Audio gesteuerte Animationen

Stefan Bindreiter

DIPLOMARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Digitale Medien

in Hagenberg

im Juni 2006

Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen entnommenen Stellen als solche gekennzeichnet habe.

Hagenberg, am 22. Juni 2006

Stefan Bindreiter

Inhaltsverzeichnis

Vorwort

An dieser Stelle möchte ich zu allererst meinem Betreuer Dipl.-Ing. Volker Christian danken, der beinahe rund um die Uhr für meine Fragen und Anliegen zur Verfügung gestanden ist. Ebenso möchte ich mich bei Dipl.-Ing. Dietmar Offenhuber bedanken, der bei der Idee zur Diplomarbeit maßgeblich beteiligt war. Nicht zuletzt gilt der Dank meiner Familie und meinen Freunden, die mich durch Höhen und Tiefen der Studienzeit begleitet haben.

Die Idee für das Thema der Diplomarbeit entstand aus der mangelnden Softwareunterstützung bei Animationen, die direkt auf Ton und Musik abgestimmt werden sollen. Leider bieten die gängigen Animationsprogramme in dieser Hinsicht kaum Möglichkeiten, die Animationen direkt mit dem Ton zu verbinden. Aus diesem Grund entstand der Plan ein Audioanalyse-Plugin für ein 3D-Programm zu entwickeln, welches das direkte Steuern von Objekten durch Audiosignale ermöglicht.

Kurzfassung

Diese Arbeit beschreibt unter anderem die Erstellung eines Audioanalyse-Plugins für die 3D-Software *Maya*. Neben einem Überblick über verschiedene Programme zur Visualisierung von Audiodaten werden auch die notwendigen Grundlagen der digitalen Signalverarbeitung behandelt. Ebenso werden Signaleigenschaften besprochen, die sich aus der zeitlichen und aus der spektralen Signaldarstellung ermitteln lassen. Neben den erwähnten Audioeigenschaften werden auch einige Beaterkennungsalgorithmen beschrieben.
Das Audioanalyse-Plugin verwendet eine Audiolibrary um die unterschiedlichen Dateiformate zu bewältigen und die eigentliche Analyse durchführen zu können. Das Plugin kann nahtlos in Maya eingebunden werden und stellt die Analysedaten beliebiger Audiodateien in der Szene zur Verfügung, sodass der Benutzer/die Benutzerin mit nur wenigen Mausklicks audiogesteuerte Animationen erstellen kann.

Abstract

The present work describes the implementation of a plugin for the 3D-software *Maya*, which analyzes audiofiles and allows the user to create audio-driven animations based on the data detected. In addition to a short overview of different kinds of software for audio visualization, the essential principles of digital signal processing are discussed. Furthermore, this thesis contains a discussion of time-based and frequency-based audio features as well as a description of different beat detection algorithms.
To cope with all the different file formats and to perform the actual analysis, an audiolibrary is used. The plugin blends seamlessly into Maya's open structure and makes the analysis data of any audio file available to Maya users. Everyone who knows how to handle Maya is able to create audiodriven animations with just a few mouseclicks.

Kapitel 1

Einleitung

3D-Programme bieten dem Anwender unzählige Möglichkeiten Animationen zu gestalten, Objekte zu modellieren und zu bewegen. Möchte der Anwender aber Bewegungen exakt mit einem Audiosignal synchronisieren, so ist dies meist mit einem enorm hohen Arbeitsaufwand verbunden. Meist müssen für jedes Einzelbild die Position der Objekte oder andere Einstellungen verändert werden. Hat man beispielsweise eine Gesichtsanimation, die sich zu einem gesprochenen Text verändern soll, so muss im einfachsten Fall für jedes Wort der Mund auf und zu gemacht werden. Könnte man die Steuerung direkt über ein Audiosignal vornehmen, so könnte man beispielsweise den Text als reines Sprachsignal rasch mit einem Mikrofon aufnehmen und anschließend die Mundbewegungen an die Audiodaten binden. Ist die Audioanalyse fortgeschrittener und verfügt über eine Formantenerkennung[1], so könnten für die Vokale *a, e, i, o, u* die dazupassenden Mundstellungen gespeichert werden und anhand der Analysedaten aufgerufen werden. Für Animationen von Gruppenbewegungen – Crowd Animationen, bei denen sich mehrere Objekte als Gruppe durch die Szene bewegen, werden meist Zufallswerte verwendet. Diese Zufallswerte lassen die Bewegungen der einzelnen Objekte individueller erscheinen. Da die wenigsten Animationen ohne Ton auskommen, würde es sich anbieten, anstelle der Zufallswerte Audiodaten zu verwenden. So kann eine Verbindung zwischen Ton und Bild hergestellt werden. Die Analysedaten könnten ebenso Farben, Lichter und Hintergrundobjekte, somit also die Stimmung der Szene automatisch beeinflussen.

1.1 Motivation

Bereits im Sommer 2005 entstand zwischen Dipl.-Ing. Dietmar Offenhuber und mir die Idee, ein Plugin für die 3D-Software *Maya* zu implementieren, welches dem Benutzer Audiodaten für die Steuerung von Animations-

[1] Die Formantenerkennung ist wichtig für die Spracherkennung und ermöglicht es, die Vokale zu unterscheiden.

elementen zur Verfügung stellt. Im Wintersemester 2005/06 entstand als Vorprojekt zur Diplomarbeit ein erster, einfacher Prototyp. Im Zuge dieser Arbeit wurde das Programm verbessert und erweitert. Diese Thematik bot die Chance, meine beiden Hauptinteressen – Animation und Audio, die während des gesamten Magisterstudiums meine Ausbildungsschwerpunkte bildeten – zu kombinieren.

1.2 Inhaltliche Gliederung

Diese Arbeit beschäftigt sich mit der Visualisierung von Audiosignalen sowie der dazu notwendigen Audioanalyse und stellt einen Prototyp in Form eines Maya-Plugins zur Verfügung. Neben einem Überblick über bestehende Programme und Komponenten zur Audiovisualisierung werden auch die nötigen Grundlagen zur Signalverarbeitung behandelt. Der Schwerpunkt dabei liegt allerdings bei den speziellen Signaleigenschaften. Die detaillierte Beschreibung der Implementierung des Prototypen soll als kurze Einführung in die Plugin-Programmierung für Maya dienen.

In Kapitel 2 wird ein kurzer Überblick der bestehenden Technik bei Visualisierungen gegeben und es werden mögliche Softwarelösungen beschrieben. Die Grundlagen der digitalen Signalverarbeitung, von zeitdiskreten Audiosignalen bis zur *Fourier Transformation*, werden in Kapitel 3 erläutert. Ein wichtiger Bestandteil dieser Arbeit – die Analyse der Eigenschaften eines Audiosignals – wird in Kapitel 4 behandelt. Neben Signaleigenschaften im Zeit- und Frequenzbereich werden auch kurz einige Algorithmen zur Beaterkennung beschrieben. Nach einer kurzen Einführung in die Struktur von Maya in Kapitel 5 wird die Implementierung des Prototypen auf Basis der gesammelten theoretischen Grundlagen in Kapitel 6 Schritt für Schritt beschrieben. Dies soll künftigen Maya-Plugin-Programmierern einen leichteren Einstieg ermöglichen.

Dieses Dokument enthält viele fachspezifische und englischsprachige Ausdrücke. Wenn diese nicht direkt im Text oder in Fußnoten erläutert werden, so sind sie im Glossar im Anhang alphabetisch aufgelistet. Weiters wird auf die doppelte Anführung geschlechtsspezifischer Ausdrücke zu Gunsten einer besseren Lesbarkeit verzichtet. So sind alle Ausdrücke wie beispielsweise „der Benutzer“ oder „der Programmierer“ als geschlechtsneutral zu betrachten.

Kapitel 2

Audiovisualisierungen

Die Bandbreite von Audiovisualisierungen reicht von einer einfacher Equalizer-Anzeige auf einem billigen Radio über Animationen und Videos bis hin zu aufwändigen Bühnenshows. In den wenigsten Fällen steckt ein komplexer Audioanalyse-Algorithmus dahinter. Meistens wird nur eine *Spektrumanalyse* durchgeführt oder fertige Abläufe und Einstellungen, die zuvor gespeichert wurden, abgespielt. In diesem Kapitel werden bestehende Softwarelösungen angeführt, zuvor aber noch eine kurze Übersicht über die verschiedenen Hardware-Komponenten gegeben, die in einer Visualisierung zum Einsatz kommen können.

2.1 Hardware

Obwohl in den meisten Anwendungsfällen (kleinere Theater, Discotheken, Museen, ...) die Lichtsteuerung mittels einer Software kontrolliert wird, so gibt es doch noch Anwendungen, bei denen herkömmliche Lichtpulte zum Einsatz kommen.

2.1.1 Steuerung

DMX Protokoll

Der Standard bei Lichtsteuerungen ist das DMX (**D**igital **M**ultiple**X**ed) Protokoll[1]. Es handelt sich dabei um ein digitales Multiplex-Steuersignal. Dieses erlaubt die störungsfreie Übertragung von bis zu 512 Lichtkreisen für Dimmer, Scanner, Motorik und vieles mehr über große Leitungslängen[2].

[1] DMX Standards: DMX-512/1990 (1990), DIN 56930-2 (2000). Die aktuellste Version des DMX Protokolls (USITT DMX512-A) wurde im Jahr 2004 durch die USITT (United States Institute for Theatre Technology) standardisiert.

[2] *Das kleine Lichtlexikon*: http://info.dimmer.de/dmx-512.htm
Wikipedia: http://de.wikipedia.org/wiki/DMX_%28Lichttechnik%29

Die ursprüngliche Verwendung, für die DMX konzipiert wurde, war die Ansteuerung von Lichtkreisen über Dimmer. Hierfür erschien die Anzahl von 512 Kanälen und die Auflösung von 8 Bit (256 Stufen [1],[3]) als ausreichend. Inzwischen werden jedoch praktisch sämtliche Geräte der Bühnen- und Effektbeleuchtung per DMX angesteuert. Besonders aufwändige Effekt-Scheinwerfer (z. B. „Moving-Head", „Scanner") benötigen zur Steuerung ihrer vielfältigen Funktionen mehrere Kanäle, darüberhinaus ist die Auflösung eines Kanals zu gering, um „glatte" Fahrten des Spiegels/des Scheinwerfers zu ermöglichen. Aus diesem Grund werden für die Panorama-Einstellungen (Pan) meist zwei Kanäle verwendet. Hinzu kommen noch Kanäle für die „Gobos"[3], Farbräder, usw. So kann es sein, dass ein Gerät bis zu 15 Kanäle benötigt. Die Anzahl der Kanäle betrifft auch direkt die Aktualisierungsrate, sodass versucht wird, nicht die gesamten 512 Kanäle auszuschöpfen. Werden mehr Kanäle benötigt, so werden einfach mehrere DMX-Anschlüsse (sogenannte „DMX-Universen") verwendet.

Es gibt derzeit noch keinerlei internationale Normung des DMX Protokolls. Die Lichtsteuerungs-Hersteller haben sich jedoch auf die Standards der USITT geeinigt, welche garantieren sollen, dass die Komponenten der verschiedenen Marken auch einwandfrei miteinander kommunizieren können.

Lichtpult

Es gibt unterschiedliche Arten von Lichtpulten[4]: „Preset-Pulte" sind einfache (meist noch analoge) Lichtpulte, die nur einige wenige Beleuchtungseinstellungen speichern können und diese unter Umständen mit einstellbarer Geschwindigkeit überblenden können. Bei Pulten mit Szenenspeicher ist oft schon ein Controller, also eine Recheneinheit nötig. Bei diesen Geräten werden meist schon DMX-Signale ausgegeben. Häufig können diese Pulte auch schon Sequenzen abspeichern. „Theater-Pulte" sind noch leistungsfähigere „Szene-Pulte" und können durch die Programmierung einer Cuelist[5], in der Einsatz- bzw. Startpunkte gespeichert werden, komplette Shows selbständig abwickeln. Die Steuerung von kopfbewegten Scheinwerfern und anderem „intelligentem" Licht ist allerdings mit diesen Pulten nicht möglich. Daher werden oft getrennte Pulte eingesetzt, wobei hier zuerst die zu steuernden Geräte vorprogrammiert werden, indem einzelne Spiegel- und Kopfpositionen zu Sequenzen zusammengefasst werden. Danach können einzelne oder mehrere dieser Sequenzen gleichzeitig abgefahren werden. In „Hybridpulten" sind die Funktionen von Theaterpulten und Pulten für „intelligentes" Licht zusammengefasst. Hier kommen dann meist Computer mit spezieller Hardware zum Einsatz, und die Pulte dienen nur mehr als Interface. Diese Pulte

[3] siehe Abschnitt 2.1.2

[4] *Wikipedia*: http://de.wikipedia.org/wiki/Lichtsteuerung

[5] Die Cuelist ist ein Liste, welche die Reihenfolge und Startpunkte von Ereignissen enthält.

bieten aufgrund des grossen Kanalumfangs häufig mehrere DMX-Universen zur Steuerung einiger tausend Kanäle und werden in großen Theater- oder Fernsehshows eingesetzt.

2.1.2 Ausgabegeräte

Projektoren

Die ältesten Projektoren sind Röhrenprojektoren, wobei für jede der drei Grundfarben eine Kathodenstrahlröhre zur Darstellung des Bildes verwendet wird. Diese Projektoren haben in der Regel auch getrennte Objektive, was die mobile Verwendung aufgrund der aufwändigen Justierung erschwert. Durch den Kathodenstrahl haben diese Geräte allerdings keine fixe Pixelauflösung, was besonders natürliche Bilder ergibt.

Mittlerweile haben sich aber zwei andere Technologien am Markt durchgesetzt: Einerseits gibt es die preisgünstigeren LCD-Projektoren (**L**iquid **C**rystal **D**isplay), bei denen über eine mit winzigen Transistoren behaftete Folie (*TFT* **T**hin **F**ilm **T**ransistor) die Ausrichtung – und somit auch die Lichtdurchlässigkeit – von Flüssigkristallen durch Anlegen unterschiedlicher Spannungen beeinflusst wird. Jedes Pixel besteht dabei aus einer Farbzelle, die wiederum für jede der drei Grundfarben einen Farbfilter enthält. Jeder dieser Filter wird von einem eigenen Transistor angesprochen. Diese Technologie steht in Konkurrenz zu den teureren DLP-Projektoren (**D**igital **L**ight **P**rocessing). Zur Bilderzeugung kommt ein DMD-Chip (**D**igital **M**icromirror **D**evice) zum Einsatz. Dabei handelt es sich um einen Chip, auf dem sich für jedes Pixel ein winziger, durch einen elektrischen Impuls kippbarer Spiegel befindet. Durch gezieltes Kippen der Spiegel wird das Licht entweder in die Projektionsoptik geleitet oder in einen Absorber abgelenkt. Durch schnelles Pulsieren der Spiegel können Helligkeitsstufen dargestellt werden. Ältere Chips verwenden ein schnell rotierendes Farbrad, wobei nacheinander alle drei Grundfarben projiziert werden. Dabei kann es bei kontrastreichen Übergängen zum „Regenbogeneffekt“ kommen. Bei besseren, neuen DLP-Projektoren werden für jede Grundfarbe separate DMD-Chips verwendet, was diesen Fehler beseitigt. Weitere Technologien, wie beispielsweise LED-, LCOS- und Laserprojektoren[6] sind noch nicht gänzlich ausgereift oder zu teuer.

