen werden

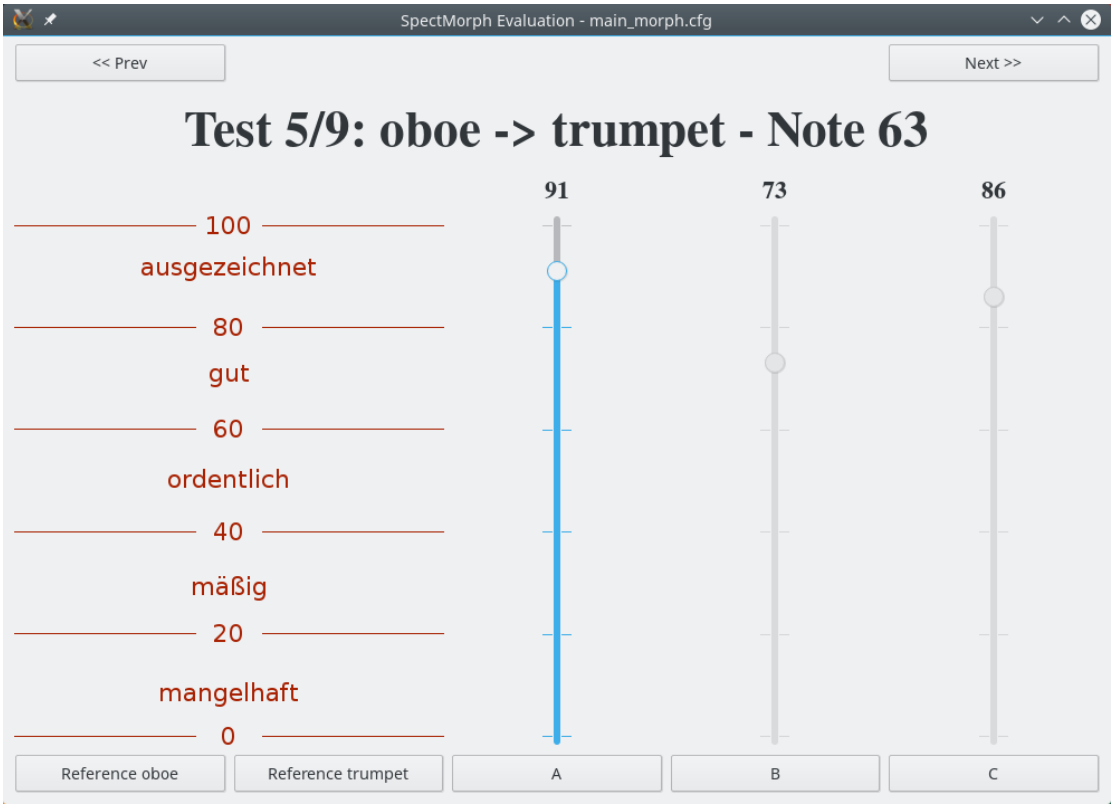


Abbildung 5.1: Screenshot der Bewertungssoftware

benötigt, um die Instrumente zu Vergleichszwecken auch im Original anhören zu können.

Zusätzlich zu den Referenzen gibt es die zu bewertenden Klangbeispiele, hier „A“, „B“ und „C“. Dies sind hier verschiedene Übergänge von Oboe zu Trompete. Die Reihenfolge der Klangbeispiele wird wiederum zufällig gewählt, damit der Test doppelblind wird.

Die Versuchsperson kann bei jedem Test alle Klangbeispiele beliebig oft anhören, um schließlich für jedes zu bewertenden Klangbeispiele eine Punktzahl zwischen 0 und 100 auf der angegebenen Skala zu vergeben. Um falsche Zuordnungen zwischen Klangbeispiel und Bewertung zu vermeiden, kann jeweils nur das Klangbeispiel mit einer Wertung versehen werden welches zuletzt angehört wurde (dies entspricht den Empfehlungen des MUSHRA Tests).

In der Regel kann die Versuchsperson nach einer Einführung die Tests selbstständig bearbeiten.

5.1.3 Wahl der Versuchspersonen

An dem Hörversuch nahmen insgesamt 10 Versuchspersonen teil, davon 9 Männer und eine Frau im Alter von 22 bis 48 Jahren. Der Altersdurchschnitt lag bei 38 Jahren. Von den 10 Versuchspersonen hatten 8 gelernt (mindestens) ein Musikinstrument zu spielen. Wir gehen davon aus, dass das Lernen eines Instruments dies eine gewisse Schulung des Gehörs mit sich bringt, was die Bewertung von Klangbeispielen erleichtert.

5.1.4 Testumgebung

Der Hörversuch fand in einem ruhigen Büroraum statt, und wurde mit Lautsprechern durchgeführt. Es wäre prinzipiell möglich gewesen, den Hörversuch auch mit Kopfhörern durchzuführen.

Allerdings war im Vorfeld bereits klar, dass unsere Analyse/Synthese prinzipbedingt immer eine Phasendifferenz zwischen dem Original und der Resynthese einführt. Mit Kopfhörern lässt sich diese in einigen Beispielen leicht hören.

Für die Bewertung der Qualität war für uns jedoch eher wichtig, welche anderen „Fehler“ entstehen. Da die Phasendifferenz bei der Verwendung von Lautsprechern weniger störend auffällt, haben wir uns daher für diese Variante entschieden.

5.1.5 Zeitablauf

Wir hatten den Test so dimensioniert, dass wir eine Bearbeitungszeit von $30 - 60$ Minuten für realistisch hielten. Nach etwa der Hälfte der Testzeit war eine kurze Pause geplant, um Ermüdung vorzubeugen. Da jeder Testteilnehmer die Beispiele beliebig oft anhören konnte, unterschieden sich die Bearbeitungszeiten deutlich. Die meisten Versuchspersonen blieben zeitlich in etwa im vorgesehenen Rahmen, in manchen Fällen dauerte der Test länger als erwartet. In solchen Fällen wurden ein bis zwei zusätzliche Pausen eingefügt.

5.1.6 Audiomaterial auf der beigelegten CD

Das gesamte Material, welches beim Hörversuch verwendet wurde, findet sich auf der beigelegten CD. Die einzelnen Kategorien, für die die Versuchspersonen Bewertungen abgegeben haben, sind in den in Tabelle 5.1 aufgeführten vier Ordnern abgelegt.

CD Ordner mit Audiomaterial
`Audiomaterial Hoerversuch/Kategorie 1 - Analyse-Synthese`
`Audiomaterial Hoerversuch/Kategorie 2 - Morphing Einzelnoten`
`Audiomaterial Hoerversuch/Kategorie 3 - Crescendo Einzelnoten`
`Audiomaterial Hoerversuch/Kategorie 4 - Morphing Melodie`

Tabelle 5.1: Verzeichnisse mit Audiomaterial auf der CD

Für jeden Einzeltest wurde ein Verzeichnis angelegt, in dem die zusammengehörenden Hörbeispiele als Wave-Dateien gespeichert wurden. Wie bei der Auswertung später auch wurden die Tonhöhen als Midinoten angegeben. Der Verzeichnisname `Cello - Fagott 36` in Kategorie 2 bedeutet also, dass hier die Klangbeispiele für den Übergang von Cello nach Fagott mit einer Midinote 36 als Tonhöhe zu finden sind.

5.2 Statistische Auswertung

5.2.1 Konfidenzintervalle

Durch die im Hörversuch abgegebenen Bewertungen können wir immer nur eine Stichprobe untersuchen. Damit ist es zwar möglich zu bestimmen, wie der Mittelwert der vorliegenden Daten ist, aber dieser entspricht nur bedingt dem wahren Mittelwert (der theoretisch ermittelt werden könnte, wenn wir unendlich vielen Stichproben untersuchen würden).

Zusätzlich zum Mittelwert geben wir daher das 95% Konfidenzintervall an. Dies bedeutet vereinfacht gesagt, dass der wahre Mittelwert mit einer Wahrscheinlichkeit von 95% in diesem Intervall liegt.