Scheinwerfer

Die Palette reicht von günstigen Halogen-Strahlern über Stroboskope bis zu teuren Lasern oder Effekt-Scheinwerfern. Es werden mittlerweile aber auch immer häufiger LED-Elemente anstelle von Glühlampen als Lichtquelle eingesetzt:

[6] *Wikipedia*: http://de.wikipedia.org/wiki/Videoprojektor

„Moving-Heads“ oder kopfbewegte Scheinwerfer sind frei bewegliche Multifunktionsscheinwerfer aus der Theater- und Veranstaltungstechnik. Sie bestehen aus einem Sockel und einem schwenkbaren Arm, auf dem der bewegliche Kopf sitzt. Diese Scheinwerfer gibt es in allen erdenklichen Leistungsstärken und sie bieten eine hohe Bewegungsfreiheit, sind aber bei Richtungsänderungen durch die eigene Masse eingeschränkt. Typischerweise kommen Lampen mit hoher Lichtleistung und Farbtemperatur zum Einsatz. Auswechsel- und drehbare Musterschablonen – Gobos aus Metall oder Glas – sowie andere optische Zusatzfunktionen wie Iris, Prisma, Shutter (für Stroboskopeffekte) und Dimmer ermöglichen eine große Zahl von Effekten und Einsatzmöglichkeiten. Bei neueren Modellen werden teilweise auch LEDs oder Laser als Lichtquelle verwendet oder sogar kombiniert. In diesen Fällen dient der Laser als Effekt oder einfach als Ausrichtungshilfe.
Im Gegensatz zu kopfbewegten Scheinwerfern ist bei „Scannern“ die eigentliche Lichtquelle nicht beweglich, sondern fix montiert. Das Licht wird über einen beweglichen Spiegel gelenkt. Scanner bieten zwar nicht diese Bewegungsfreiheiten, sind dafür aber – da nur der kleine Spiegel bewegt werden muss – schneller. Auch Scanner können mit Gobos, Prismen, Shutter und Dimmer ausgestattet sein.

2.2 Software

Für (Audio-)Visualisierungen gibt es eine Unmenge von unterstützender Software. Downloadmöglichkeiten von einfacheren VJ Tools und ähnlicher Visualisierungssoftware für alle Plattformen, sowie aller nachfolgend beschriebenen Tools sind unter
http://www.audiovisualizers.com/toolshak/vjprgpix/softmain.htm zu finden.

2.2.1 VJ Tools

In Discotheken werden kommerzielle Programme wie z. B. *ArKaos*[7] verwendet. Im speziellen Fall der Discothek „Empire“[8] in Linz wird zur Steuerung der Lichtanlage ein *WholeHog PC*[9] mit entsprechender Software verwendet. Die beweglichen Bühnen- und Traversenelemente werden mit der Software *Chain Master*[10] gesteuert. Diese Softwarelösungen unterstützen alle das DMX-Protokoll und sind für Benutzer ohne Programmierkenntnisse geeignet.

[7] http://www.arkaos.net
[8] Information von Thomas Schober, Discothek Empire, Holzstr. 3, 4020 Linz
[9] http://www.flyingpig.com
[10] http://www.chainmaster.de

2.2.2 Graphische Programmierung

Die in diesem Abschnitt beschriebenen Programme ermöglichen graphisches Programmieren. Dabei können ganz einfach mit der Maus verschiedene Funktionsblöcke auf eine Arbeitsfläche gezogen werden. Durch setzen der Eingangs- und Ausgangsattribute sowie durch Verbinden der Blöcke kann ein Datenfluss erzeugt werden. Diese Programme eignen sich für schnelles Prototyping, bieten dennoch mit wenigen Mausklicks eindrucksvolle Ergebnisse. Ein grundlegendes Programmierverständnis ist aber trotzdem ein Vorteil.

MAX/MSP/Jitter

MAX/MSP/Jitter ist ursprünglich für den Mac entwickelt worden, steht aber mittlerweile auch für Windows zur Verfügung. MAX ist die Basis des Programms und enthält die Benutzeroberfläche, die Einstellungen, die Synchronisationsobjekte sowie die MIDI Funktionen. Auf dieser Basis wurden einige Erweiterungen entwickelt, wobei MSP ein Reihe von Funktionen zur Audiobe- und -verarbeitung bietet. Jitter ist ebenso eine Erweiterung von MAX und bietet Funktionen für die Echtzeitmanipulation von Video, 3D Grafik und anderen Daten.
http://www.cycling74.com/products/jitter

vvvv

vvvv ist ein freies Tool für Videosynthese in Echtzeit und im Funktionsumfang ähnlich zu MAX/MSP/Jitter. Sowohl Animationen als auch Video, Audio und andere Daten können verarbeitet werden. Der Focus liegt allerdings auf 3D-Animationen sowie der Interaktion mit externen Geräten. vvvv unterstützt die Kommunikation und Steuerung von Hardware über DMX, MIDI, RS232, TCP/IP, UDP und andere Formate. Momentan ist dieses Programm nur für Windows erhältlich.
http://vvvv.meso.net

Virtools und Eyesweb

Diese beiden grafischen Programmierumgebungen sind nicht speziell für Audiovisualisierungen implementiert worden, man kann sie allerdings für diese Zwecke verwenden. Virtools ist ein mächtiges Tool zur Informationsvisualisierung und Eyesweb bietet umfangreiche Funktionen zur Bildverarbeitung.
http://www.virtools.com und http://www.eyesweb.org

2.2.3 Visualisierungsplugins

Die meisten gängigen Media Player[11] bieten die Möglichkeit, während des Abspielens von Audiodateien auch Visualisierungen darzustellen. Neben Plugins, die der Hersteller der Software zur Verfügung stellt, können auch Anwender eigene Visualisierungsplugins schreiben und der Allgemeinheit zur Verfügung stellen.

Winamp

Winamp bietet eine eigene SDK[12] und ein eigenes Forum für jene Anwender, die eigene Plugins entwickeln wollen. Mit kurzen Tutorials wird man in die Pluginprogrammierung für Winamp eingeführt.

Windows Media Player

Ebenso können für den *Windows Media Player* eigene Visualisierungsplugins implementiert werden. Hier bietet Microsoft mit der *MSDN-Libary*[13] eine Vielzahl von Funktionen. Spezielle „Plugin-Wizards“[14] erleichtern das Erstellen dieser Visualisierungen.

VLC - Video Lan Client und foobar2000

Diese beiden Programme gewähren ihren Anwendern besonders tiefe Einblicke in den Quelltext und fordern dazu auf, bei der Implementierung des Programms – nicht nur auf Plugins beschränkt – selbst aktiv zu werden.

QuickTime Player und Real Player

In diesen beiden Fällen sind kaum bzw. keine Hinweise zu finden, dass die Anwender selbst (Visualisierungs-) Plugins implementieren dürfen.

Im Gegensatz zu den bisher beschriebenen Tools, werden für die Verwendung dieser SDKs und Librarys Programmierkenntnisse vorausgesetzt.

[11] Programme zur Wiedergabe von Multimediadateien. Eine Auflistung und Gegenüberstellung dieser Programme, sowie eine Liste der entsprechenden Download-Links, sind in http://en.wikipedia.org/wiki/Comparison_of_media_players zu finden.

[12] *Winamp*: http://www.winamp.com/nsdn/winamp/plugins

[13] *Visualisations for Windows Media Player*:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmplay%10/mmp_sdk/aboutplayercustomvisualizations.asp

[14] Ein „Wizard“ ist ein Hilfsprogramm, dass den Benutzer Schritt für Schritt durch einen Prozess hindurchführt.

2.2.4 Audiolibrarys

Audiolibrarys sind Programmierschnittstellen und bieten dem Programmierer eine Menge nützlicher Funktionen. Diese vereinfachen die Audioprogrammierung entscheidend, da sowohl die Behandlung verschiedenster Dateiformate als auch der Umgang mit Mehrkanalton usw. von der Schnittstelle übernommen werden. Der Anwender kann sich auf die „wesentlichen“ Elemente bei der Programmierung der eigenen Anwendung konzentrieren. Es folgt eine kurze Auflistung der gängigsten Schnittstellen. Diese Audiolibrarys stehen sowohl für Windows und MacOS als auch für Linux-Benutzer kostenlos zur Verfügung und bieten umfangreiche Funktionen für 2D- und 3D-Soundanwendungen. In allen Fällen sind auch die grundlegendsten Analysefunktionen wie die Spektrumanalyse oder Lautstärkenangaben der unterschiedlichen Kanäle implementiert.

- FMOD: http://www.fmod.org
- Open AL: http://www.openal.org
- BASS Audiolibrary: http://www.un4seen.com/bass

Diese Aufzählung ist nicht komplett, zeigt aber die bekanntesten Bibliotheken.

2.2.5 Audiounterstützung in 3D-Anwendungen

In den beiden 3D- und Animationsprogrammen *3D Studio MAX*[15] und *Maya*[16] ist die Audiounterstützung auf ein Minimum beschränkt. In beiden Fällen können zwar Audiodateien in die Szenen geladen werden, diese können aber nur bedingt zur Steuerung der Objekte verwendet werden. Ebenso werden nur wenige Dateitypen unterstützt. Beide Programme verfügen über einen großen Benutzerkreis und erlauben die individuelle Erweiterung der Anwendung durch Plugins. Es gibt in den diversen Foren nur wenige Hinweise auf Plugins, die sich im weitesten Sinne mit dem Thema Audio beschäftigen. Darunter sind aber hauptsächlich Plugins, die mehrere Dateiformate unterstützen. Erst jenes Maya-Plugin, welches im Kapitel 6 dieser Arbeit beschrieben wird, ermöglicht die Steuerung von Objekten durch Audiosignale. Ein weiteres Maya-Plugin[17] verfügt über ähnliche Funktionen, verwendet allerdings eine andere Audiolibary.

[15] Entwickler: Autodesk (früher Discreet und Kinetix)

[16] Entwickler: Alias | Wavefront

[17] Eine Kopie des Auszugs aus der Forumsdiskussion im *Highend3D Computergrafik Forum* auf http://www.highend3d.com/boards/lofiversion/index.php/t213186.html ist auf dem beiliegenden Datenträger enthalten.

Kapitel 3

Audioanalyse – Digitale Signalverarbeitung

3.1 Digitale Signalverarbeitung

3.1.1 Zeitdiskrete Signale durch Analog/Digitalumsetzung und Quantisierung

Der Weg von einem analogen (zeit- und wertkontinuierlichen) Signal zu einem „computertauglichen“, digitalen (zeit- und wertdiskreten) Signal wird am einfachsten durch ein zweistufiges Modell, bestehend aus einem Abtaster und einem Amplitudenquantisierer, bewerkstelligt. Im ersten Schritt – dem Abtastvorgang – wird in regelmäßigen zeitlichen Abständen (im Abtastintervall T) Signalwerte entnommen. Diese Abtastwerte können als zeitdiskretes Signal $x[n]$ dargestellt werden. Dabei sind die Amplituden noch wertkontinuierlich. Das Signal ist aber nur mehr zu den diskreten Zeitpunkten n definiert. Erst im zweiten Schritt werden die Werte einem Raster zugeordnet. Am Beispiel einer Audio-CD hat dieser Raster eine Auflösung von 16 Bit, um die Amplitude des Signals darzustellen. In der Abbildung 3.1 sind die Auswirkungen der soeben beschriebenen Schritte auf das Signal dargestellt.

Bei der Abtastung ist die Wahl der Abtastrate ($\mathrm{f}_s = 1/T$) entscheidend. Bei exakt bandbegrenzten Signalen und bei der richtigen Wahl der Abtastrate ist eine exakte Wiederherstellung des ursprünglichen Analogsignals aus den Abtastwerten möglich. So besagt das *Nyquist-Shannon'sche-Abtasttheorem*[1] [2, S. 152]:

> Ein Signal, das keine Frequenzkomponenten mit einer Frequenz größer als f_g enthält, kann durch äquidistante Abtastwerte im Abstand $T \leq \frac{1}{2}\mathrm{f}_g$ vollständig repräsentiert werden.

[1] Nyquist, Harry, amerikanischer Physiker, geb. 1889 – †1976 und Shannon, Claude Elwood, amerikanischer Mathematiker, geb. 1916 – †2001

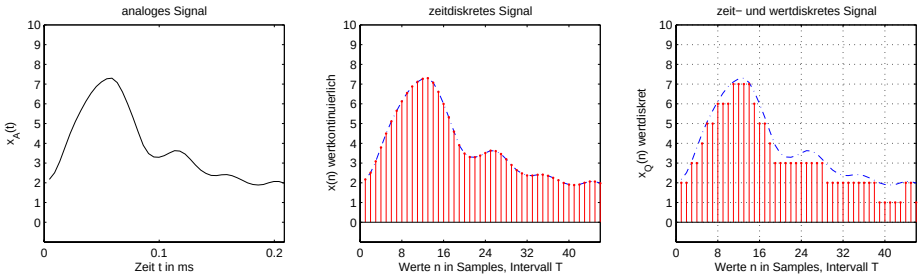


Abbildung 3.1: Bei der Abtastung analoger Signale wird ein zeit- und wertkontinuierliches Signal in ein zeit- und wertdiskretes Signal umgewandelt. In der linken Grafik ist ein analoges Sprachsignal $x_A(t)$ im Zeitbereich dargestellt. Die mittlere Grafik zeigt die Abtastwerte n, die im Abtastintervall T entnommen wurden ($x[n] = x_A(nT)$). In der rechten Grafik ist das wertdiskrete, also quantisierte Signal $x_Q[n]$ dargestellt.

Diese Aussage kann am einfachsten anhand eines sinusförmigen Signals überprüft werden, welches nur dann eindeutig konstruiert werden kann, wenn es mindesten dreimal pro Periode abgetastet wird. Dazu muss das Abtastintervall (T) kleiner als die halbe Sinusperiode ($T_0 = 1/\mathrm{f}_0$) sein, also $T < T_0/2$ oder $\mathrm{f}_s > 2\mathrm{f}_0$. Drei Abtastwerte pro Periode reichen somit nicht aus, da sonst der Fall eintreten kann, dass genau in den Nulldurchgängen abgetastet wird. Werden noch weniger Abtastwerte pro Periode verwendet, dann tritt bei der Rekonstruktion die *Aliasing-Komponente* auf, deren Frequenz durch $\mathrm{f'}_0 = \mathrm{f}_0 - \mathrm{f}_s/2$ gegeben ist. Durch diese zusätzlichen Frequenzkomponenten kann das Ausgangssignal massiv verändert werden. Ein einfaches Beispiel für *Aliasing* kann man Tag für Tag im Fernsehen, Kino oder auch in Tunnels (mit künstlichem Licht) betrachten, wenn man die Felgen eines (vorwärts) fahrenden Autos betrachtet. So entsteht manchmal der Eindruck, als würden sich die Reifen rückwärts drehen. Dieser Effekt tritt dann auf, wenn die Bildrate von Fernsehen und Film[2] oder bei künstlichem Licht die Netzfrequenz des Stromnetzes[3] kleiner ist als die Anzahl der Radumdrehungen. Detailliertere Beschreibungen zu diesem Phänomen sind beispielsweise in [11] oder [9] zu finden.

Bei der Amplitudenquantisierung werden die Amplituden der Abtastwerte einer bestimmten Wortlänge zugeordnet. Wie schon erwähnt sind bei Audiodaten 16 Bit, bei Bilddaten 8 Bit üblich. Beim Quantisierungsvorgang werden die Amplituden in Quantisierungsstufen unterteilt und alle Werte eines solchen Intervalls werden auf einen einzigen Ausgangswert abgebildet

[2] PAL 25 Bilder pro Sekunde, Film 24 Kader pro Sekunde

[3] Die Netzfrequenz beträgt in Europa üblicherweise 50 Hz, in Nordamerika 60 Hz.

(siehe Abbildung 3.1). Sind alle Intervalle gleich groß, dann spricht man von „linearstufiger“ Quantisierung. Ist das nicht der Fall, so spricht man von „nichtlinearstufiger“ Quantisierung. Das dabei enstehende Fehlersignal weist aufgrund der vielen Sprungstellen ein breitbandiges Frequenzspektrum auf und wird als *Quantisierungsrauschen* bezeichnet. Wie in [2, S. 156] begründet, kann das Quantisierungsrauschen unabhängig vom Ausgangsignal als *amplitudenmäßig gleichverteiltes, weißes Rauschsignal* beschrieben werden. Es kommt also beim Quantisieren zu einer Verschlechterung der Signaldynamik. Generell gilt (übernommen aus [2, S. 156]):

> Die Vergrößerung der Wortlänge um 1 Bit erhöht den *Signal-Rausch-Abstand* des quantisierten Signals um 6 dB.

Nichtlinearstufige Quantisierer eignen sich im Gegensatz zu linearstufigen besser für Signale, bei denen der Anteil der kleinen Signalamplituden überwiegt. So kann ein nichtlinearstufiger Quantisierer, der kleine Amplituden besser auflöst, eine wesentlich bessere Signaldynamik erzielen und mit geringer Bit-Auflösung einen größeren Signal-Rausch-Abstand erreichen.