Die von uns berechneten und angegebenen Konfidenzintervalle gelten exakt nur unter der Annahme, dass die Werte der Stichprobe normalverteilt sind. Sind die Daten nicht normalverteilt, aber die Stichprobe ist genügend groß, sind die Konfidenzintervalle ebenfalls genau.

Bei der Berechnung der Konfidenzintervalle als Gesamtüberblick liegen zum Beispiel 9 Beispiele vor, die wir 10 Personen präsentiert haben. Insgesamt gibt es also 90 Bewertungen für das dB-lineare Morphing. Zwar sind diese Stichproben mit je 90 Datenpunkten nicht normalverteilt. Aber die Stichprobengrösse ist relativ groß, weshalb die Konfidenzintervalle relativ genau sein werden.

Anders ist es bei Einzelauswertungen, also beispielsweise ein einzelner Übergang von Trompete zu Geige. Jetzt hat unsere Stichprobe nur noch 10 Werte, und zwar die Bewertung der 10 Versuchsteilnehmer. Das von uns berechnete Konfidenzintervall gilt nur exakt unter der Annahme, dass die Werte der Stichprobe normalverteilt sind. Da dies nicht unbedingt sichergestellt ist, entspricht das berechnete Konfidenzintervall in diesem Fall nur noch ungefähr dem gewünschten Intervall.

Wir haben die Konfidenzintervalle in diesem Fall dennoch angegeben, da sie einen Eindruck der Genauigkeit der Schätzung des Mittelwerts geben, jedoch ist aus den genannten Gründen das Konfidenzintervall in diesen Fällen nur als grober Anhaltspunkt zu sehen.

5.2.2 Test von Hypothesen

Im folgenden werden wir die Qualität von verschiedenen Morphing Verfahren direkt vergleichen. Anhand der Stichproben-Mittelwerte können wir nicht immer sagen, ob ein Verfahren besser ist als ein anderes, da diese nur grob den wirklichen Mittelwert abbilden. Wenn die Konfidenzintervalle nicht überlappen, ist die Frage einfach zu entscheiden, da wir dann mit 95%-iger Sicherheit wissen, dass sich die beiden wirklichen Mittelwerte tatsächlich unterscheiden.

Im anderen Fall verwenden wir den t-Test für verbundene Stichproben (paired t-test) um festzustellen ob sich die Qualität beider Verfahren unterscheidet. Dieser t-Test berechnet für jedes einzelne korrespondierende Wertepaar aus den Stichproben die Bewertungsdifferenz. Das bedeutet, dass nur jeweils direkt die Bewertungen der gleichen Person zum gleichen Beispiel verglichen wird. Dies bildet eine bessere Entscheidungsgrundlage als der Vergleich der Gesamtmittelwerte.

Wenn die beiden Verfahren qualitativ gleichwertig sind, ist der Mittelwert der Bewertungsdifferenzen Null. Für unseren Hypothesentest ist dies die sogenannte Nullhypothese, wenn diese zutrifft, reichen die gesammelten Daten nicht aus, um einen Qualitätsunterschied festzustellen. Der t-Test berechnet einen p-Wert für die Frage, ob der Mittelwert der Bewertungsdifferenzen nicht Null ist. Ist dieser kleiner als 0,05, so können wir mit 95%-iger Sicherheit davon ausgehen, dass tatsächlich ein Qualitätsunterschied der Verfahren vorliegt (statistische Signifikanz). Ansonsten konnten wir nicht genügend Beweise sammeln, um die Nullhypothese zu widerlegen.

Ein t-Test für verbundene Stichproben ist exakt, wenn normalverteilte Bewertungsdifferenzen vorliegen. Aufgrund von in den Daten vorkommenden Ausreißern ist dies bei uns nur grob zutreffend. Allerdings nimmt mit zunehmender Stichprobengrösse die Robustheit des Tests gegenüber nicht normalverteilten Daten zu. Für die hier untersuchten Hypothesen stützen wir uns auf Stichproben mit jeweils mindestens 70 Werten, sodass der t-Test für unsere Daten zulässig sein sollte.

Als Alternative hatten wir in Betracht gezogen, den Wilcoxon-Vorzeichen-Rang-Test für die Bewertungsdifferenzen zu verwenden. Dieser erfordert nicht, dass diese normalverteilt sind, sondern vergleicht nur die relative Position der Elemente. Schließlich bleibt noch zu erwähnen, dass wir

durch die Varianzanalyse mit wiederholten Messungen (rmANOVA, Repeated measures Analysis of Variance) eine präzisere Auswertung für den direkten Vergleich von mehr als zwei Stichproben bekommen könnten. Da wir maximal drei Verfahren auf einmal gegeneinandergestellt haben, begnügen wir uns aber hier mit paarweise ausgeführten t-Tests für verbundene Stichproben.

5.3 Kategorie 1 - Analyse/Synthese

5.3.1 Fragestellung und Material

Grundsätzlich findet bei der Verwendung unserer Software als erster Schritt immer eine Analyse des Klangs (Zerlegung in Frames, Teiltöne, Noiseanteil, ...) statt, und bei der Wiedergabe eine Synthese (Überlappen von Frames, ...) wie in Kapitel 2 und Kapitel 3 beschrieben. In den Tests der Kategorie 1 werden diese beiden Schritte isoliert getestet. Als Testmaterial kommen Aufnahmen einzelner Noten von Instrumenten und die von der Software resynthetisierte Version zum Einsatz. Es findet kein Morphing statt. Ein Attack-Envelope wie in 2.4 beschrieben wird bei Analyse und Synthese verwendet (was zum Beispiel beim Flügel wesentlich zur Qualität beiträgt).

Typ	**Beschreibung**
Referenz	Originalaufnahme (vor der Analyse)
zu bewerten	Originalaufnahme (vor der Analyse)
zu bewerten	Resynthese (Aufnahme nach Analyse/Synthese)

Tabelle 5.2: Testmaterial Kategorie 1

Die Versuchsperson erhält in jedem Test dieser Kategorie drei Klangbeispiele, wie in Tabelle 5.2 angegeben. Das erste ist die Referenz, die als solche gekennzeichnet ist. Zu bewerten sind zwei weitere Klangbeispiele. Das eine Beispiel ist erneut die Referenz, das andere die Resynthese der Referenz. Da die zu bewertenden Klangbeispiele von der Bewertungssoftware in zufälliger Reihenfolge präsentiert werden, ist der Versuchsperson nicht klar, welches der beiden Beispiele das Original ist.

Dieser Aufbau entspricht dem ABC/HR Test [IR15a], der zur Bewertung kleiner Qualitätsunterschiede geeignet ist. Wir haben lediglich die dort vorgeschlagene Skala (von 5.0 bis 1.0) durch die MUSHRA Skala (von 0 bis 100) ersetzt, um die Versuchspersonen nicht mit der Benutzung verschiedener Skalen für Tests verschiedener Kategorien zu belasten.

Für jeden Test (ein Test entspricht hier einem Ton eines Instruments) gilt es zwei Klänge „A“ und „B“ zu bewerten, wobei zusätzlich als Referenz das Original vorhanden ist. Die Versuchsperson muss als erstes durch Hören herausfinden ob „A“ der Referenz entspricht oder „B“ der Referenz entspricht. Gelingt dies nicht, und beide Klangbeispiele klingen subjektiv genauso gut wie die Referenz, so soll die Versuchsperson beide auf der Skala mit 100 bewerten, da in dieser Situation beide so gut sind wie die Referenz. In diesem Fall würde die Analyse/Synthese subjektiv keinen Qualitätsverlust darstellen.

Kann jedoch einer der beiden Klänge als abweichend von der Referenz identifiziert werden, so ist dieser auf der Skala zu bewerten, wie gut er der Referenz entspricht. In diesem Fall würde die Analyse/Synthese einen Qualitätsverlust einführen, der durch die Bewertung quantifiziert wird.