3.1.2 Signalprozessoren

Ist das Signal nun in zeit- und wertdiskreter Form, so kann es mit einem Signalprozessor bearbeitet werden. Ein digitaler Signalprozessor (DSP) ist entweder eine spezielle CPU oder ein Algorithmus, welche(r) zur digitalen Signalverarbeitung eingesetzt wird. DSPs ersetzen nicht nur aufwändige analoge Filtertechnik, sondern können darüberhinaus Aufgaben ausführen, die analog nur schwer oder überhaupt nicht lösbar sind:

- Frequenzfilter hoher Ordnung
- Dynamikkompression und Rauschunterdrückung
- Implementierung von Effekten wie Echo, Hall, ...
- Datenkomprimierung zur digitalen Weiterverarbeitung
- ...

Die in Hardware implementierten DSPs können anhand ihres Befehlssatzes und der Fähigkeit, Rechenoperationen mittels

- Festkommaarithmetik oder mittels
- Gleitkommaarithmetik
- (oder mittels Blockgleitkommaarithmetik)

durchzuführen, unterschieden werden. DSPs mit Festkommaarithmetik sind im Aufbau meist einfacher und haben eine geringen Stromverbrauch. Dafür ist die Implementierung von bestimmten Algorithmen komplizierter, da bei jeder Berechnung vom Programmierer kontrolliert werden muss, ob es zu möglichen Überläufen in der Zahlendarstellung kommen kann und auf welcher Stelle sich der Kommapunkt befindet.

Signalprozessoren mit Gleitkommaarithmetik sind komplexer im Aufbau, da ihre Rechenwerke die kompliziertere Darstellung der Gleitkommazahlen verarbeiten können. Damit ist bei gleicher Rechenleistung meist ein höherer Stromverbrauch verbunden. Der Vorteil liegt in der meist einfachen Implementierung von komplizierten Algorithmen. In [2] wird dieses Thema vertiefend behandelt.

Reine Softwarelösungen von DSP-Applikationen, wie sie im Bereich der Multimediaanwendungen mit PCs (Modems, MP3-Encoder/Decoder, ...) vorkommen, stellen schon lange keine Probleme mehr dar. Da auch herkömmliche CPUs mit immer mehr DSP-Befehlen ausgestattet werden, sind schon Taktraten über 500 MHz für für die Stereosignalverarbeitung im Echtzeitbetrieb vollkommen ausreichend [2, Kap. 1]. Herkömmliche Audiolibrarys, wie z. B. *FMOD*[4], bieten dem Programmierer eine Menge vorgefertigter DSP-Funktionen und die Möglichkeit selbst Funktionen für die DSP-Einheit zu implementieren.

3.2 Darstellung zeitdiskreter Signale

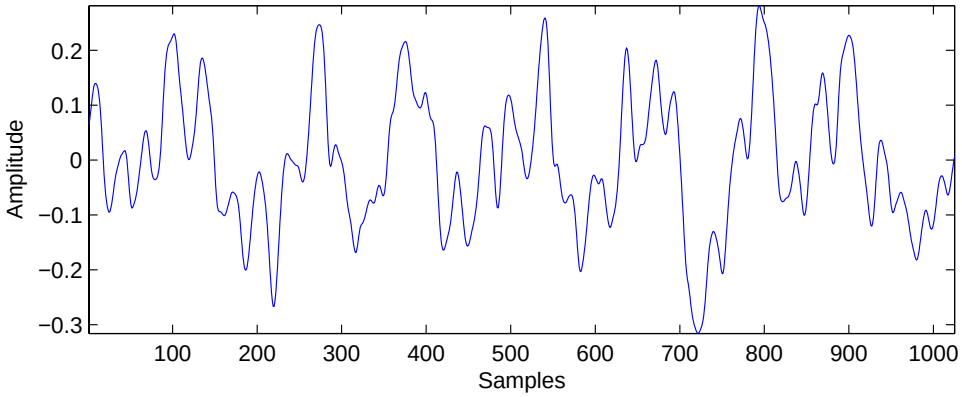


Abbildung 3.2: Ausschnitt eines Audiosignals von etwa 24 ms Länge. Die Audiodatei „audio2.wav“ ist auf dem beiliegenden Datenträger enthalten.

[4] http://www.fmod.org

Darstellung im Zeitbereich

Die Darstellung eines Audiosignals erfolgt im Zeitbereich als Wellenform (Waveform), wobei nur der Spannungsverlauf des Audiosignals über die Zeit zu erkennen ist. In den Abbildungen 3.4 und 3.2 sind Beispiele für die Darstellung im Zeitbereich gegeben. Zu jedem Zeitpunkt kann ein Spannungswert abgelesen werden. Diese Darstellung gibt aber keinerlei Auskunft darüber, aus welchen Komponenten sich das Signal zusammensetzt und wie die Energie im Signal verteilt ist. Aus den genannten Gründen ist die zeitliche Darstellung eines Signals nicht sehr aussagekräftig.

Darstellung periodischer Signale

Jede periodische Funktion lässt sich durch einfache Summierung von Cosinus- und Sinusschwingungen beliebig genau annähern. Diese Komponenten werden als *Fourier Reihe* bezeichnet. Zur Zerlegung eines periodischen Signals in seine Bestandteile wird die *Fourier Analyse* verwendet. In Abb. 3.3 sind die spektralen Komponenten des periodischen Signals aus Abb. 3.4 zu sehen. Die Fourier Analyse gilt nur für periodische Signale.

Darstellung im Frequenzbereich

Der Großteil der Information ist in der Frequenz, der Phase und Amplitude der spektralen Bestandteile des Signals enthalten. Um diese Information zu erhalten, ist es notwendig, das Frequenzspektrum des Signals zu ermitteln. Mit mathematischen Techniken wird das nichtperiodische Signal in Sinus- und Cosinuswellen zerlegt. Allgemein wird dieser Vorgang als *Fourier Transformation* bezeichnet.

3.2.1 Fourier Analyse

Überlagerung von Sinuswellen – Sinusoid Superposition

Jean Baptiste Fourier[5] entdeckte, dass jedes kontinuierliche periodische Signal als Summe ausgewählter Sinusschwingungen (Fourier Reihen) dargestellt werden kann. Eine Sinusschwingung ist eine mathematische Funktion, die einen einfachen harmonischen Oszillator beschreibt: Sei ω ein Winkel, der entlang der X-Achse und dem Bogen des Einheitskreises gegen den Uhrzeigersinn aufgespannt wird. Dann ist $\sin(\omega)$ die vertikale Koordinate des Punkts am Ende des Kreisbogens. Diese Schwingung ist periodisch mit einer Dauer von 2π und ω kann somit als 2πf dargestellt werden, was wiederum in der Sinusfunktion

$$y(t) = A\sin(2\pi ft + \phi) \tag{3.1}$$

[5]Fourier, Jean Baptiste, französischer Mathematiker und Physiker, geb. 1768 – † 1830

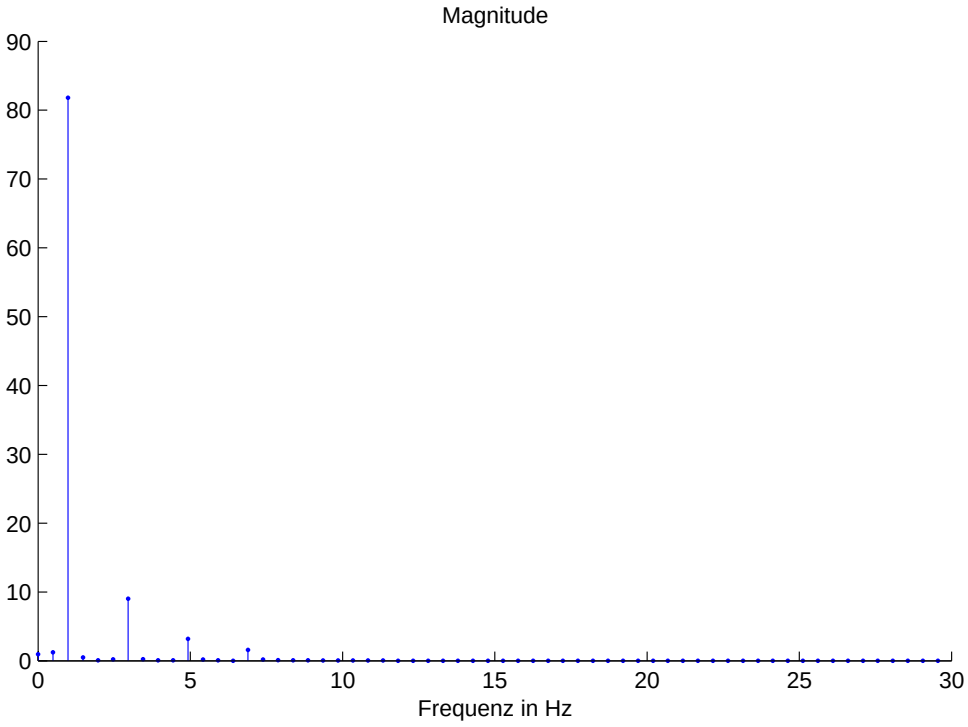


Abbildung 3.3: Fourier Analyse der Annäherung eines Dreieck-Signals (Abb. 3.4). 4 Spitzen, mit 1, 3, 5 und 7 Hz sind im Spektrum erkennbar, welche den Cosinus-Funktionen $cos1 = 0.81057 * \cos(2\pi * t)$, $cos2 = 0.09007 * \cos(2\pi * 3t)$, $cos3 = 0.03243 * \cos(2\pi * 5t)$ sowie $cos4 = 0.01654 * \cos(2\pi * 7t)$ entsprechen.

resultiert, wobei A die Amplitude der Welle, f die Frequenz des Signals und ϕ die Phase darstellt. So können alle – auch noch so komplizierten – Signale durch die Addition von einfachen Sinusfunktionen mit unterschiedlicher Frequenz, Amplitude und Phase dargestellt werden. Das gilt auch für alle Audiosignale, da diese sich durch Wellen ausbreiten. Bei der Erzeugung eines Signals durch Überlagerung von Sinuswellen, wie beispielsweise in Abbildung 3.4 zu sehen ist, spricht man von *Fourier Synthese*.

3.2.2 Fourier Transformation – FT

Um ein periodisches Signal nun in seine Bestandteile aufzuschlüsseln wird die Fourier Analyse verwendet. So kann festgestellt werden, welche Frequenzen im Signal vorkommen und wie stark jeder dieser spektralen Komponenten auf das Gesamtsignal wirkt. Um die Fourier Analyse für kontinuierliche

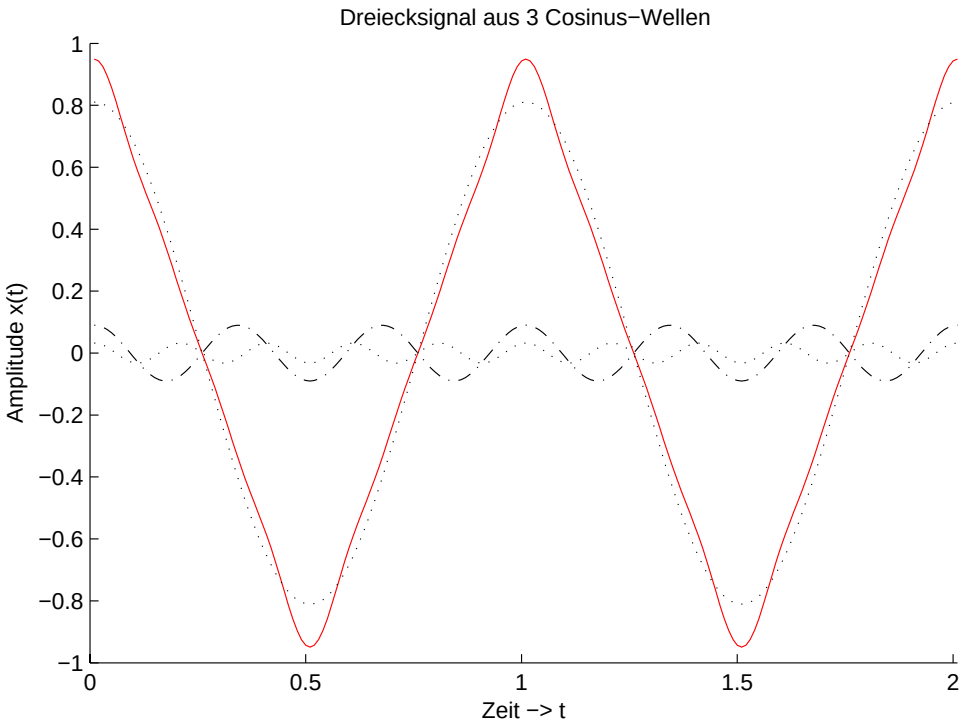


Abbildung 3.4: In dieser Grafik sind vier verschiedene Signale zu erkennen. Das periodische Dreieckssignal (rote Linie) entsteht, wenn man die drei Cosinus-Signale addiert. So kann jedes periodische Signal durch die Summe einfacher Sinus- oder Cosinusfunktionen dargestellt werden.

Signale durchzuführen wird die *Fourier Transformation* (3.2) verwendet:

$$X(f) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} x(t) e^{-i\omega t} dt \qquad \omega = 2\pi f \tag{3.2}$$

Die *inverse Fourier Transformation* (*IFT* 3.3) ermöglicht wiederum aus einem Spektrum den entsprechenden Zeitverlauf des Signals zu ermitteln.

$$x(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} X(f) e^{i\omega t} df \qquad \omega = 2\pi f \tag{3.3}$$

Im Allgemeinen ist die *Fourier-Transformierte* eine komplexwertige Funktion in ω. $X(e^{i\omega})$ wird manchmal auch in karthesischer Form

$$X(e^{i\omega}) = X_{\Re}(e^{i\omega}) + iX_{\Im}(e^{i\omega})$$

oder in polarer Form als

$$X(e^{i\omega}) = |X(e^{i\omega})| e^{i\angle X(e^{i\omega})}$$

angeschrieben. Die Größen $|X(e^{i\omega})|$ und $X(e^{i\omega})$ werden als „Amplituden"- bzw. „Phasengang" der Fourier-Transformierten bezeichnet. Diese wiederum wird meist als „Fourier-Spektrum" oder einfach nur als „Spektrum" bezeichnet. An Stelle des Amplitudengangs finden oftmals auch die Begriffe „Betragsspektrum" oder „Amplitudenspektrum" Verwendung. Der Phasengang wird hin und wieder auch als „Phasenspektrum" bezeichnet.

Wird die Fourier Transformation auf ein Signal angewendet, so wird es von der Darstellung im Zeitbereich in die Darstellung im Frequenzbereich transformiert. Es ist anzumerken, dass beides Möglichkeiten zur Darstellungs des selben Signals sind. Wird das Signal in einem der beiden Bereiche verändert, so wirkt sich diese Änderung natürlich auch im anderen Bereich aus. So entspricht beispielsweise eine Faltung im Zeitbereich einer Multiplikation im Frequenzbereich und umgekehrt.

Die Fourier Transformation gilt theoretisch für Signale unendlicher Länge. Es wird die spektrale Information des gesamten Signals ermittelt, jedoch auf die Veränderungen im Zeitbereich keinerlei Rücksicht genommen. Für periodische Signale, wie jene in den genannten Beispielen, ist diese Darstellung durchaus akzeptabel, für die Analyse nichtperiodische Signale (die meisten Audiosignale) bringt diese Darstellung allerdings nichts. Mit der *Short-Time Fourier Transformation (STF)* kann das Problem aber gelöst werden. Hier wird das Signal in kurze Abschnitte unterteilt und jeweils die FT ausgeführt. So können auch die Veränderungen, die über die Zeit geschehen berücksichtigt werden. Bei dieser Transformationsmethode entsteht dann allerdings ein neues Problem: Die FT erfordert ein unendlich langes Signal, dazu werden die Abschnitte geloopt. Durch das „Zerstückeln" des Signals entstehen aber an den Enden der Ausschnitte abrupte Kanten, welche bei der Analyse spektrale Komponenten hervorbringen, die im Originalsignal nicht vorhanden sind. Dieses Phänomen wird als *Leckeffekt* [9] bezeichnet. Wird der Ausschnitt zuvor mit einer *Fensterfunktion* multipliziert, die das Signal an den Enden auf Null skaliert, so kann der Effekt zwar nicht gänzlich verhindert werden, aber die falschen Frequenzen können mit der Wahl der richtigen Fensterfunktion [9, S. 537 ff] deutlich reduziert werden. Häufig verwendete Fensterfunktionen sind das *Hamming-* oder das *Hamming*-Fenster, welche einen guten Mittelweg zwischen Rechenzeit und Qualität bieten. Da durch die Multiplikation mit der Fensterfunktion an den Ausschnittsenden Information verloren geht, ist es üblich, die Ausschnitte überlappend zu wählen.