Es sei an dieser Stelle noch erwähnt, dass es softwarebasierte Verfahren gibt, die die Frage beantworten, wie die subjektive Klangqualität eines Samples bezüglich einer Referenz ist, bekannt als

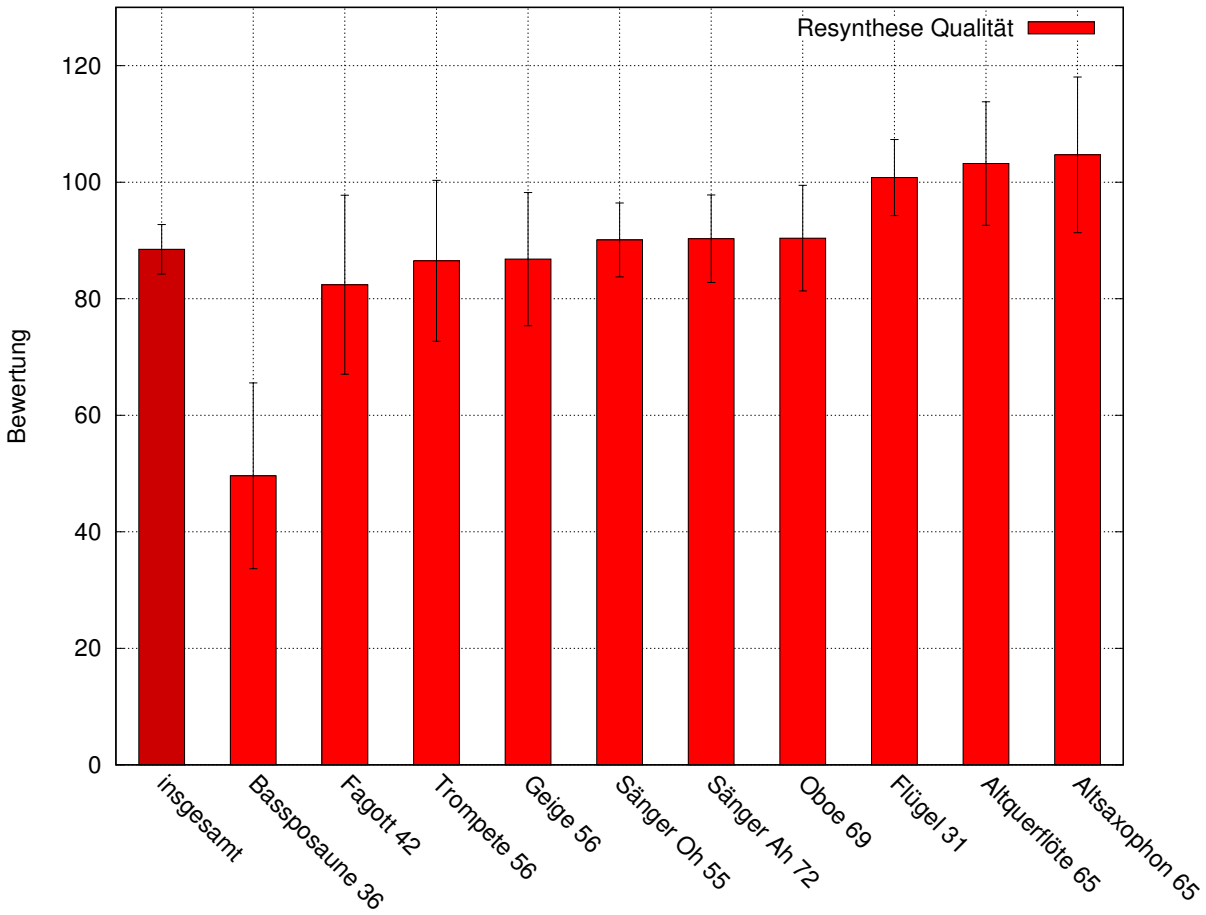


Abbildung 5.2: Qualität der Resynthese

PEAQ (Perceptual Evaluation of Audio Quality) Algorithmus [IR01]. Damit wäre eine Qualitätskontrolle auch für große Mengen von Testmaterial realisierbar. Eine Open Soure Implementation findet sich beispielsweise in GstPEAQ [HZ15].

5.3.2 Kommentare

Manche Versuchspersonen gaben außer den Bewertungen noch Kommentare ab, die wir hier zusammengefasst wiedergeben. Einige Versuchspersonen merkten an, dass die Resynthese der Bassposaune sehr deutlich schlechter als das Original war.

Eine Versuchsperson hätte gerne die Möglichkeit gehabt, Punkte über 100 zu vergeben, wenn die Resynthese besser klingt als das Original.

5.3.3 Auswertung

Generell gilt bei der Auswertung von ABC/HR Tests, dass die beiden Bewertungen nicht unabhängig sind, da die Versuchsperson weiß, dass immer mindestens eine der beiden mit 100 bewertet werden muss, da sie der Referenz entspricht. Es wird daher empfohlen für die statistische Betrachtung die Differenz der Werte zu verwenden. Wir addieren noch 100, damit die Differenzwerte wiederum auf der Skala liegen, sei also X_{ref} die Bewertung der Referenz und X_{syn} die Bewertung

der Resynthese, betrachten wir für die Auswertung nur den Wert

$$X_{diff} = 100 + X_{syn} - X_{ref}$$

Die Auswertung der Differenzwerte der insgesamt 10 Tests der Kategorie 1 findet sich in Abbildung 5.2. Zusätzlich zum Mittelwert ist das 95% Konfidenzintervall angegeben, welches die Genauigkeit der Schätzung des Mittelwerts beschreibt.

Am linken Rand gibt der etwas dunklere Balken die Gesamtwertung an, gemittelt über alle Versuchspersonen und alle Instrumente. Man sieht dass die Klangqualität insgesamt mit 88,5 bewertet wurde. Dies liegt im Bereich „ausgezeichnet“ auf unserer Bewertungsskala und bedeutet dass Klänge durch die Analyse/Synthese insgesamt betrachtet nur unwesentlich an Qualität verlieren.

Danach folgt die Auswertung geschlüsselt nach Einzelinstrumenten.Hierbei ist das Instrument und die Tonhöhe als Midinote angegeben, tiefere Töne entsprechen tieferen Midinoten. Da die Tests in zufälliger Reihenfolge präsentiert wurden (also jede Versuchsperson eine andere Abfolge der Tests bewertet hat), haben wir die Instrumente hier nach Mittelwert sortiert.

Fast alle Instrumente liegen im Bereich von 80 bis 100, in drei extremen Fällen (Flügel, Altquerflöte und Altsaxophon) wurden Mittelwerte über 100 erreicht. Eine Differenzwertung kann dann über 100 liegen, wenn die Resynthese besser bewertet wurde als das Original ($X_{syn} > X_{ref}$). Bei solchen Bewertungen muss man dann annehmen, dass Original und Resynthese sehr sehr ähnlich sind, da bei einem klaren Qualitätsunterschied jede Versuchsperson die Referenz erkennen können sollte.

Eine Ausnahme zu den ansonsten durchweg guten Ergebnissen findet sich bei der Bassposaune, die nur mit 49,6 („ordentlich“) bewertet wurde. Eine genauere Betrachtung der Situation legt nahe, dass die verwendete Aufnahme der Bassposaune nicht nur Teiltöne in der Nähe der Vielfachen der Grundfrequenz $G, 2G, 3G, ...$ enthält, sondern auch einige Peaks zwischen diesen Frequenzen auftreten. Dadurch wird die minimale Peakbreite wie im Kapitel 2.3.3 nicht erreicht, da die Peaks eng zusammenliegen. Dies wiederum führt dazu dass die Peaks nicht als Sinusanteil modelliert werden, sondern als Noiseanteil. Dies führt zu einem deutlichen Qualitätsunterschied zwischen Original und Resynthese.