3.2.3 Diskrete Fourier Transformation – DFT

In der digitalen Signalverarbeitung, wo zeitdiskrete Signale verarbeitet werden, kommt die *Diskrete Fourier Transformation* (*DFT* 3.4) zum Einsatz. Sie ist das Äquivalent zur Fourier Transformation aus dem analogen Bereich.

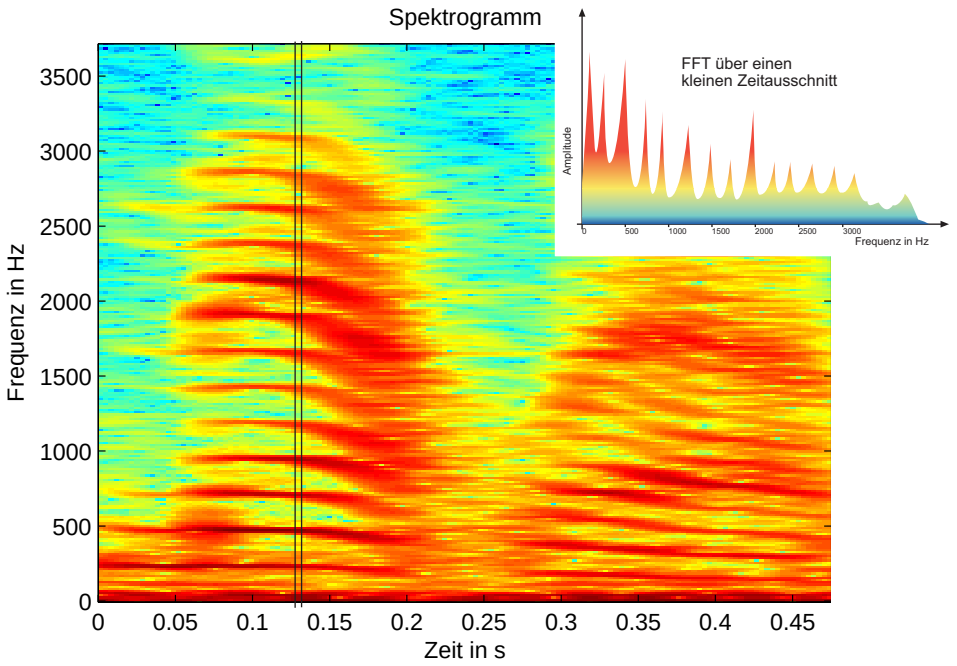


Abbildung 3.5: Das Spektrogramm ist eine grafische Darstellung des Spektrums im Zeitverlauf. Auf kleine Zeitausschnitte wird die FFT angewandt und die Ergebnisse in Blöcken nebeneinander dargestellt. Rote Bereiche weisen hohe Energiewerte auf, blaue Bereiche niedrige Energiewerte. Für die FFT wurde eine Fenstergröße N von 512 Werten festgelegt, was bei einer Abtastrate von 22050 Hz ein zeitliche Auflösung von ca. 43 FFT-Blöcken pro Sekunde ergibt. Die spektrale Auflösung liegt bei $\frac{N}{2}$, also bei 256 Werten.

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{-i2\pi \frac{kn}{N}} \qquad k = 0, \ldots, N-1 \tag{3.4}$$

Die *inverse Diskrete Fourier Transformation* lautet:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{i2\pi \frac{kn}{N}} \qquad n = 0, \ldots, N-1 \tag{3.5}$$

Für Blocklängen N , die sich als Potenz von 2 darstellen lassen, kann die Berechnung mit dem Algorithmus der *Fast Fourier Transformation (FFT)* erfolgen. Die Qualität der DFT/FFT, kann durch die Größe N des gewählten Fensters bestimmt werden. Je größer N , desto feiner die Analyse des Spektrums. In Abbildung 3.5 ist die Darstellung eines Sprachsignals zu erkennen,

bei der die FFT-Blöcke über die Zeit dargestellt sind. Diese Darstellungsform bezeichnet man als *Spektrogramm*. Dabei wird das Signal in kleinen zeitlichen Ausschnitte fourier-transformiert und die Resultate über die Zeit dargestellt. So kann man die spektralen Veränderungen gut erkennen.

3.3 Fensterfunktionen

Fensterfunktionen werden einerseits für den Entwurf digitaler Filter verwendet, andererseits um bei der Spektralanalyse den Leckeffekt zu minimieren. Beim Filterentwurf ist es im Sinne klarer Filtergrenzen erstrebenswert, möglichst steile Flanken zu erlangen. Wie im rechten Teil der Abbildung 3.6 zu erkennen ist, bietet das Rechteckfenster das schmalste Hauptmaximum, ist aber aufgrund relativ ungedämpften Nebenmaxima wenig geeignet. Es gilt, einen Kompromiss zwischen hoher Dämpfung der Nebenmaximas und eines möglichst schmalen Hauptmaximum zu finden. Häufig verwendete Fensterfunktionen (neben dem Rechteckfenster) sind das Hann oder Hanning-Fenster, weiters das Hamming-, Blackman, Bartlett- und Kaiser-Fenster. Abbildung 3.6 zeigt eine Auswahl der genannten Fenster im Zeit- und im Frequenzbereich.

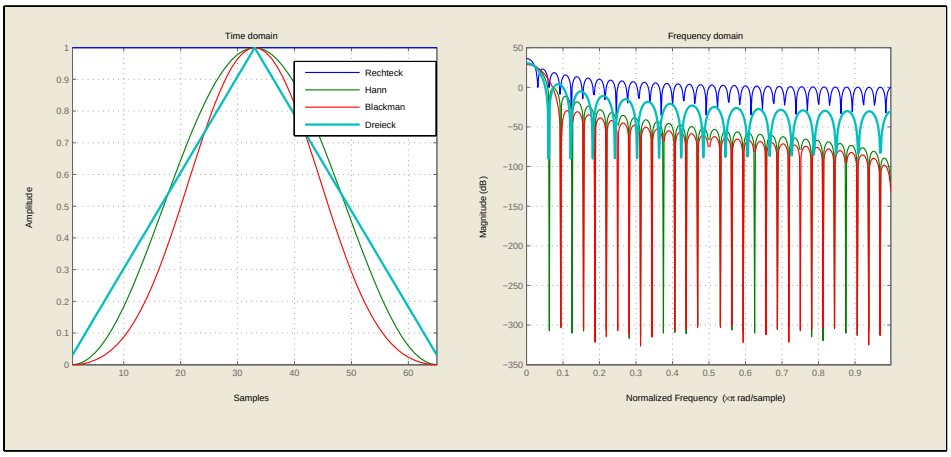


Abbildung 3.6: Fensterfunktionen im Zeit- und Frequenzbereich.

In [9, Kapitel 7 und 11] sind all diese Fensterfunktionen sowie ihre Anwendungen im Filterentwurf und der Spektralanalyse genauer beschrieben.

Kapitel 4

Audioanalyse – Eigenschaften, Features

Spricht man von Audioeigenschaften, so muss man zu Beginn zwei Begriffe unterscheiden. Physikalische Audioeigenschaften beziehen sich auf Messungen und Berechnungen, die direkt von einem Audiosignal ermittelt werden. Dazu gehören alle in diesem Kapitel beschriebenen Eigenschaften. Die zweite Gruppe – *perceptual Features* – sind subjektive Ausdrücke, welche die menschliche Wahrnehmung von Audiosignalen beschreiben. So können Eigenschaften wie *loudness, richness, pitch, timbre, rhythm, ...*, dieser Gruppe zugeordnet werden. Natürlich stehen all diese *perceptual Features* irgendwie in Verbindung zu den physikalischen Eigenschaften des Audiosignals, da die menschliche Wahrnehmung des Audiosignals über die physikalischen Schwingungen der Luft geschieht. Manche dieser Verbindungen scheinen relativ eindeutig, so kann die Amplitude des Signals mit der empfundenen Lautheit (loudness) in Verbindung gesetzt werden oder die Grundfrequenz (fundamental frequency) mit der empfundenen Tonhöhe (pitch). Dabei ist allerdings zu beachten, dass diese scheinbar offensichtlichen Verbindungen nicht immer korrekt sind. So hängt beispielsweise die empfundene Lautstärke sehr stark von der spektralen Energieverteilung des Signals ab und weniger von der Stärke der Amplitude. Viele dieser perceptual Features lassen sich mathematisch nur schwer beschreiben, da einige von ihnen komplizierte Kombinationen physikalischer Signaleigenschaften sind. Detailiertere Informationen zu perceptual Features und der menschlichen Wahrnehmung von Audiosignalen findet man beispielsweise in [12, Kap. 5] und in [5, Kap. 6 und 17].

Die physikalischen Eigenschaften lassen sich noch weiter unterteilen. So unterscheidet man *Features im Zeitbereich* und *Features im Frequenzbereich*. Die Formeln[1] zur Berechnung der verschiedenen Audioeigenschaften wurden

[1] Signale im Zeitbereich werden durch Kleinbuchstaben dargestellt. Signale im Frequenzbereich werden durch Großbuchstaben dargestellt.

aus [16], [17], sowie [6] entnommen. Die Grafiken in diesem Kapitel wurden mit *Matlab* [2] erzeugt. Matlab ist ein mächtiges Tool in der Signalanalyse.

4.1 Audioeigenschaften im Zeitbereich

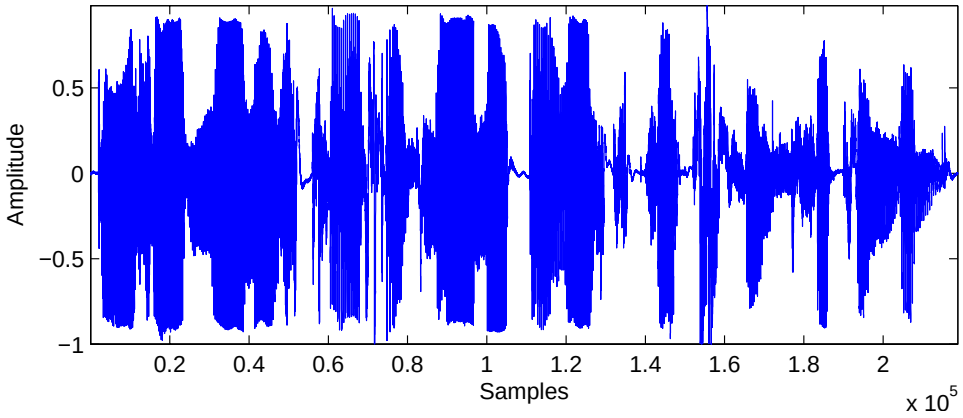


Abbildung 4.1: Diese Grafik zeigt jenes Sprachsignal, aus dem die in diesem Kapitel beschriebenen Features ermittelt werden.

Features im Zeitbereich werden direkt aus der Wellenform ermittelt (Abbildung 4.1). Alle Audioeigenschaften wurden aus der Testdatei „audio1.wav“, die auf dem beiliegenden Datenträger enthalten ist, ermittelt.

4.1.1 Short-Time Energy Function

Die *Short-Time Energy Function* (Abb. 4.2) zeigt die durchschnittliche Energie im Signal pro Zeitausschnitt. Bei Sprachsignalen können stimmhafte[3] und stimmlose Signalanteile unterschieden werden. Ist das Signal nicht zu komplex, so kann man mit der Short-Time Energy Function die stimmhaften Abschnitte durch die deutlich höheren Energiewerte erkennen. Stimmlose Abschnitte weisen deutlich geringere Energiewerte auf [11]. Weiters kann, wenn der Signal-Rausch Abstand ausreicht, Stille von hörbaren Abschnitten unterschieden werden. Ebenso können Rückschlüsse über die Periodizität und den Rhythmus des Signals anhand der Energiewerte gezogen werden.

$$E_n = \frac{1}{N} \sum_m [x(m)w(n-m)]^2, \tag{4.1}$$

[2] http://www.mathworks.com

[3] Stimmhafte Anteile eines Sprachsignals werden von den Stimmbändern erzeugt. Diese liefern die Grundfrequenzen und Harmonien des Signals. Die stimmlosen Anteile werden im Mund- und Rachenraum mit der Zunge und den Zähnen erzeugt.

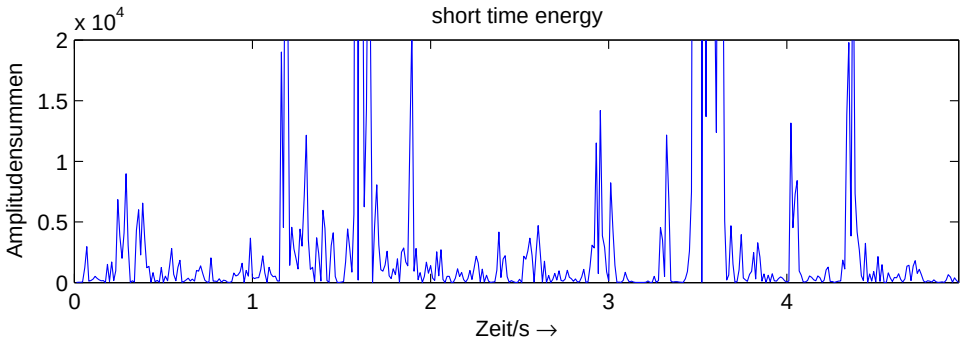


Abbildung 4.2: Short Time Energy: Hohe Werte lassen auf hohe Energie schließen. Bei Sprachsignalen weisen stimmhafte Abschnitte höhere Energien auf als stimmlose.

Dabei ist $x(m)$ das zeitdiskrete Audiosignal, n der Zeitindex der Short-Time Energy und $w(m)$ ist ein *Rechteck-Fenster*:

$$w(n) = \begin{cases} 1, & 0 \leq n \leq N-1, \\ 0, & sonst. \end{cases} \tag{4.2}$$

Dieser Algorithmus ist sehr stark abhängig von der vordefinierten Fensterlänge. Generell eignet sich dieses Feature, um die Energien in einem Audiosignal darzustellen. Der Algorithmus bietet eine gute Grundlage, um „energiereiche“ Songs von „energiearmen“ zu unterscheiden. Bei unterschiedlichen Musikrichtungen zeigen sich häufig große Unterschiede. So erreichen Songs aus den Musikstilen Rock oder Heavy Metal durchschnittlich höhere Amplitudensummen als beispielsweise klassische Musik.

4.1.2 Zero Crossing Rate

Ein *Zero Crossing* tritt – im Sinne von zeitdiskreten Signalen – dann auf, wenn die aufeinander folgenden Samples unterschiedliche Vorzeichen aufweisen. Die Anzahl der Vorzeichenwechsel wird in der *Zero Crossing Rate (ZCR)* dargestellt. An dieser kann beispielsweise bei einem reinen Sinussignal sogar die Frequenz des Signals abgelesen werden. Bei komplexeren Signalen gibt die ZCR Auskunft über die „Verrauschtheit“[4] eines Signals. Bei Sprachsignalen kann man wiederum stimmlose von stimmhaften Abschnitten unterscheiden. Da die stimmlosen Anteile in der Regel höher frequent sind und die stimmhaften Anteile großteils unter 3 kHz auftreten[11], so kann man

[4]Ist ein Signal stark „verrauscht“, so ist die Energie auf sehr, sehr viele unterschiedliche Frequenzen verteilt. Sind nur wenige Frequenzen im Signals vorhanden oder die Frequenzen harmonisch (Vielfache der Grundfrequenz), so klingt dieses klarer und wenig „verrauscht“.

davon ausgehen, dass bei einer hohen ZCR stimmlose Anteile, und bei einer niedrigen ZCR stimmhafte Anteile vorherrschen.

$$ZCR_n = \frac{1}{2}\sum_m |sgn[x(m)] - sgn[x(m-1)]|w(n-m), \tag{4.3}$$

für

$$sgn[x(n)] = \{ \begin{array}{ll} 1, & x(n) \geq 0, \\ -1, & x(n) < 0, \end{array} \tag{4.4}$$

wobei $w(n)$ der Rechteck-Funktion (4.2) entspricht.

4.1.3 Low Energy Function

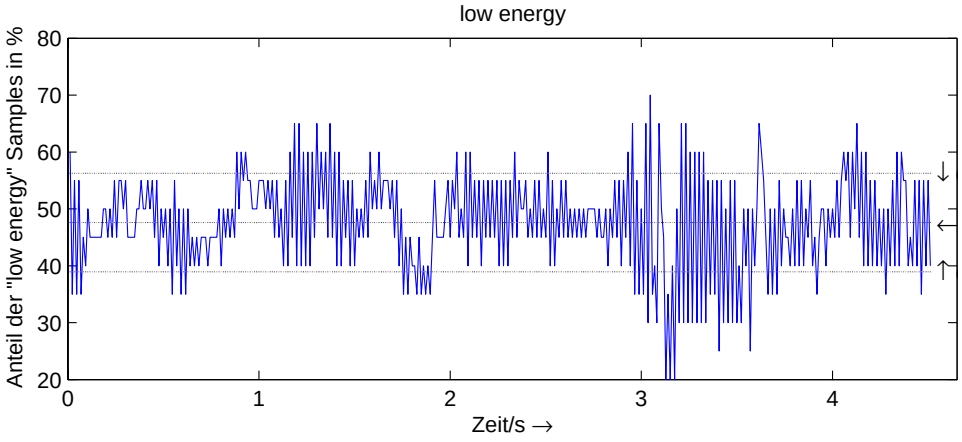


Abbildung 4.3: Low Energy: Hohe Werte weisen auf starke Energieschwankungen im Signal hin, niedrige Werte lassen auf ein wenig perkussives Signal schließen.

Die *Low Energy Function* (Abb. 4.3) liefert den Prozentsatz des Signals, der weniger als durchschnittliche Energie aufweist. Die Low Energy Function gibt Auskunft über die Perkussivität eines Signals und darüber, wie stark ein Signal in der Lautstärke variiert. Je größer der ermittelte Prozentsatz, desto perkussiver ist das Signal und desto stärker sind die Lautstärkenunterschiede, je kleiner der Prozentsatz, desto gleichförmiger das Signal.

4.2 Audioeigenschaften im Frequenzbereich

Neben der Signalbeschreibung im Zeitbereich besteht auch noch die Möglichkeit, das Signal im Frequenzbereich zu analysieren. Wie bereits erwähnt,

kann ein Signal entweder im Zeit- oder im Frequenzbereich betrachtet werden. Dabei ist zu beachten, dass in der zeitlichen Darstellung keine Information über die spektrale Zusammensetzung und im Spektrum keine Zeitinformation enthalten ist. Das bedeutet, dass im Spektrum nur erkannt werden kann, wie stark eine bestimmte Frequenz im Signal enthalten ist, jedoch nicht wie lange. Eine Möglichkeit beide Darstellungen zu kombinieren ist durch ein 3D-artiges *Spektrogramm* gegeben. Im Spektrogramm (Abb. 3.5) ist sowohl die zeitliche, als auch die spektrale Information enthalten. Hier ist allerdings die Genauigkeit nicht beliebig wählbar. So wird – vereinfacht ausgedrückt – das Signal in kurze zeitliche Abschnitte zerlegt, auf diese dann die FFT angewendet wird. Je länger diese Abschnitte sind, desto ungenauer ist zwar die zeitliche Auflösung, desto besser aber die spektrale Auflösung. In Matlab ist das Spektrogramm eines Signals einfach mit folgender Codezeile aufzurufen:

```
specgram(signal);
//SYNTAX: specgram(a,f,fs,window,numoverlap)
//a ... Signal
//f ... Anzahl der FFT-Werte
//fs ... Skalierung
//window ... Fensterfunktion
//numoverlap ... Überlappung der FFT-Blöcke in Samples
```

Standardmäßig wird eine FFT-Blockgröße von 256 angenommen, die Überlappung beträgt die halbe Blockgröße und das Signal wird mit einem *Hanning*-Fenster multipliziert.

Meist genügt für die Analyse eine grobe Darstellung der Energieverteilung im Spektrum. Aus diesem Grund kann bei spektralen Features sowohl auf die Phase, als auch in den meisten Fällen auf eine hohe Auflösung verzichtet werden. Spektrale Features sind entscheidend für die generelle Klassifizierung von Audiodaten und für die Spracherkennung.

4.2.1 Cepstrum, (linear) Cepstral Coefficients (mittels DFT)

Die Bezeichnung *Cepstrum* als Anagramm des Wortes Spectrum deutet auf die Art der Berechnung hin: Das *Cepstrum* kann nämlich als Spektrum des Spektrums bezeichnet werden. Wird das Signal $x(n)$ aus dem Zeitbereich fourier-transformiert, so entsteht die spektrale Darstellung – das Spektrum $X(\omega)$. Das Cepstrum $c_x(n)$ wiederum ist definiert durch die Diskrete Fourier Transformation von $C_x(\omega)$, wobei $C_x(\omega) = \log_e X(\omega)$. Es gibt zwei Varianten des Cepstrums: Einerseits das *Real Cepstrum*, welches die Logarithmusfunktion für reelle Zahlen verwendet, und andererseits das *Complex Cepstrum*, welches die Logarithmusfunktion für komplexe Zahlen verwendet. Das komplexe Cepstrum erlaubt die Rekonstruktion des ursprünglichen

Signals, da sowohl Informationen über Energie und Phase erhalten bleiben. Beim realen Cepstrum werden nur die Energiewerte des Spektrums verwendet.
Das Cepstrum gibt somit Auskunft über die Veränderungen in den verschiedenen Frequenzbändern. Aufgrund der Logarithmierung im Frequenzbereich werden harmonische Anteile im Signal deutlicher erkennbar, auch wenn sie nur relativ kleine Amplituden aufweisen. In der Audioanalyse sind die *Cepstral*[5] *Coefficients* ein geeigneter Eigenschaftsvektor, um Sprache und Musiksignale abzubilden, da aufgrund der logarithmischen Skalierung des Spektrums die harmonischen Anteile in einem Signal von den übrigen Anteilen getrennt werden.

Mit den Cepstral Coefficients wird versucht, die spektrale Energieverteilung zu approximieren, wobei bei den *Linear Cepstral Coefficients* auf die menschliche Wahrnehmung nicht Rücksicht genommen wird. Die grobe spektrale Form wird bereits durch die ersten Koeffizienten definiert, wobei der allererste Koeffizient – die Energie – normalerweise vernachlässigt wird. Deshalb ist auch nur eine geringe Anzahl (ca. 10) von Koeffizienten notwendig, um eine zufriedenstellende Annäherung zu erhalten [6]. Die Genauigkeit des Ergebnisses ist zwar von der Anzahl der verwendeten Koeffizienten abhängig, eine starke Erhöhung der Koeffizientenanzahl bringt allerdings keine signifikante Verbesserung der Approximation [6].

Mit den Matlab-Befehlen `cceps`, `icceps` und `rceps` können die komplexe Cepstral-Analyse sowie das inverse komplexe Cepstrum oder das reale Cepstrum ermittelt werden. Für die Berechnung des komplexen Cepstrums werden mit dem Befehl `cceps(signal)` folgende Schritte durchgeführt:

```
x = signal;
h = fft(x); //fast fourier transformation des signals
logh = log(abs(h)) + sqrt(-1)*rcunwrap(angle(h));
//logarithmus der fft-werte + phase
y = real(ifft(logh)); //inverse fft -> ausgabe nur realteil
```

Die Anwendungsgebiete der *Cepstrumanalyse* [9] in der Signalanalyse werden in der Gegenwart immer vielfältiger: Neben der Sprachanalyse und Spracherkennung [9] wird die Cepstrumanalyse für seismische Signale (z. B. das Auffinden von seismischen „Echos“ nach Erdbeben oder Bombenexplosionen), biomedizinische Signale, antike Akustikaufnahmen oder Sonarsignale verwendet. Mittlerweile findet die Cepstrumanalyse sogar bei der Schadensfrüherkennung an Maschinen Anwendung. In diesem Fall wird der Umstand ausgenutzt, dass sich Schäden an Maschinen mit umlaufenden Bauteilen häufig durch einen Anstieg der harmonischen Komponenten im Luft- oder Körperschall andeuten.

[5] *cepstral* ist ein Anagramm von spectral

4.2.2 Mel Frequency Cepstral Coefficients

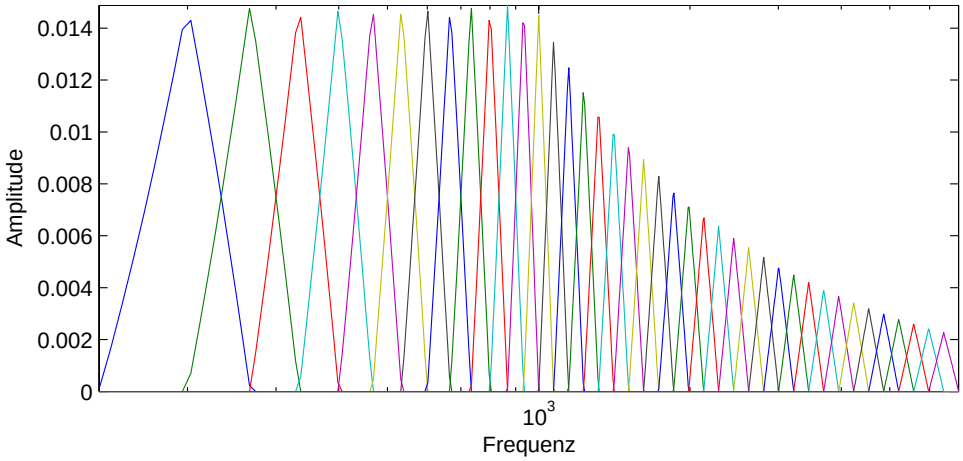


Abbildung 4.4: MFCC Filterbank bestehend aus 40 Filtern, die alle mit Dreiecksfunktionen gebildet werden: Im unteren Frequenzbereich werden 13 Filter mit einem linearen Abstand von 133.33 Hz zwischen den Mittenfrequenzen verwendet. Im oberen Frequenzbereich folgen 27 Filter mit logarithmischen Abständen von jeweils dem 1.0711703-fachen der Mittenfrequenz des Filters. Die Grafik wurde mit Matlab erstellt und die X-Achse ist logarithmisch skaliert.

In der menschlichen Wahrnehmung ist der Frequenzbereich von 100 bis 200 Hz gleich wichtig wie der Frequenzbereich von 10 bis 20 kHz. Um ein wahrnehmungsgetreueres Ergebnis zu erhalten, wäre eine Frequenzskalierung – die besonders auf die menschliche Wahrnehmung eingeht – vor der Berechnung der Koeffizienten besser geeignet. Bei den *Mel Frequency Cepstral Coefficients (MFCC)* wird dieser Umstand berücksichtigt und das Spektrum auf der Mel[6]-Skala basierend skaliert. Die MFCC werden vorwiegend für die Spracherkennung verwendet, weil vor allem die der menschlichen Wahrnehmung entsprechenden Frequenzanteile erfasst werden. In Abbildung 4.4 ist die Filterbank zu sehen, die zur Skalierung des Spektrums verwendet wird. Die Filterbank besteht im unteren Frequenzbereich aus 13 Filtern mit linearem Abstand (133.33 Hz) zwischen den Mittelfrequenzen der Filter, also $MF_{i+1} = MF_i + 133.33\ Hz$, und im oberen Bereich aus 27 Filtern mit logarithmischem Abstand der Mittenfrequenzen ($MF_{i+1} = MF_i * 1.0711703$). Durch diese Skalierung ist die Auflösung im unteren Frequenzbereich relativ hoch, während sie im oberen Frequenzbereich logarithmisch abnimmt ([8] und [13]).

[6]Mel ist eine Einheit für die empfundene Tonhöhe in der Psychoakustik.

In Matlab können die Koeffizienten mit Hilfe der *Auditory Toolbox*[7] ermittelt werden. Die Dokumentation zur Auditory Toolbox ist in [13] zu finden. Der Matlab-Befehl lautet wie folgt:

```
[ceps, freqresp, fb, fbrecon, freqrecon]
= mfcc(input, samplingRate, [frameRate]);
//SYNTAX
//ceps ... sind die Mel-frequenzabhängigen
//Cepstralen Koeffizienten
//optionale Rückgabewerte:
//freqresp ... detailierte FFT-Magnitude
//fb ... log10 Mel-skalierte Filterbankausgabe
//fbrecon ... rekonstruierte Filterbankausgabe
//durch Invertierung der Cosinus-Transformation
```

4.2.3 Rasta (= RelAtive SpecTrAl)

Rasta wurde ursprünglich implementiert, um die Differenzen zwischen unterschiedlichen Sprechern zu minimieren und die wichtige Information trotzdem zu erhalten. Rasta wendet einen Bandpassfilter auf die Energien aller Subbänder an, um Störungen über kurze Zeit zu glätten und einen konstanten Offset zu entfernen (z. B. Telefon). Die Subbänder werden anhand der Bark[8]-Skala ermittelt. Rasta wurde ursprünglich entwickelt, um Umwelteinflüsse zu korrigieren und Adaptionsprozesse im Bereich der menschlichen Wahrnehmung darzustellen. Im Grunde werden niedrigfrequente Anteile (unter 1 Hz) herausgefiltert, da diese Veränderungen in der Umwelt darstellen. Ebenfalls entfernt werden Frequenzanteile über 13 kHz, da diese Veränderungen schneller sind, als sie vom menschlichen Sprachapparat erzeugt werden können. Als Input dient ein Array von spektralen Daten, das entweder von *LPC* oder von *MFCC* Routinen generiert wird. Der ursprüngliche Rasta Filter ist für eine Framerate von 100 Hz vorgesehen. Mit dem Matlab-Befehl `y=rasta(x, fs)`(`x` entspricht dem eingelesenen Signal und `fs` ist die Abtastfrequenz), der ebenso aus der bereits erwähnten Auditory Toolbox stammt, kann diese Feature einfach berechnet werden.

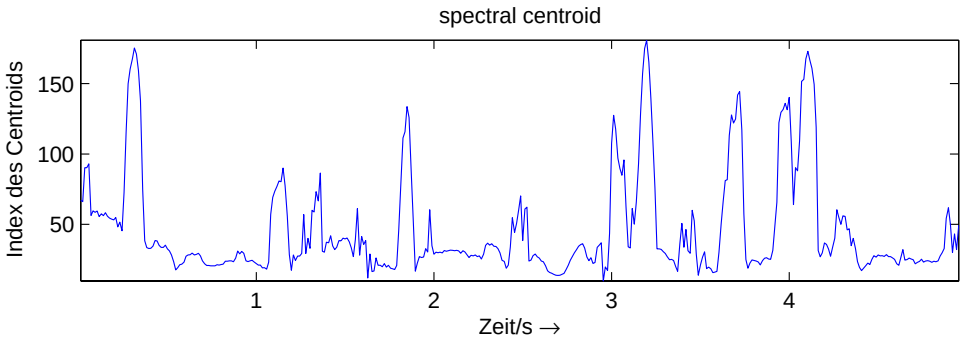


Abbildung 4.5: Der Spectral Centroid stellt den Schwerpunkt des Signals dar. Auf der Y-Achse ist der jeweilige Index des Schwerpunkts im FFT-Array dargestellt.