Würde man die minimale Peakbreite generell reduzieren, könnte man zwar diesen speziellen Fall besser behandeln, aber die Anzahl der detektierten Peaks in allen anderen Fällen würde sich auch erhöhen, was wiederum einen negativen Einfluss auf Performance und Speicherbedarf hätte. Es bliebe daher entweder die Möglichkeit, Analyseparameter pro Instrument oder Note einzustellen, oder ein anderes Peak-Kriterium zu entwickeln, welches auf dem gesamten Material funktioniert.

5.4 Kategorie 2 - Morphing von Einzelnoten

5.4.1 Fragestellung und Material

In Kapitel 4 haben wir drei Verfahren für das Morphing von Klängen entwickelt. Die beiden einfacheren Verfahren sind das lineare Morphing, bei dem die Amplituden der Teiltöne linear kombiniert werden und dB-lineares Morphing, bei dem die Amplituden der Teiltöne auf einer dB-Skala linear kombiniert werden. Weiterhin wurde das LPC Morphing erklärt, welches die spektralen Hüllkurven der Teiltöne mittels LPC/LSF Koeffizienten berechnet.

In diesem Test erstellen wir mit den drei Morphingverfahren Übergänge zwischen Einzelnoten, also beispielsweise den Übergang von einem Oboen- in einen Geigenton. Diese sind immer sieben Sekunden lang, wobei erst eine Sekunde des Ausgangstons gespielt wird, fünf Sekunden lang das Morphing stattfindet, und schließlich eine Sekunde lang der Zielton zu hören ist. Das Material wur-

de Lautstärkenormiert und mit Loops versehen, da die Originalsamples nicht lang genug gewesen wären.

Typ	**Beschreibung**
Referenz	Instrument 1 alleine
Referenz	Instrument 2 alleine
zu bewerten	lineares Morphing von Instrument 1 zu Instrument 2
zu bewerten	dB-lineares Morphing von Instrument 1 zu Instrument 2
zu bewerten	LPC Morphing von Instrument 1 zu Instrument 2

Tabelle 5.3: Testmaterial Kategorie 2

Jeder Test besteht hier aus fünf Klangbeispielen, wie in Tabelle 5.3 angegeben. Wie zuvor werden die beiden Referenzen als solche gekennzeichnet, sie ermöglichen der Versuchsperson die Instrumente einzeln zu hören. Dies hilft zu unterscheiden, ob Qualitätsprobleme bereits im Ausgangsmaterial enthalten sind, oder durch das Morphing verursacht wurden.

Die drei Übergänge erscheinen wiederum als „A“, „B“ und „C“ in zufälliger Reihenfolge, sodass die Versuchsperson nicht weiß, welches Verfahren zu bewerten ist, aber direkt zwischen allen Varianten vergleichen kann.

Die Versuchspersonen wurden instruiert, als höchste Wertung 99 zu vergeben. Diese Regel wurde deshalb eingeführt, weil die Voreinstellung in der Software 100 ist, und es leicht möglich wäre, dass ein einzelner Übergang aus Versehen gar nicht bewertet würde. So kann in einem solchen Fall die Versuchsperson erinnert werden, dass ein Hörbeispiel nicht bewertet wurde, wenn eine Bewertung 100 ist.

Insgesamt wurde jede Versuchsperson gebeten zu bewerten, wie natürlich der Übergang zwischen den Instrumenten klingt.

5.4.2 Kommentare

Eine Versuchsperson sagte, dass durch den Test das Morphing immer nur in einem Übergang zu hören sei, und dass es schön gewesen wäre, direkt einen Hybridklang zu bewerten. Wir hatten dies beim Testdesign erwogen, aber darauf verzichtet, um die Gesamtmenge an Material nicht zu groß werden zu lassen.

Zwei Versuchspersonen empfanden die geloopen Gesangsaufnahmen als unnatürlich und störend.

5.4.3 Auswertung

Die Auswertung des Morphings von Einzelnoten ist in Abbildung 5.3 dargestellt. Wie zuvor ist am linken Rand in den dunkleren Balken die gemittelte Wertung der drei Verfahren über alle Instrumentenkombinationen angegeben. Danach folgt die Auswertung nach Instrumentkombination mit der Tonhöhe als Midinote. Diese wurden wieder nach Qualität sortiert, da sie ohnehin in zufälliger Reihenfolge präsentiert wurden (die Wertungen der drei Morphingverfahren wurden zur Sortierung gemittelt). Zusätzlich zum Mittelwert ist jeweils das 95% Konfidenzintervall angegeben.

In der Gesamtwertung liegen die drei Verfahren sehr nah zusammen, am besten wurde das LPC-

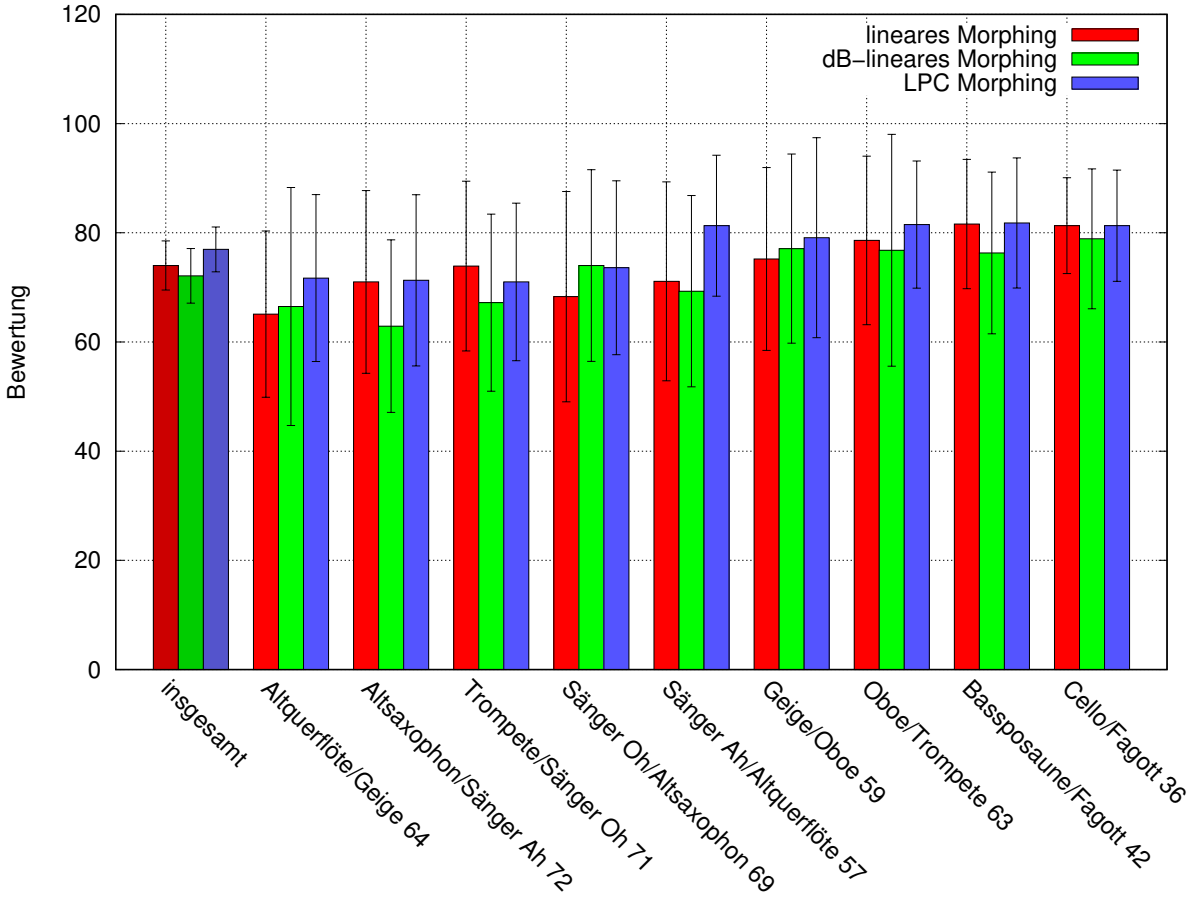


Abbildung 5.3: Qualität des Morphings von Einzelnoten

Morphing mit 77 gemittelt, das lineare Morphing wurde im Schnitt mit 74 bewertet, am schlechtesten schnitt das dB-lineare Morphing mit 72,1 ab. Es bleibt zu klären, ob wir nun aus dem Hörversuch schließen können, dass das LPC-Verfahren besser ist als die beiden anderen Morphing Methoden.

Die Konfidenzintervalle erlauben uns nicht direkt zu schließen, dass eines der Verfahren besser wäre als ein anderes, da sie paarweise überlappen. Wir verwenden den zuvor eingeführten t-Test für verbundene Stichproben.

Verfahren 1	**Verfahren 2**	**p-Wert**
dB-linear	LPC	0,016
linear	LPC	0,089
dB-linear	linear	0,195

Tabelle 5.4: t-Test: Morphing von Einzelnoten

Die Ergebnisse sind in Tabelle 5.4 dargestellt. In unserem Hörversuch ist das LPC Morphing statistisch signifikant besser als das dB-lineare Morphing, da der p-Wert unter 0,05 liegt. Die unterschiedliche Lage der Mittelwerte (77 und 72,1) ist also nicht zufällig entstanden, sondern entspricht einem tatsächlichen Qualitätsunterschied.

Alle anderen p-Werte liegen über 0,05, das lineare Morphing ist damit nicht klar einzuordnen. Es ist zwar wahrscheinlich qualitativ schlechter als das LPC Morphing und besser als das dB-lineare

Morphing, aber könnte aufgrund der p-Werte jeweils auch gleich gut sein.

5.5 Kategorie 3 - Crescendo von Einzelnoten

5.5.1 Fragestellung und Material

In diesem Test geht es nicht wie zuvor um das Morphing von Klängen verschiedener Instrumente, sondern um das Morphing von Klängen des gleichen Instruments. Dies ist ein praktischer Anwendungsfall, bei dem als Ausgangsmaterial einzelne Aufnahmen des Instruments, beispielsweise in leise, mittellaut und laut (also ppp, mf und fff) verwendet werden. Wir bewerten hier nur zwei Varianten von Crescendo, da der Klang von Decrescendo vergleichbare Qualität haben sollte.

Es wurde hierbei ein 3 Sekunden langes Crescendo-Beispiel und ein 10 Sekunden langes Crescendo-Beispiel verwendet, welches jeweils ein Morphing vom leisen über den mittleren zum lauten Ton darstellt. Die verschiedenen Längen wurden gewählt um dem Hörer die Möglichkeit zu geben, den Übergang in den unterschiedlichen Geschwindigkeiten wahrzunehmen und eine Bewertung zu finden. Das Ausgangsmaterial wurde mit Loops versehen, um die entsprechende Länge zu erhalten.

Typ	**Beschreibung**
Referenz	Instrument leise (ppp)
Referenz	Instrument mittellaut (mf)
Referenz	Instrument laut (fff)
zu bewerten	3 Sekunden Crescendo (ppp -> mf -> fff)
zu bewerten	10 Sekunden Crescendo (ppp -> mf -> fff)

Tabelle 5.5: Testmaterial Kategorie 3

Jeder Test ist so aufgebaut, wie in Tabelle 5.5 angegeben, es kommen meist drei Lautstärkeebenen, manchmal auch vier Lautstärkeebenen vor, je nach Instrument. Intern kam hierbei immer dB-lineares Morphing zum Einsatz, da nur für dieses das für diese Kategorie notwendige Morphing mehrerer Instrumente implementiert war. Die zu bewertenden Beispiele wurde nicht in zufälliger Reihenfolge präsentiert, da es klar zu hören ist, welches Crescendo kurz bzw. lang ist.

Die Versuchspersonen wurden instruiert, als höchste Wertung 99 zu vergeben, um sicherstellen zu können, dass alle Beispiele bewertet wurden (wie in Kategorie 2).

Insgesamt wurde jede Versuchsperson gebeten zu bewerten, wie natürlich der Übergang zwischen den Lautstärkeebenen klingt.

5.5.2 Kommentare

Vermutlich die wichtigste Kritik bezüglich der Natürlichkeit bezieht sich auf das Ende des Klangs, der nach dem Erreichen des Zielklangs schnell ausgeblendet wurde. Zwei Versuchspersonen bemerkten dazu, dass das für jedes Instrument typische Ausschwingen nicht vorhanden war, was störend auffiel. Eine andere ähnliche Bemerkung war, dass die Ziellautstärke bei mehreren Instrumenten nicht erreicht wurde. Dies ist zwar vermutlich technisch gesehen falsch, da jedes Instrument bis zur Ziellautstärke gemorpht wurde. Hätte man aber eine zusätzliche Sekunde am Anfang im

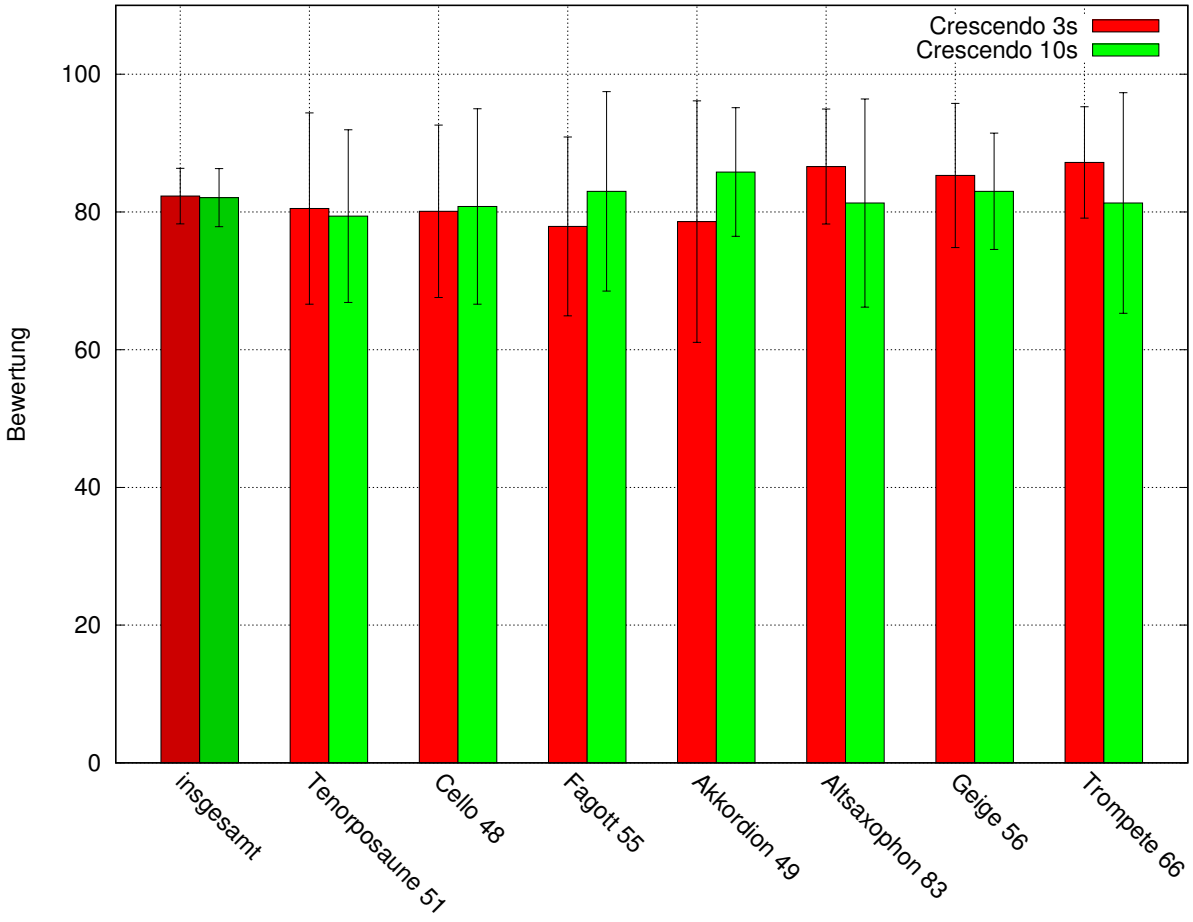


Abbildung 5.4: Qualität beim Crescendo von Einzelnoten

leisesten Sample und am Ende im lautesten Sample verbracht, wäre eher der subjektive Eindruck des Erreichens der Endlautstärke entstanden.