4.2.4 Spectral Centroid

Der *Spectral Centroid (SC)* stellt den Schwerpunkt des Signals dar und ist ein Indikator für die „Fülle“ eines Audiosignals. Bei gleichbleibender Tonhöhe hat ein Signal mit mehreren oder stärkeren *Harmonien* einen höheren Schwerpunkt. Das gerundete Ergebnis der folgenden Formel entspricht dem Index im FFT-Array, der dem Spectral Centroid am nächsten ist.

$$SC = \frac{\sum_{k=0}^{\frac{N}{2}} k \cdot |X(k)|}{\sum_{k=0}^{\frac{N}{2}} |X(k)|} \tag{4.5}$$

Die grafische Darstellung der Centroids (Abb. 4.5) reicht nicht aus, um konkrete Schlüsse ziehen zu können. Der Schwerpunkt ist auch abhängig von der Tonhöhe des Signals. Der eigentliche Indikator für die „Fülle“ eines Klangs ergibt sich erst aus dem Verhältnis der Spectral Centroids zur Tonhöhe des Signals. Somit ist Spectral Centroid als Feature ohne die Kombination mit der Tonhöhe des zu analysierenden Signals eher ungeeignet.

4.2.5 Spectral Fluctuation

Spectral Fluctuation (Abb. 4.6) beschreibt die lokalen spektralen Veränderungen und somit Veränderungen in der Frequenz. *Spectral Fluctuation* de-

[7] Download-Möglichkeiten:
http://www.mathtools.net/MATLAB/Signal_processing/Acoustics/ oder
http://www.speech.cs.cmu.edu/comp.speech/Section1/HumanAudio/auditory.tlbx.html

[8] Bark ist ebenso eine Einheit für die empfundene Tonhöhe in der Psychoakustik. Mehr Informationen zu Mel und Bark ist im Anhang A oder auf http://www.wikipedia.de zu finden.

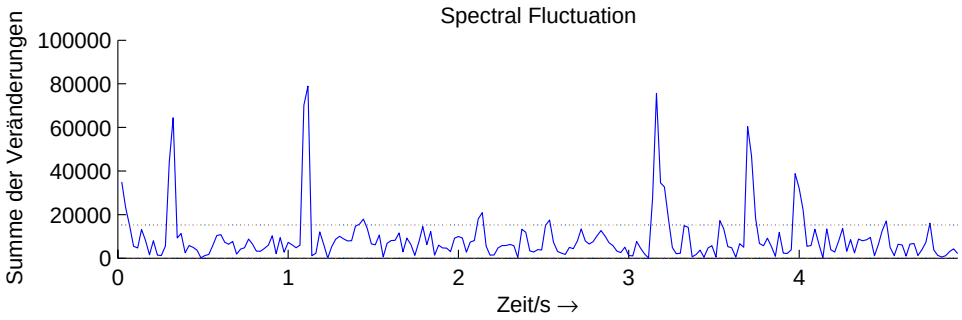


Abbildung 4.6: Die Spectral Fluctuation stellt die spektralen Veränderungen eines Signals im Zeitverlauf dar.

finiert die Differenz der Amplituden bei der FFT zwischen aktuellem und nachfolgendem Frame.

$$flux_t = |\sum_{k=0}^{\frac{N}{2}} FFT_t - \sum_{k=0}^{\frac{N}{2}} FFT_{t-1}| \tag{4.6}$$

für

$$t \ldots Zeit \qquad N \ldots Fenstergre$$

Die Spektrale Fluktuation liefert als Ergebnis nicht nur die spektralen Veränderungen eines Signals, sondern definiert im Grunde auch, wie stark sich ein Signal verändert, beziehungsweise ob es eher monoton wirkt. Es ist also sinnvoll, dieses Feature als Vergleichsmaß zwischen unterschiedlichen Musikrichtungen zu verwenden.

4.2.6 Spectral Rolloff

Spectral Rolloff (Abb. 4.7) ist wie der Spectral Centroid ein Maß für die spektrale Form eines Signals. Dieses Feature liefert den Index des FFT-Arrays, an dem alle tiefer liegenden Frequenzen 85% der gesamten Energie im Spektrum erreicht haben [17]. Sind viele tieffrequente Anteile vorhanden, so ist dieser Index niedriger, sind viele hohe Frequenzen im Spektrum vorhanden, so ist der Index höher.

4.2.7 Spectral Bandwidth

$$BW = \sqrt{\frac{\sum_{k=0}^{\frac{N}{2}} (k - SC)^2 \cdot |X(k)|^2}{\sum_{k=0}^{\frac{N}{2}} |X(k)|^2}} \tag{4.7}$$

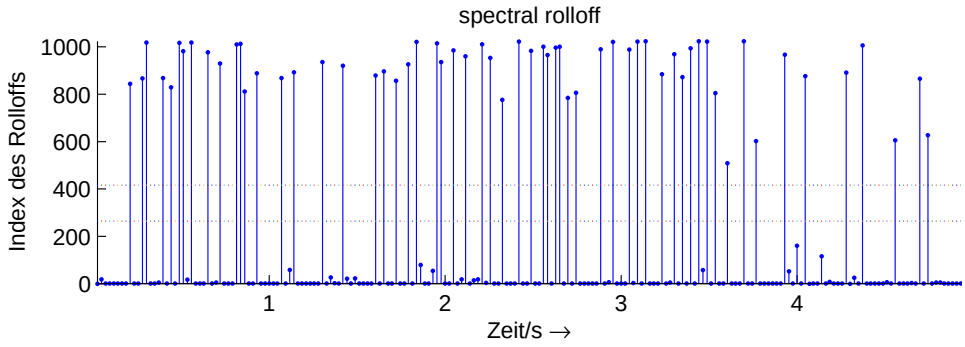


Abbildung 4.7: Spectral Rolloff zeigt jenen Index des FFT-Arrays, an dem alle tieferen Frequenzen 85% der gesamten Energie des Signals aufweisen.

Spectral Bandwidth beschreibt die Frequenzbreite eines Signals, wobei schwache spektrale Komponenten nur wenig berücksichtigt werden. Die Berechnung ist auch abhängig von der Position des Spectral Centroid. Diese Signaleigenschaft wird aussagekräftiger, wenn sie gemeinsam mit dem Spectral Centroid und/oder der Tonhöhe betrachtet wird. Je kleiner die spektrale Bandbreite, desto konzentrierter die Energieverteilung im Signal. So ist bei stimmhaften Lauten die spektrale Bandbreite wesentlich geringer als bei einem Signal, das einen hohen „Rauschanteil“ hat.

4.2.8 Fundamental Frequency

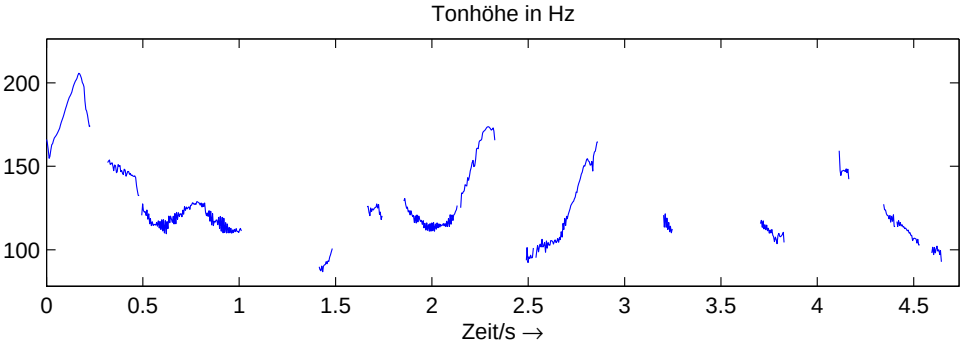


Abbildung 4.8: Die Grundfrequenz des Sprachsignals ist nur dort eingezeichnet, wo ausreichend stimmhafte Sprachanteile für die Analyse vorhanden sind.

Ein harmonischer Klang besteht aus einer Reihe von bedeutenden Fre-

quenzkomponenten, welche die *Fundamental Frequency*, also die Grundfrequenz und die ganzzahligen Vielfachen derselben beinhalten. In Abbildung 4.8 ist die Grundfrequenz des Sprachsignals eingezeichnet. Sie ist abhängig von den stimmhaften Sprachanteilen. Diese können – aufgrund ihrer Oberfrequenzen und Harmonien – als harmonisch bezeichnet werden. Dort, wo stimmlose Sprachanteile vorherrschen, ist in der Grafik keine Grundfrequenz eingezeichnet. Die Grundfrequenz lässt sich nur bei harmonischen Klängen (vor allem Sprache und Musik) sinnvoll herausfiltern. Die meisten Geräusche in unserem Umfeld sind eher nicht harmonisch, obwohl es auch hier Ausnahmen gibt (z. B. manche Türklingeln, Vogelgesang, ...).

4.3 Beaterkennung

Die Grundlagen zu diesem Kapitel können in [10] und in [14] nachgelesen werden.

4.3.1 „Statistical streaming beat detection“

„simple sound energy“

Das menschliche Gehör bestimmt den Rhythmus von Musik über eine pseudoperiodische Folge von Beats. Das Signal, das vom Ohr aufgenommen wird, beinhaltet eine gewisse Energie, die im Ohr in ein elektrisches Signal umgewandelt wird. Diese elektrische Form des Signals wird vom Gehirn interpretiert. Das Signal scheint natürlich umso lauter, je mehr Energie vorhanden ist. Ein Beat wird aber nur dann wahrgenommen, wenn eine grobe Variation der Soundenergie auftritt. Wenn also ein monotoner Klang wahrgenommen wird, der von Zeit zu Zeit starke Energiespitzen aufweist, ist es dem Menschen möglich, einen Beat zu erkennen. Wird allerdings ein Klang wahrgenommen, der kontinuierlich gleich laut ist, so kann kein Beat extrahiert werden. Beats sind also plötzliche Veränderungen in der Energie eines Klangs. Dauert ein Beat länger als eine Sekunde wird er nicht mehr als solcher wahrgenommen.

Der einfachste der folgenden Beat Detection Algorithmen arbeitet mit Energiespitzen. Bei der Berechnung wird die durchschnittliche Energie eines Sounds mit der aktuellen Energie verglichen. Dazu wird das Signal in einem Pufferspeicher (Buffer) von einer Sekunde Länge gehalten. Bei einer Abtastfrequenz von 44100 Hz und einer Fensterlänge von 1024 Samples, werden somit immer 43 Frames beobachtet. Die Amplituden eines solchen Frames werden addiert, anschließend quadriert und im Buffer gespeichert. Kommt nun ein neues Frame in den Buffer, wird die Summe der Abtastwerte quadriert und mit dem Durchschnitt der 43 Werte im Buffer verglichen. Ist das Energiequadrat des neuen Frames höher als der Durchschnitt, so wird ein

Beat registriert. Das älteste Frame fällt aus dem Buffer und das aktuelle Frame wird hineingeschrieben. So wird das gesamte Testsignal durchlaufen. Natürlich kann man diesen Algorithmus noch etwas verbessern. So kann man noch detektieren, wie perkussiv ein Signal ist. Anhand der Abweichungen der Energiewerte vom Durchschnitt im Buffer kann man einen Faktor ableiten. Multipliziert man diesen Faktor mit dem Durchschnitt der Energiewerte und vergleicht diesen erst im Anschluss mit dem Energiewert des neuen Frames, so kann man die „Empfindlichkeit“ des Algorithmus beeinflussen. Je höher der Faktor, umso unempfindlicher der Algorithmus, was bei perkussiven Songs (z. B. Techno, Rap, ...) durchaus von Vorteil sein kann. Bei stark verrauschten Signalen, mit E-Gitarren und anderen verzerrenden Elementen (beispielsweise Rock- oder Heavy-Metal-Songs) sollte der Faktor niedriger sein, sodass der Algorithmus noch Beats erkennen kann. Hier stößt dieser Algorithmus aber auch rasch an seine Grenzen. Der Algorithmus detektiert nur die größten Energiewerte eines Signals, kann aber nicht wie der Mensch aus einem Signal leisere Elemente, wie zartere Beats mit dem Drumstick herauslesen. Der Algorithmus würde nur die Energiewerte des lauteren Signals wahrnehmen.

„frequency selected sound energy“

Um zwischen verschiedenen Tonlagen unterscheiden zu können, muss der weiterentwickelte Algorithmus die Möglichkeit haben, zwischen den verschiedenen Frequenzbändern zu unterscheiden und dort festzustellen, ob erwähnenswerte Beats vorkommen. Grundsätzlich wird nun in diesem Algorithmus geprüft, ob es starke Variationen in der Energie eines Sounds gibt und in welchem Subband diese Veränderungen auftreten. So können auch Beats detektiert werden, die im Zeitbereich durch lautere Frequenzanteile maskiert wurden. Dadurch kann eine Gewichtung auf niederfrequente beziehungsweise auf hochfrequente Beats erreicht werden.
Bei diesem Algorithmus wird das Signal mittels einer Fast Fourier Transformation vom Zeit- in den Frequenzbereich gebracht. Anschließend wird das Signal dort in verschiedene Frequenzbänder aufgeteilt und wieder in den Zeitbereich zurück transformiert. Jedes dieser Bänder wird dann wie im einfachen “Simple Sound Energy“-Algorithmus behandelt. Natürlich gibt es auch hier weitere Verbesserungsmöglichkeiten. So kann auch hier ein Faktor verwendet werden, der die Empfindlichkeit des Algorithmus steuert. Weiters besteht die Möglichkeit, dass innerhalb eines Bufferzeitraums ein aktuelles Maximum ermittelt wird. So wird beispielsweise ein Beat erst dann als Beat gewertet, wenn dieser mindestens 30 Prozent des stärksten Beats im Buffer überschreitet. Ist ein Beat noch stärker als das aktuelle Maximum, so wird dieser das Maximum. Ist ein Buffer komplett neu gefüllt, kann der Durchschnitt des alten Buffers als neues Maximum gesetzt werden. So wird verhindert, dass mit jeder neuen Bufferfüllung ein neuer Beat registriert wird.

Eine weiteres Kriterium zur Ermittlung eines Beats sind auch die Abweichungswerte an sich: Eine negative Abweichung im Vergleich zum Durchschnitt kann kein Beat sein. So wird ein Beat nur dann als Beat gewertet, wenn eine gewisse Energiemenge vorhanden ist und gleichzeitig die aktuelle Abweichung positiv ist.

4.3.2 „filtering rhythm detection“

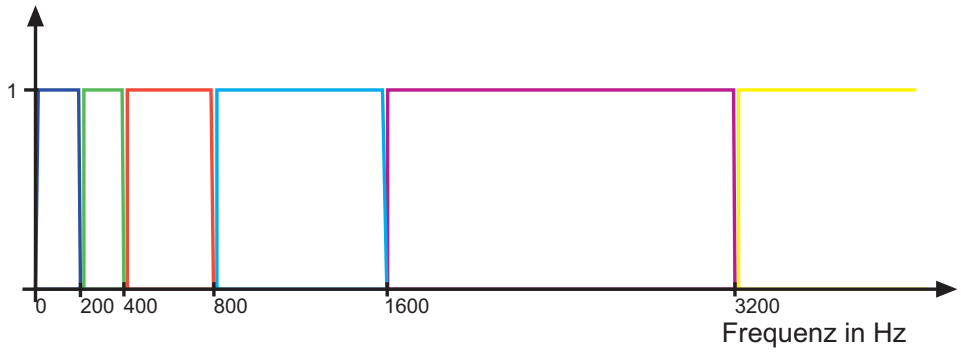


Abbildung 4.9: Beim „filtering rhythm detection“-Algorithmus wird das Audiosignal in 6 Frequenzbänder aufgeteilt.

Im Gegensatz zu den bisher besprochenen Algorithmen werden in diesem Algorithmus die Beatschläge nicht „gezählt“, sondern die Grundidee besteht darin, dass man mittels einer Funktion zwei Signale auf Ähnlichkeiten überprüft. Diese Funktion wird als *Interkorrelationsfunktion* bezeichnet.

Aufteilung in Frequenzbänder

In einfacher Form kann der Algorithmus auch ohne Unterteilung des Signals in einzelne Frequenzbänder verwendet werden. Dabei ist aber wieder zu beachten, dass keine Rücksicht auf die Energieverteilung im Spektrum genommen wird. Wie in [14] beschrieben, wird in diesem Algorithmus das Signal in 6 Frequenzbänder (Abb. 4.9) unterteilt. So können bei Musikstücken unterschiedliche Instrumentengruppen besser unterschieden werden und vorhandene Periodizitäten leichter erkannt werden.