Eine Versuchsperson bemerkte, dass das Crescendo nicht gleichmäßig verläuft. Dies ist allerdings kein Fehler im Morphing, sondern wurde dadurch verursacht, dass schon das Ausgangsmaterial eine schwankende Lautstärke hatte.

Ein Teilnehmer empfand drei Sekunden als zu schnell. Ein weiterer Teilnehmer lobte besonders die Qualität des Akkordeons.

5.5.3 Auswertung

Die Ergebnisse dieser Kategorie sind in Abbildung 5.4 zu sehen. Links (etwas dunkler) ist der Mittelwert aller Tests angegeben. Rechts daneben sind die Ergebnisse für die einzelnen Instrumente (mit der Tonhöhe als Midi-Note) dargestellt. Die Tests wurden in zufälliger Reihenfolge präsentiert, daher haben wir sie für die Auswertung nach gemittelter Qualität sortiert.

Die Gesamtwertung betrug 82,3 für das 3 Sekunden Crescendo und 82,1 für das 10 Sekunden Crescendo. Damit lagen die Ergebnisse knapp in dem als „ausgezeichnet“ benannten Bereich der Bewertungsskala. Anhand der fast identischen Konfidenzintervalle sieht man, dass kaum einen Qualitätsunterschied zwischen den Beispielen vorliegt. Der Vollständigkeit halber haben wir den t-Test für verbundene Stichproben durchgeführt, der einen p-Wert von 0,902 ergibt, es gibt also (sehr deutlich) keinen statistisch signifikanten Unterschied zwischen den 3- und 10-Sekunden

Crescendos.

5.6 Kategorie 4 - Morphing einer Melodie

5.6.1 Fragestellung und Material

In der letzten Kategorie untersuchen wir, wie natürlich sich das Morphing zwischen zwei Instrumenten anhört, wenn es während einer Melodie stattfindet. Es gibt zwei Gründe, warum dies eine gute Ergänzung zum bisherigen Material darstellt. Bei Tests, die auf längeren Einzelnoten basieren, wird der Anfang des Tons kaum bewertet, sondern hauptsächlich der Teil nach der Einschwingphase. Daher ist es sinnvoll, eine Melodie zu präsentieren, sodass auch die Einschwingphase, besonders das Morphing der Einschwingphase von zwei Instrumenten, oft und gut zu hören ist, und in die Bewertung mit einfließt. Außerdem können so in einem einzelnen Test sehr viele verschiedene Noten der jeweiligen Instrumente präsentiert werden, wodurch der Test eine bessere Abdeckung des Materials erzielt.

Typ	**Beschreibung**
Referenz	Melodie nur mit Instrument 1
Referenz	Melodie nur mit Instrument 2
zu bewerten	Melodie, dB-lineares Morphing von Instrument 1 zu Instrument 2
zu bewerten	Melodie, LPC Morphing von Instrument 1 zu Instrument 2

Tabelle 5.6: Testmaterial Kategorie 4

In Tabelle 5.6 sind die Hörbeispiele aufgelistet. Es wird als Referenz die Melodie jeweils auf einem Instrument alleine präsentiert. Zu bewerten ist dann das dB-lineare Morphing und das LPC Morphing welches langsam zwischen beiden Instrumenten stattfindet. Wir haben hier auf das zusätzliche Bewerten des linearen Morphings verzichtet, da das dB-lineare Morphing und das lineare Morphing sehr ähnlich funktionieren, und ähnliche Resultate produzieren sollten.

Die zwei Übergänge erscheinen wiederum als „A“ und „B“ in zufälliger Reihenfolge, sodass die Versuchsperson nicht weiß, welches Verfahren zu bewerten ist, aber direkt zwischen beiden Varianten vergleichen kann.

Die Versuchspersonen wurden instruiert, als höchste Wertung 99 zu vergeben, um sicherstellen zu können, dass alle Beispiele bewertet wurden (wie in Kategorie 2 und Kategorie 3).

Insgesamt wurde jede Versuchsperson gebeten zu bewerten, wie natürlich der Übergang zwischen den Instrumenten klingt.

5.6.2 Kommentare

Eine Versuchsperson merkte an, dass eine langsame Melodie eventuell besser für das Bewerten des Morphings gewesen wäre. Allerdings wollten wir hier besonders die Einschwingphase in die Wertung einfließen lassen, sodass wir nach wie vor die gegebene (schnellere) Melodie für geeignet halten.

Ein weiterer Kommentar war, dass das Original (die Referenz) bereits künstlich klingt. Dies liegt

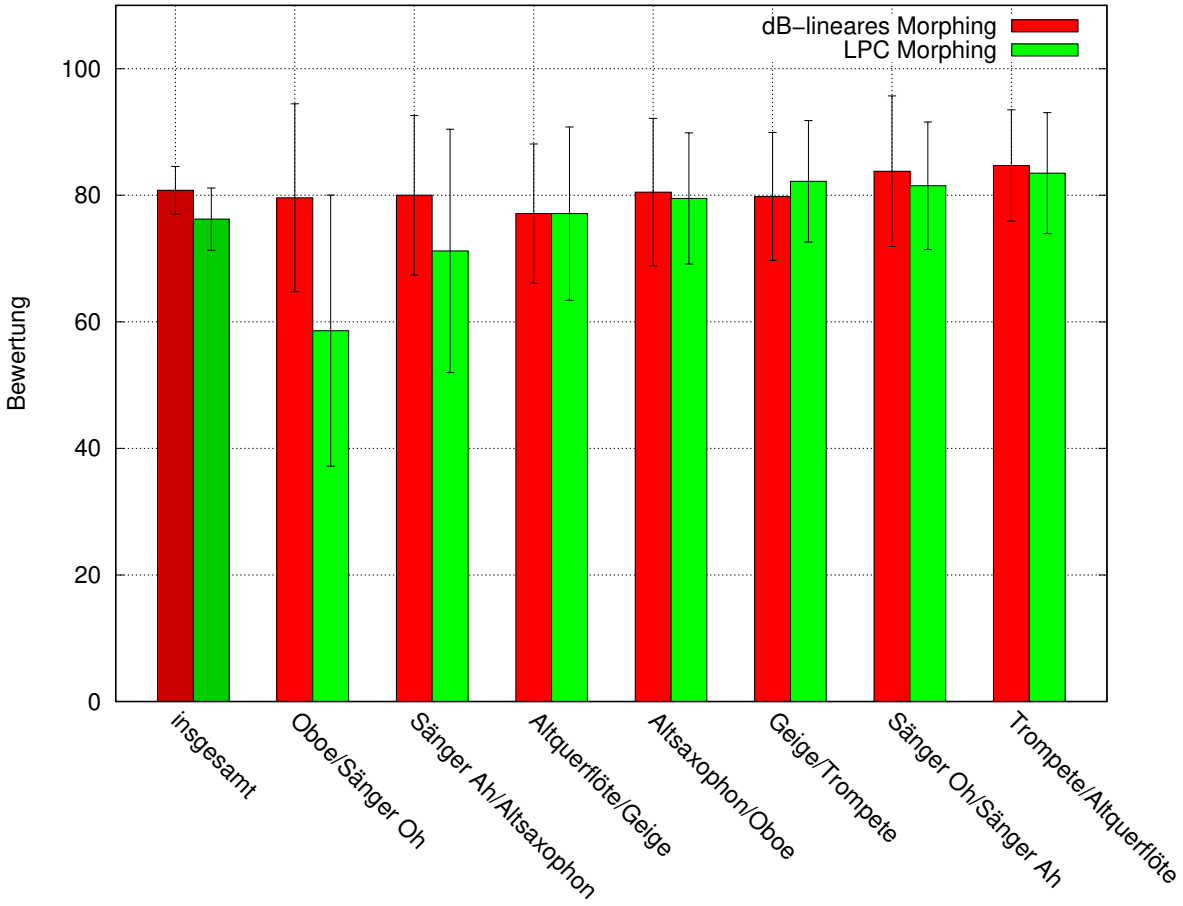


Abbildung 5.5: Qualität beim Morphing einer Melodie

daran, dass wir die Melodie aus Aufnahmen von Einzeltönen zusammengesetzt haben, und nicht als ganzes vom Originalinstrument aufgenommen haben.