Ableitung und Kammfilter

Um auf die Beats Per Minute – BPM[9], also die Songgeschwindigkeit zu kommen, wird das Audiosignal mit Kammfiltern unterschiedlicher Periodizitäten

[9] BPM . . . Schläge pro Minute. In der Musik übliches Maß für die Geschwindigkeit eines Musikstücks.

verglichen. So könnte man Kammfilter in einem Bereich von 60 – 240 BPM einrichten und mit dem Signal vergleichen. Das „Vergleichen“ erfolgt durch eine Faltung im Zeitbereich. Um diese Prozedur zu vereinfachen, werden die beiden Signale fouriertransformiert und dann im Spektrum miteinander multipliziert. Je besser die rhythmischen Strukturen des Signals mit einem der Kammfilter zusammenpassen, desto höher ist die Energie der Faltung.

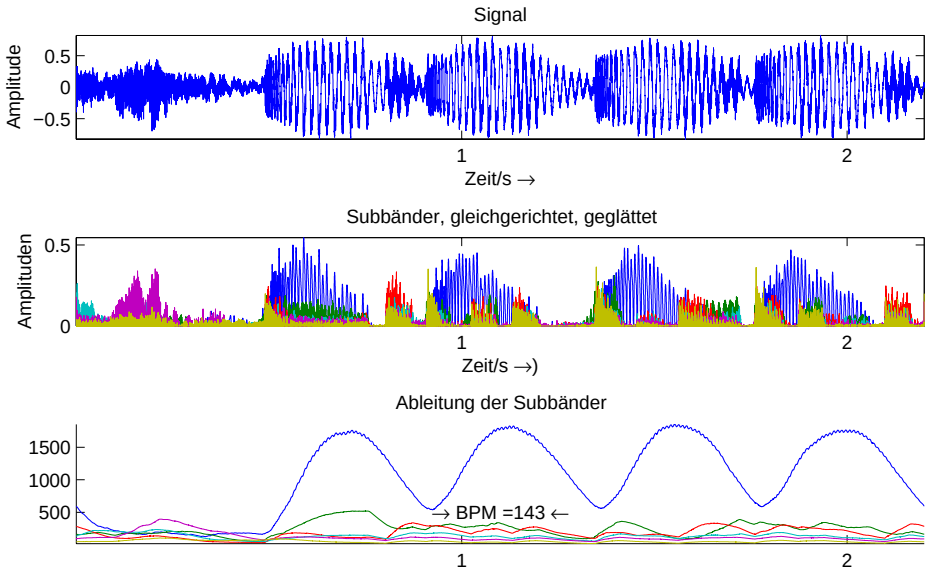


Abbildung 4.10: Der „filtering rhythm detection“-Algorithmus angewandt auf ein perkussives Musikstück. Die Audiodatei „audio3.wav“ ist auf dem beiliegenden Datenträger enthalten. In dieser Grafiken sind die einzelnen Bearbeitungsschritte vor der Faltung mit den Kammfiltern dargestellt. Deutlich ist zu erkennen, wie sich aus einer wenig aussagekräftigen Darstellung des Signals im Zeitbereich (ganz oben) die Sichtbarkeit der Beat-Struktur durch vorbereitende Maßnahmen stark verbessern lässt.

Um den Algorithmus zu optimieren wird, das Signal vorher noch einigen Bearbeitungsschritten unterzogen.

- Filterbank:
  Hier wird das Signal im Frequenzbereich in einzelne Teilbänder zerlegt und diese werden dann wieder in den Zeitbereich zurücktransformiert.

- Tiefpassfilter und Signal-Gleichrichtung:
  In diesen Bearbeitungsschritten wird das Signal im Zeitbereich gleich-

gerichtet, sodass keine negativen Amplitudenwerte mehr vorkommen. Es wird somit der Betrag der Amplitude verwendet. Anschließend wird das Signal wieder in den Frequenzbereich transformiert, wo das Signal mit der FFT des abfallenden Hälfte eines Hanning Fensters multipliziert, was einer Faltung im Zeitbereich entspricht und einer Tiefpassfilterung gleichkommt. Durch diese Schritte werden die Signale der Teilbänder noch weiter vereinfacht.

- Ableitung:
  Durch Differenzieren werden die Beats leichter detektierbar und der Wertebereich verringert sich, da nicht mehr mit den hohen Amplitudenwerten gerechnet werden muss.

- Kammfilter:
  In diesem finalen Schritt wird das Signal mit den verschiedenen Kammfiltern gefalten, und die Faltung mit dem Kammfilter, dessen Peakabstände am genauesten zum Rhythmus des Signals passen, wird die höchste Energie aufweisen. Dieser Schritt kann auch öfter aufgerufen werden, um die Kammfilter immer feiner an das Songtempo anzupassen.

Trotz all dieser Optimierungsschritte ist dieser Algorithmus äußerst rechenintensiv und zeitaufwändig. Er eignet sich somit nicht für Audiostreams oder die Analyse kompletter Songs. Dieser Algorithmus wird nur auf einen kleinen Ausschnitt von nur wenigen Sekunden ausgeführt. Dieser Ausschnitt wird zufällig aus dem Musikstück entnommen, was allerdings die Annahme voraussetzt, dass die Geschwindigkeit des Musikstücks überall gleich ist.

Die Auswirkungen all dieser Schritte sind in der Abbildung 4.10 zu erkennen.

4.3.3 Beat Histogramm

Auch dieser Ansatz (aus [7]) ermittelt das Tempo des Signals durch „Vergleichen“. Zuvor wird das Signal aber in einigen Schritten vorbereitet:

- „Wavelet Decomposition“:
  Zu allererst wird das Signal in mehrere Frequenzbänder unterteilt. Dazu wird die *Diskrete Wavelet Transformation (DWT)* [9] verwendet. Die DWT ist eine Alternative zur Diskreten Fourier Transformation und ermöglicht das Zerlegen eines Audiosignals in „oktaven“-breite Frequenzbänder.

- Hüllkurvenermittlung:
  Nach Zerlegung in die Teilbänder, werden folgende Schritte angewandt, um die Hüllkurven der einzelnen Bandsignale im Zeitbereich zu ermitteln.

  - Gleichrichter und Tiefpassfilter
    Wie schon im vorhergehenden Beaterkennungsalgorithmus wird auch hier das Signal gleichgerichtet und dann ein Tiefpassfilter angewendet. Dieser glättet das Signal im Zeitbereich und soll die Beats deutlicher erkennbar machen.
  - „Mean Removal“
    In diesem Schritt wird vom Signal der Mittelwert entfernt, was das Signal um den Wert 0 zentriert.

- Autokorrelationsfunktion und Beathistogramm
  Nach Ermittlung der Hüllkurve werden die Amplituden der einzelnen Teilbänder wieder zusammengezählt und auf die entstehende Summe wird die Autokorrelationsfunktion berechnet.

$$x[k] = \frac{1}{N} \sum_{n} x[n]x[n-k] \tag{4.8}$$

  Die dominanten Spitzen der resultierenden Autokorrelationsfunktion hängen mit den Zeitabständen zusammen, in denen das Signal die größte „Selbstähnlichkeit“ hat. Im Beat Histogramm entspricht jeder Eintrag einem gewissen Tempo in BPM und pro Analyse-Fenster werden die drei stärksten Beats ins Histogramm eingetragen. Die stärksten Beats im Histogramm entsprechen somit den stärksten Beats im Signal.

Kapitel 5

Plugins

5.1 Plugins in Windows

Die Ausführungen in diesem Abschnitt wurden frei aus http://www.wikipedia.de übernommen.

5.1.1 Component Object Model

Das **C**omponent **O**bject **M**odel (COM) ist eine von Microsoft entwickelte proprietäre Technologie, um in Windows Klassen aus Dynamic Link Libraries (DLLs) zu exportieren. (Aus DLLs können Funktionen aufgerufen werden, die aber einen Zeiger auf ein Exemplar einer Klasse als Rückgabewert haben müssen.) Somit soll COM eine leichte Wiederverwendung von bereits geschriebenem Programmcode möglich machen.

COM basiert auf dem Client/Server-Prinzip: Der COM-Server bietet die zu exportierenden Klassen - die so genannten COM-Objekte, welche über eine COM-Schnittstelle definiert werden - an. Der Server umfasst also den Code. Der Client ist das Programm, welches die vom Server angebotenen Funktionen benutzt. Der Client kennt die Funktionen, die vom COM-Server angebotene COM-Objekte bieten, da diese in den entsprechenden COM-Schnittstellen deklariert sind. Die Vermittlung zwischen Client und Server übernimmt der sogenannte „Marshaler“, der beispielsweise Datentypen konvertiert.

Eine COM-Schnittstelle selbst ist COM-intern ein Zeiger, der auf die so genannte *VTable* zeigt, eine Tabelle, die wiederum Zeiger auf die Funktionen enthält, die vom COM-Objekt angeboten werden. Eine Schnittstelle hat außerdem eine weltweit einmalige Identifikationsnummer, die GUID (Globally Unique Identifier), welche es eindeutig identifiziert, so dass auch mehrere Schnittstellen mit demselben Namen existieren können (aber nicht mit derselben GUID). Dies vermeidet Namenskonflikte zwischen COM-Objekten unterschiedlicher Hersteller.

Durch den Einsatz von COM erhält ein Programmierer die Möglichkeiten

- sprachunabhängig,
- versionsunabhängig,
- plattformunabhängig,
- objektorientiert,
- ortsunabhängig

zu programmieren. Viele der Funktionen des *Windows Platform SDKs* sind über COM zugänglich. COM ist die Basis, auf der OLE[1]-Automation und ActiveX realisiert sind. Mit der Einführung des *.NET-Frameworks* verfolgt Microsoft allerdings die Strategie, COM unter Windows durch ebendieses Framework abzulösen.

5.1.2 ActiveX

ActiveX ist die Bezeichnung für ein Softwarekomponenten-Modell von Microsoft für aktive Inhalte. ActiveX-Komponenten entsprechen einer Erweiterung des COM-Standards von Microsoft.

Es handelt sich hierbei um Software-Komponenten für andere Anwendungen, Makroprogrammierungen und Entwicklungsprogramme; sie können gleich in verschiedenen Programmiersprachen und Umgebungen verwendet werden. Einige Programme nutzen zum Beispiel den Internet Explorer zur Anzeige von Informationen. ActiveX gibt es nur für das Betriebssystem Windows. Bis zur Einstellung des Internet Explorers für den Mac waren auch ActiveX-Steuerelemente auf Macintosh-Computern ausführbar.

Der Einsatz von ActiveX-Komponenten in Webbrowsern ist problematisch, da die Technologie dafür nicht hundertprozentig geeignet ist. Soll eine ActiveX-Komponente in einem Browser genutzt werden, müssen bei ihrer Programmierung bestimmte Regeln befolgt werden, um Sicherheitslücken möglichst zu vermeiden. Da diese Regeln zu selten eingehalten werden und auch die ActiveX-Unterstützung des Internet Explorers fehlerhaft ist, wird der Einsatz von ActiveX-Komponenten in Webbrowsern oft kritisiert.

Doch es gibt noch andere Einsatzgebiete: So wird die ActiveX-Technologie von Visual Basic (bis einschließlich Version 6.0) und dessen Ableger VBA ausgiebig genutzt - vorrangig zur Oberflächengestaltung.

5.2 Plugins in Maya

In Windows sind Maya-Plugins herkömmliche Dynamic Link Libraries mit der speziellen Dateiendung `.mll` anstelle von `.dll`. In Unix-Umgebungen

[1] **O**bject **L**inking and **E**mbedding ist ein von Microsoft entwickelter Standard für den einfachen Datenaustausch via Drag'n'Drop oder Copy/Paste – http://de.wikipedia.org/wiki/OLE_%28EDV%29

sind die Plugins Dynamic Shared Objects mit der Dateiendung `.so`. Standardmäßig wird bei der Installation von Maya das *Maya Development Kit* mitinstalliert. Im Development Kit sind alle notwendigen Header- und Library-Files für das Erstellen von Plugins enthalten. Im Maya-Verzeichnis müssten dann die Ordner

- *maya_verzeichnis/include*
- *maya_verzeichnis/libs*
- *maya_verzeichnis/devkit*

vorhanden sein. Da ein Maya-Plugin eine Dynamic Link Library ist, müssen auch hier ein Einstiegs- und Ausstiegspunkt bereitgestellt werden. Diese Funktionen werden aufgerufen, wenn das Plugin zum ersten Mal geladen oder „entladen“ wird. Bei Windows-Plugins wird das normalerweise von der jeweiligen `DllMain`-Funktion erledigt. In der Maya C++ API müssen zu diesem Zweck in jedem Plugin die beiden nachstehend angeführten Funktionen enthalten sein:

```
MStatus initializePlugin( MObject obj )
MStatus uninitializePlugin ( MObject obj )
```

Mehr Details zu Syntax, zur Installation und Tutorials zur jeweils aktuellen Maya-Version sind unter http://www.davidgould.com zu finden.

Wichtig: Im Unterschied zum „herkömmlichen“ objektorientierten Programmieren sind in Maya die Funktionen und Daten *nicht* in einer Klasse vereint. Hier werden Daten und Funktionen getrennt, was bei der Vererbung zu beachten ist. Alle Klassen sind von der Klasse `MObject` abgeleitet, alle Funktionen von der Klasse `MFn Maya Function Sets`.

5.2.1 Aufbau von Maya

Um den Aufbau von Maya-Plugins besser zu verstehen, ist es notwendig, die grundlegenden Konzepte von Maya zu kennen: Maya kann als offene Architektur für das Erstellen von Computergrafik betrachtet werden. Der Benutzer hat die Möglichkeit, das komplette User Interface[2] zu verändern oder neue Programmteile hinzuzufügen. Die gesamte grafische Benutzeroberäche ist in *Maya Embedded Language (MEL)* geschrieben und kann auch mittels MEL-Befehlen gesteuert werden. Neben der Kontrolle über die Benutzeroberfläche kann mit MEL auch auf alle Einstellungen von Maya zugegriffen werden. Mit den beiden Programmier-Schnittstellen (MEL und C++) können sowohl Datenimport- als auch Exportroutinen implementiert werden,

[2] Benutzeroberfläche

um beispielsweise das Exportieren von 3D-Objekten oder anderen Daten in beliebigen Datenformaten zu ermöglichen. Die gesamte Funktionalität von Maya kann in eine eigenständige benutzerdefinierte Anwendung kompiliert werden. So können Maya- und eigener Programmcode kombiniert werden, um komplette Anwendungen zu kreieren.

Ebenso können Aufgaben, die oft wiederholt werden müssen, mit einfachen MEL-Scripts automatisiert werden. Exaktes Arbeiten gelingt meist mittels MEL-Commands schneller und besser als mit der Benutzeroberfläche. Maya kann auf jeder der drei Ebenen in Abbildung 5.1 beliebig erweitert werden. Sogar Erweiterungen des *Dependency Graphs (DG)*[3] sind möglich und lassen sich nahtlos in Maya einbinden. Als Maya-Benutzer oder Programmierer hat man zwei Möglichkeiten, Maya nach seinen Bedürfnissen zu gestalten und zu erweitern: MEL oder C++. In beiden Fällen bietet Maya eine sehr gute Dokumentation, die ganz einfach in Maya im Menü *Help* zu finden ist. Vertiefende Informationen zur Programmierung in MEL sind in [15] zu finden. Für die Maya C++ API ist [4] der richtige Einstieg.

MEL:

MEL hat eine C-ähnliche Syntax und ist eine speziell für das Arbeiten in Maya maßgeschneiderte Programmiersprache. Aufgrund der etwas einfacheren Syntax und Struktur ist MEL bei den Maya-Benutzern weiter verbreitet als C++. MEL ist eine „interpretierte“ Sprache, das bedeutet, dass sie vor der Ausführung nicht kompiliert werden muss, sondern sofort eingesetzt werden kann. MEL kann komplett in Maya geschrieben, debugged und getestet werden. Es sind keine zusätzlichen Programme oder Compiler für die Entwicklung eigener Programmteile notwendig. Andererseits ist MEL dadurch um einiges langsamer als ein Programm in C++.