Ein Testteilnehmer meinte, dass er bei 6 von 7 Beispielen gar keinen Qualitätsunterschied feststellen konnte.

5.6.3 Auswertung

Die Auswertung des Morphings von Melodien ist in Abbildung 5.5 dargestellt. Die dunkleren Balken links geben die Gesamtwertung an, rechts davon sind die einzelnen Instrumentkombinationen aufgelistet. Zusätzlich zu den Mittelwerten sind jeweils die 95% Konfidenzintervalle aufgetragen. Wie zuvor wurden die Einzeltests nach Qualität sortiert abgebildet.

Insgesamt erzielt das Melodiemorphing einen Mittelwert von 80,8 bei dB-linearem Morphing und einen Mittelwert von 76,2 bei LPC Morphing. Die Konfidenzintervalle überlappen, also verwenden wir wiederum den t-Test für verbundene Stichproben um zu sehen, ob es einen Qualitätsunterschied zwischen beiden Verfahren gibt. Der p-Wert ist hierbei 0,01, was bedeutet, dass das LPC Morphing statistisch signifikant schlechter als das dB-lineare Morphing abschneidet.

Zusätzlich dazu ist es interessant, dass es in dieser Kategorie zwei Beispiele gibt, wo sich die Mittelwerte der Bewertung sehr stark unterscheiden, und zwar bei der Kombination Sänger Oh/Oboe und bei der Kombination Sänger Ah/Altsaxophon. In diesen beiden Fällen schneidet das LPC Morphing deutlich schlechter als das dB-lineare Morphing ab.

5.7 Zusammenfassung der Ergebnisse

Kategorie	Verfahren	Mittelwert
1 - Analyse/Synthese	Resynthese	88,5
2 - Morphing Einzelnoten	dB-linear	72,1
	linear	74,0
	LPC	77,0
3 - Crescendo	3 Sekunden	82,3
	10 Sekunden	82,1
4 - Morphing Melodie	dB-linear	80,8
	LPC	76,2

Tabelle 5.7: Ergebnisse aus allen Kategorien

Die Mittelwerte der Bewertungen der einzelnen Kategorien sind in Tabelle 5.7 aufgelistet. Zunächst ist festzustellen, dass alle Bewertungen entweder im Bereich „gut“ $(60 - 80)$ oder sogar im Bereich „ausgezeichnet“ $(80 - 100)$ liegen. Die Analyse/Synthese wurde bis auf ein Beispiel (Bassposaune) sehr gut bewertet, für dieses Beispiel haben wir oben Verbesserungsideen skizziert.

Was den Vergleich der Morphingverfahren angeht, war ein Hauptanliegen, zu ermitteln, ob die Verwendung des LPC-Verfahrens (mit $K = 26$) eine Verbesserung ergibt. Wenn man es direkt mit dem dB-linearem Morphing vergleicht, ergab sich beim Morphing von Einzelnoten (Kategorie 2) ein signifikant besseres Ergebnis. Auf der anderen Seite klang es beim Morphing von Melodien (Kategorie 4) signifikant schlechter.

Insbesondere fiel auf, dass die Qualität bei zwei Melodiebeispielen sehr stark einbrach. Da eine gleichbleibende Qualität über alle Instrumente wünschenswert ist, würden wir daher die Verwendung des LPC-Morphings generell nicht empfehlen. Das LPC-Morphing benötigt zudem zusätzlichen Aufwand bei der Analyse und mehr Rechenzeit zur Berechnung der Teiltonlautstärken, der zumindest durch die vorliegenden Ergebnisse nicht gerechtfertigt ist. Nur falls sich durch genaue Untersuchung die Probleme beim Melodiemorphing beheben ließen, wäre das LPC-Morphing in allen Fällen zuverlässig einsetzbar.

Ein interessantes Ergebnis ist, dass in Kategorie 2 das lineare Morphing besser bewertet wurde als das dB-lineare Morphing. Von der Konstruktion der Verfahren hätten wir erwartet, dass das dB-lineare Morphing die Qualität gegenüber dem linearen Morphing verbessern würde. Allerdings ist der p-Wert mit 0,195 zu niedrig um zu sagen, ob das lineare Morphing tatsächlich qualitativ besser ist, oder ob es sich hier um einen zufälligen Effekt handelt. Generell ließe sich auch ein weiteres Morphingverfahren konstruieren, welches LPC Morphing mit linearer Amplitudenberechnung durchführt. Dieses müsste, wenn die bisherige Tendenz gleichbleibt, zumindest für Einzelnoten noch besser abschneiden, als das LPC Morphing.

6 Schlussfolgerungen

In dieser Arbeit konnten wir zeigen, wie sich ein Morphing der Klangfarbe verschiedener Musikinstrumente mittels Spektralmodellierung realisieren lässt. Dadurch, dass der Ansatz dabei auf der Analyse von Aufnahmen (Samples) basiert, entstehen realistische Modelle der zugrundeliegenden Klangfarben. Im Hörversuch konnten wir zeigen, dass die Fehler, die durch die Kombination von Analyse und Synthese entstehen, normalerweise sehr klein sind. Anders gesagt bedeutet dies, dass der Klang eines Instruments, beispielsweise eines Saxophontons, durch die bei der Analyse verwendete Zerlegung in Frames, deren Spektrum als Summe der Teiltöne und des Geräuschanteils repräsentiert wird, ein gutes Abbild des Originals darstellt.

Obwohl dies für Aufnahmen vieler natürlicher Musikinstrumente, beispielsweise Fagott, Trompete oder Saxophon gilt, hat diese Analysestrategie jedoch Grenzen. Im Hörversuch fiel bereits eine Aufnahme Bassposaune mit deutlich schlechterer Qualität auf. Generell gilt bei der von uns beschriebenen Suche nach lokalen Maxima im Spektrum aus Kapitel 2.3.3, dass ein lokales Maximum (Peak) eine gewisse Mindestbreite erreichen muss. Idealerweise hat jedes Kurzzeitspektrum Peaks bei der Grundfrequenz G und deren ganzzahligen Vielfachen ($2G$, $3G$, ...). In seltenen Fällen wird diese Regel bei normalen Musikinstrumenten verletzt, deren Resynthese klingt dann wie beim Beispiel der Bassposaune deutlich anders. Eine Verringerung der geforderten minimalen Peakbreite könnte in diesem Fall Abhilfe schaffen.

Anders sieht es bei synthetischen Klängen aus. Oft werden hier Klänge produziert bei denen die Peaks sehr nah aneinander liegen, beispielsweise bei der subtraktiven Synthese die Summe zweier leicht verstimmter Oszillatoren. Für solche Klangfarben liegen im Spektrum Peaks so dicht beieinander oder löschen sich gegenseitig teilweise aus, dass die Analyse der Spektren scheitert, was sich auch nicht durch die Anpassung der minimalen Peakbreite korrigieren lässt. Es bleibt also festzustellen, dass das von uns beschriebene Morphing zwar auf einer großen Klasse von Klangfarben funktioniert, vor allem von Musikinstrumenten, aber nicht allgemein genug ist, um sämtliche Klänge abzudecken.

Sind die Voraussetzungen für die Analyse erfüllt, kann das Morphing natürlich klingende Übergänge zwischen Klangfarben erstellen. Insgesamt haben wir drei Verfahren, lineares Morphing, dB-lineares Morphing und LPC-Morphing beschrieben. Im Hörversuch zeigte sich, dass die subjektive Qualität sich kaum unterscheidet. Das mathematisch aufwändigere LPC-Verfahren, welches sowohl mehr Rechenzeit als auch zusätzlichen Speicher benötigt, können wir jedoch aufgrund der Ergebnisse des Tests im allgemeinen nicht empfehlen, da dieses in einigen Fällen deutlich hörbare Verschlechterungen erzeugte. Die zusätzliche Komplexität des LPC-Verfahrens lohnt sich nicht, da kein entsprechend deutlicher Qualitätsgewinn des Morphings entsteht.

Viele unserer Bemühungen waren nicht nur auf eine generelle Beschreibung des Morphings gerichtet, sondern an den Anforderungen einer Echtzeitanwendung orientiert. Für die Analyse haben wir in Kapitel 2.5 eine Repräsentation der Modelldaten entwickelt, die kompakt ist, sodass weniger Hauptspeicher und Festplattenplatz benötigt wird. Die effiziente Umrechnung während der Synthese ist möglich. Detailliert wurden verschiedene Optimierungen der Synthese aufgezeigt. Durch die Berechnung der Syntheseframes im Frequenzbereich fällt die Generierung der einzelnen Teiltöne im Zeitbereich weg, und auch der Noiseanteil kann mit derselben inversen FFT erzeugt werden. Benchmarks zeigen (Kapitel 3.6), dass die so implementierte Synthese alleine sehr effizient ist und wie gewünscht für Echtzeitenwendungen geeignet ist.

Wird zusätzlich ein Morphing vorgenommen, kostet dies je nach Verfahren deutlich mehr Zeit als die Synthese alleine. Wir verweisen dazu auf die Benchmarks in Kapitel 4.5. Dennoch erreichen alle Morphingverfahren eine hohe theoretische Polyphonie. Auch das generell aufwändigere gitterbasierte Morphing, bei dem vier Klangfarben kombiniert werden, ist sinnvoll in Echtzeitanwendungen

einsetzbar.

Raum für Verbesserungen bleibt bei der Behandlung der Einschwingphase. Im quasistationären Zustand (nach dem Einschwingen) sind die Teiltöne einer Aufnahme eines Instruments normalerweise stabil und bleiben innerhalb eines Analyseframes etwa gleichlaut. Die Kurzzeit-Spektren ergeben in diesem Fall ein realistisches Bild der Klangfarbe, sodass das Morphing dementsprechend genau ist.

Probleme treten beispielsweise bei Klavieraufnahmen auf, die einen harten Anschlag am Anfang des Samples haben. Um zu vermeiden, dass dieser durch die Analyse verschmiert wird, hatten wir in Kapitel 2.4 die Berechnung zusätzlicher Modellparameter für den Anfang und das Ende des Attacks beschrieben. Werden diese bei der Synthese berücksichtigt, konnte eine deutliche Qualitätsverbesserung erzielt werden. Es blieb uns hier nicht genug Zeit, um unsere Implementation des Morphings so zu erweitern, dass diese Parameter verwendet werden, und dies zu testen. Daher bleibt uns hier nur auf die in Kapitel 4.2.8 gegebenen Ideen zu verweisen, wie die Integration möglich wäre.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[All77] Jont B. Allen. Short term spectral analysis, synthesis, and modification by discrete fourier transform. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-25(3):235–238, 1977.

[Ecu96] Pierre L' Ecuyer. Maximally equidistributed combined tausworthe generators. *Mathematics of Computation*, 65(213):203–213, 1996.

[FG66] J. L. Flanagan and R. M. Golden. Phase vocoder. *Bell Systems Technical Journal*, 45(9):1493–1509, 1966.

[FJ05] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005.

[Ger03] David Gerhard. Pitch extraction and fundamental frequency: History and current techniques. Technical Report TR-CS 2003-06, University of Regina, Canada, 2003.

[Gre75] John M. Grey. An exploration of music timbre. Dissertation STAN-M-2, Center for Computer Research in Music and Acoustics, Stanford University, 1975.

[Har78] Frederic J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[HZ15] Martin Holters and Udo Zölzer. GstPEAQ - an open source implementation of the PEAQ algorithm. In *Proc. of the 18th Int. Conference on Digital Audio Effects (DAFx-15)*, 2015.

[IR01] ITU-R. Recommendation ITU-R BS.1387-1, Method for objective measurements of perceived audio quality, 2001.

[IR15a] ITU-R. Recommendation ITU-R BS.1116-3, Methods for the subjective assessment of small impairments in audio systems, 2015.

[IR15b] ITU-R. Recommendation ITU-R BS.1534-3, Method for the subjective assessment of intermediate quality level of audio system, 2015.

[KR86] Peter Kabal and Ravi Prakash Ramachandran. The computation of line spectral frequencies using chebyshev polynomials. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 34(6):1419–1426, 1986.

[Moo73] James A. Moorer. The hetrodyne filter as a tool for analysis of transient waveforms. Technical Report STAN-CS-73-379, Computer Science Department, Stanford University, 1973.

[Moo78] James A. Moorer. The use of the phase vocoder in computer music applications. *Journal of the Audio Engineering Society*, 26(1/2):42–45, 1978.

[MQ84] Robert J. McAulay and Thomas F. Quatieri. Magnitude-only reconstruction using a sinusoidal speech model. In *Proceedings IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 27.6.1–27.6.4, 1984.

[MQ86] Robert J. McAulay and Thomas F. Quatieri. Speech analysis/synthesis based on a sinusoidal representation. *IEEE Transactions on Acoustics, Speech and Signal Processing*,

ASSP-34(4):744–754, 1986.

[Nei15] Melissa E. O' Neill. PCG: A family of simple fast space-efficient statistically good algorithms for random number generation. `http://www.pcg-random.org/paper.html`, 2015. Aufgerufen am 8. Mai 2016.

[PK95] K.K. Paliwal and W.B. Kleijn. Quantization of LPC parameters. *Speech Coding and Synthesis*, pages 433–466, 1995.

[Por76] Michael R. Portnoff. Implementation of the digital phase vocoder using the fast fourier transform. *IEEE Transactions on Acoustics, Speech and Signal Processing*, ASSP-24(3):243–248, 1976.

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[RD92] X. Rodet and P. Depalle. Spectral envelopes and inverse FFT synthesis. In *Audio Engineering Society Convention 93*, 1992.

[Ser89] Xavier Serra. A system for sound analysis/transformation/synthesis based on a deterministic plus stochastic decomposition. Dissertation STAN-M-58, Center for Computer Research in Music and Acoustics, Stanford University, 1989.

[SS87] Julius O. Smith and Xavier Serra. PARSHL: An analysis/synthesis program for non-harmonic sounds based on a sinusoidal representation. In *Proceedings of the International Computer Music Conference*, pages 290–297, 1987.

[SS90] Xavier Serra and Julius O. Smith. Spectral modeling synthesis: A sound analysis/synthesis system based on a deterministic plus stochastic decomposition. *Computer Music Journal*, 14(4):12–24, 1990.

[Str80] John Strawn. Approximation and syntactic analysis of amplitude and frequency functions for digital sound synthesis. In *Proceedings of the International Computer Music Conference*, pages 116–137, 1980.

[ZSL+98] Fang Zheng, Zhanjiang Song, Ling Li, Wenjian Yu, Fengzhou Zheng, and Wenhu Wu. The distance measure for line spectrum pairs applied to speech recognition. In *International Conference on Spoken Language Processing*, volume 3, page 1123, 1998.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den _____ Unterschrift: _____