C++:

Mit C++ sind die Möglichkeiten, Maya zu erweitern, beinahe unerschöpflich. Die Plugins können nahtlos in Maya integriert werden. Mit Hilfe der Maya C++ API kann auf alle Maya-Klassen und -Objekte zugegriffen werden.

Die Entscheidung zwischen diesen beiden Möglichkeiten ist abhängig von den speziellen Anforderungen, die an den neuen Programmteil gesetzt werden. In den meisten Fällen reichen die Funktionen, die von MEL geboten werden, vollkommen aus. Ist aber eine besonders hohe Leistung gefragt, oder werden Funktionen benötigt, die nicht in der MEL-Schnittstelle enthalten sind, so wird die Wahl auf eine Lösung mit C++ fallen. Das bedeutet aber nicht, dass

[3] Der Dependency Graph (Szenengraph) ist das Herzstück von Maya, die gesamte Szene – alle Elemente (3D-Objekte, Befehle, ...) – sind im Szenengraph enthalten. Der DG wird noch im Verlauf dieses Kapitels erklärt.

die C++-Schnittstelle eine Obermenge der MEL-Funktionen bildet, sondern es kann durchaus sein, dass einige Funktionen, die in der MEL-Schnittstelle enthalten sind, nicht in der C++ API zu finden sind und umgekehrt.

Ein weiteres Entscheidungskriterium kann natürliche auch die Programmiererfahrung des Maya-Benutzers sein. Die MEL-Syntax ist zwar der C-Syntax relativ ähnlich, jedoch wurden Zeiger oder Speicherallokierungen entfernt, was zwar einerseits die Möglichkeiten einschränkt, andererseits die Programmiersprache leichter lesbar und verständlicher macht. MEL ist ideal für das schnelle Entwickeln von Prototypen, da beim Erstellen der Scripts nicht auf Datentypen geachtet werden muss. Dieser Umstand kann einerseits ein Vorteil, andererseits ein Nachteil sein, z. B. wenn ein gefinkelter Bug[4] auftritt, der nachher nur schwer zu finden ist. Man kann allerdings den Datentypen bei der Variablendeklaration angeben.
Soll ein Maya-Plugin plattformunabhängig sein, so stellt dies mit der Verwendung von MEL kein Problem dar. Mit C++ ist das ungleich komplizierter, da die unterschiedlichen C++ Compiler meist nicht kompatibel sind. Oft werden auch unterschiedliche C-Librarys verwendet. Beschränkt man sich nur auf die Maya C++ Klassen und Bibliotheken und verwendet nur wenige bis keine externen Bibliotheken, sollte das Erreichen einer Plattformunabhängigkeit erleichtern. Möchte man jedoch eigene Knoten (*Nodes*) und Befehle (*Commands*[5]) implementieren, so ist man gezwungen, die C++ API zu verwenden. Die Gestaltung von individuellen Benutzeroberflächen ist hingegen nur mit MEL möglich. Als Programmierer wird man somit höchstwahrscheinlich beide Schnittstellen verwenden müssen.

Verwendet man Maya und interagiert mit der Benutzeroberfläche, so werden im Hintergrund MEL-Befehle an die *MEL Command Engine* geschickt. Dort werden sie interpretiert und ausgeführt. Maya kann auch im *Batch*-Modus ausgeführt werden, in dem keine Benutzeroberfläche vorhanden ist. Hier kann man die MEL-Befehle direkt eingeben. Die meisten MEL-Befehle wirken direkt auf den Dependency Graph, welcher – vereinfacht gesagt – die komplette Szene enthält. Im DG ist nicht nur definiert, welche Daten und Objekte in der Szene enthalten sind, sondern auch noch, wie die Szene aufgebaut ist, wie die Objekte zusammenhängen und wie die Daten verarbeitet werden.

5.2.2 Dependency Graph

Jeder Teil einer Maya-Szene, egal ob 3D-Objekt, Animation, Licht, Textur oder sogar Befehl wird als ein oder mehrere Knoten im Dependency Graph dargestellt. Diese Knoten werden somit auch als „DG Nodes“ bezeichnet. Jeder Knoten enthält einige Attribute. Diese speichern die Charakteristika des Knoten. So enthält zum Beispiel ein Knoten „Würfel“ die Attribute „Höhe“,

[4]Fehler, Programmierfehler in einer Software
[5]Nodes und Commands werden im Laufe dieses Kapitels noch erklärt.

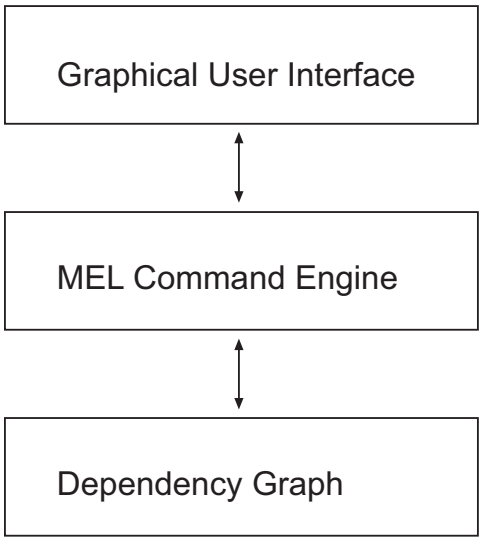


Abbildung 5.1: In dieser Grafik ist der Aufbau von Maya aus Sicht des Benutzers dargestellt. Der Benutzer kann auf alle diese Ebenen einwirken, sie verändern und sie auf seine Bedürfnisse einstellen.

„Position“, ...
All diese Knoten ergeben mit ihren Verbindungen (Connections) den Dependency Graph oder auch Szenengraph (Abbildung 5.2).

Entwickelt man nun ein Plugin für Maya, so bewegt man sich in einem engen Korsett. Jeder Knoten besteht aus den Input-Attributen und den Output-Attributen, die von der `compute`-Funktion ermittelt werden. Die `compute`-Funktionen sind sozusagen die „Gehirne“ im Szenengraph. Verändern sich die Input-Attribute werden die Output-Attribute von der jeweiligen `compute`-Funktion berechnet. Dies geschieht aber auch nur dann, wenn das Output-Attribute mit irgendeinem anderen Attribut in der Szene verbunden ist. Ist das nicht der Fall, wird Rechenzeit und -leistung gespart. Auch wenn ein Knoten nur Attribute speichert und diese nicht verändert, so muss dieser über eine `compute`-Funktion verfügen. Die Attribute eines Knotens definieren eine Datenschnittstelle für alle anderen Knoten im Szenengraph. Einzig durch diese Schnittstelle ist ein Datenaustausch der Knoten möglich. Durch sogenannte Plugs, welche die Daten der Attribute beinhalten, können die Attribute miteinander verbunden werden. Jeder Datenzugriff funktioniert über Plugs. Die Attribute definieren lediglich den Datentyp und den Namen des Attributs.
Jeder Befehl und jeder Knoten, die entwickelt werden, müssen weiters Mayas „Undo“ und „Redo“-Mechanismen unterstützen. So ist garantiert, dass dem

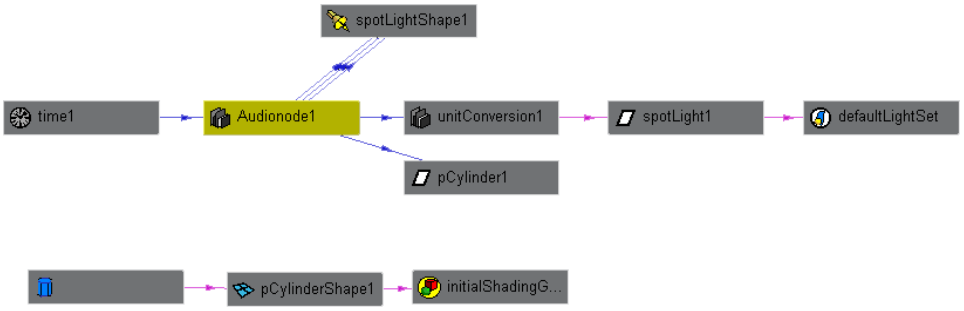


Abbildung 5.2: Der Szenengraph enthält alle in der Szene enthaltenen Knoten und ihre Verbindungen. In diesem Beispiel werden eine Lichtquelle und ein Zylinder vom *Audionode* gesteuert.

Benutzer die volle Funktionalität von Maya – auch bei selbst geschriebenen Plugins und Skripten – erhalten bleibt. Befolgt der Benutzer die Regeln, so lässt sich das Plugin vollständig in die Programmstruktur einbinden.

Die Szene:

Die Szene enthält alle Modelle, ihre Animation, Texturen, sowie Lichter, Kameras und andere Informationen (wie Renderoptionen, ...). In Maya ist der Dependency Graph die Szene! Der Szenengraph definiert die gesamte Szene durch das Netz verbundener Knoten. Der Aufbau wird nicht – wie in anderen 3D-Programmen – in einer seperaten Datenstruktur gespeichert, sondern die Daten befinden sich direkt in den Knoten. Da jederzeit Knoten hinzugefügt und entfernt werden können, entsteht eine sehr dynamische Datenstruktur. Glücklicherweise übernimmt Maya die interne Organisation des Dependency Graphs, sodass der Benutzer durch bequeme Funktionen (z. B. „alle Lichter auswählen“, „alle Objekte mit der Textur X“, ...) von der unterliegenden Struktur verschont bleibt.

Datenfluss:

Anhand der Grafik 5.2 kann der Datenfluss recht gut beschrieben werden. Man sieht die ein- und ausgehenden Verbindungen zum Knoten *Audionode*. Der Eingang des gelb gekennzeichneten Audionode ist mit dem *Timenode* verbunden. Ändert sich die Zeit, so wird die Änderung an den Audionode weitergeleitet, dieser berechnet seine Ausgangswerte neu. Diese Veränderungen wiederum werden dann entlang der Verbindungen weitergeleitet, sodass sich die nachfolgenden Knoten ebenso ändern.

5.2.3 Nodes und Commands

Mit Hilfe der C++ API kann der Maya Benutzer eigene Commands und Nodes erzeugen. Commands werden genutzt, um bestimmte Operationen auszuführen, welche die Szene verändern. So ist beispielsweise der Befehl `sphere` dazu da, Kugeln zu erzeugen und zu editieren. Jeder Befehl kann in verschiedenen Modi (`create, edit, query`) aufgerufen werden. Darauf muss der Programmierer bei der Implementierung eines Commands achten. Mehr dazu aber im Kapitel 6. Knoten hingegen sind meist komplexer als Commands, da sie Daten speichern können. Die meisten 3D-Objekte bestehen aus mehreren Knoten. So besteht jedes sichtbare Element in Maya zumindest aus einem Knoten für die Transformation und einem Knoten für die Form. Jeder Node wiederum umfasst ein oder mehrere Eigenschaften, welche in Maya als „Attribute“ bezeichnet werden.

Attribute:

Jeder Knoten kann ein oder mehrere Attribute beinhalten. Die Attribute bestehen aus Namen und Datentyp. Mittels Punktoperator kann auf die Attribute zugegriffen werden (z. B. `pSphere1.radius = 5;`). Ein Attribut kann aus allen üblichen Datentypen bestehen, wie beispielsweise `int, boolean, byte, char, short, long, float, double` ....
Ebenso können kompliziertere Datentypen verwendet werden. Die Bandbreite reicht von einfachen Arrays über Vektoren bis zu komplexen Geometrien. Diese Datentypen sind auch beliebig kombinierbar, so können mit „Eltern-Kind-Beziehungen“ (Parent-Child), komplexe Datentypen erstellt werden. Zum Beispiel: `Punkt A` aus den „Kinder-Attributen“ `float x`, `float y`, `float z`.

***compute*-Funktion:**

Die `compute`-Funktion kann als Gehirn des Knotens bezeichnet werden. Die Attribute beinhalten die Daten, aber all die Berechnungen werden in der `compute`-Funktion durchgeführt. Da sowohl die Eingabe- als auch die Ausgabeattribute beinahe jede Form annehmen können, kann auch in dieser Funktion alles mögliche kalkuliert werden. Die Funktion kann beispielsweise eine NURBS[6] -Fläche als Eingangsparameter haben und aus ihr eine neue NURBS-Fläche berechnen. Dabei ist besonders wichtig, dass die `compute`-Funktion nur Daten aus den Eingabeattributen verwendet und keine Informationen von anderen Stellen verwendet! Dieser Umstand muss auf jeden Fall berücksichtigt werden. Nur so kann garantiert werden, dass jeder Knoten mehr oder weniger als „Black Box“ funktioniert. Der Benutzer muss nicht

[6] **N**on **U**niform **R**ational **B**asis **S**plines sind mathematisch definierte Kurven oder Flächen.

wissen, wie die Ausgabe generiert wird. Er soll einfach die Attribute verbinden. Es ist auch erlaubt, dass eine `compute`-Funktion nichts berechnet. Am Beispiel des *Timenodes* in Maya sieht man, dass es auch gültig ist, wenn ein Knoten nur Attribute speichert und nichts neu berechnet wird. Der Timenode verfügt nur über ein Ausgabeattribut, in dem die aktuelle Zeit aus der Zeitleiste gespeichert wird. Sind nun viele Knoten miteinander verbunden, entsteht ein „neurales“ Netzwerk.

Beim Entwickeln eines Knoten muss der Programmierer darauf achten, dass der Knoten eine eindeutige Identität (ID) erhält. Zum Testen und Entwickeln können Identitäten von 0 bis 0x7FFFF verwendet werden. Möchte man ein Plugin veröffentlichen, so kann man bei *Alias | Wavefront* eindeutige, öffentliche IDs anfordern. Detailliertere Infos sind in [4] zu finden.

5.2.4 Maya und Microsoft VisualStudio.NET

Wenn bei der Installation von Maya das Microsoft VisualStudio.NET bereits installiert war, so sollte der Maya Plugin-Wizard automatisch mitinstalliert worden sein. Dieser erleichtert das Erstellen eines Maya-Plugins sehr, da man durch einfaches Durchklicken die wichtigsten Grundeinstellungen des VisualStudio.NET-Projekts rasch einstellen kann. Der Maya Plugin-Wizard lässt sich beim Erstellen eines neuen VisualStudio.NET-Projekts aus der Liste der möglichen Projektarten auswählen. Installationsanleitungen für aktuelle Maya- und VisualStudio.NET-Versionen können unter www.davidgould.com nachgeschlagen werden.
Beim Debuggen ist darauf zu achten, dass Maya nicht geöffnet ist. Hat man in Microsoft VisualStudio.NET den Debug-Modus eingestellt und beginnt zu debuggen, so wird man aufgefordert, eine ausführbare Datei anzugeben. Das geschieht normalerweise nur beim ersten Mal. In diesem Fall muss man die Datei *maya.exe* (meist im *bin*-Unterverzeichnis des Maya-Installationsverzeichnis) angeben. Dann öffnet sich Maya, man wählt über den *Plugin Manager* das Plugin (Wichtig: Die Debug-Version!) aus und kann damit arbeiten. Kommt das Programm zu einem Breakpoint[7], wird Microsoft VisualStudio.NET wieder eingeblendet und man kann – wie bei jedem anderen Programm – problemlos debuggen.
Syntax-Beispiele und Details über den genauen Aufbau eines Plugins, das sowohl Befehle als auch Knoten erzeugt, sind im folgenden Kapitel 6 nachzulesen.

[7]Beim Debuggen (Fehlersuchen) kann man Punkte definieren, an denen das Programm anhalten soll.

Kapitel 6

Audioanalyse-Plugin für Maya

Dieses Kapitel zeigt die Implementierung eines Maya-Plugins konkret am Beispiel des Audioanalyse-Plugins. Das Plugin umfasst die Benutzeroberfläche, welche in MEL implementiert wurde, sowie den Befehl `analyzesound`, welcher den `Audionode` erzeugt, und den eben genannten Knoten, der die Audioanalyse übernimmt. Der gesamte Quelltext ist auf dem beiliegenden Datenträger nachzulesen. In diesem Kapitel werden nur kurze aber wesentliche Code-Ausschnitte angeführt, wobei „Sicherheitsüberprüfungen", wie sie im Plugin vorkommen, aus Gründen der Lesbarkeit weggelassen wurden.

6.1 Funktionen und Aufbau

Mit Hilfe des im Zuge der Diplomarbeit entwickelten Plugins kann der Maya-Benutzer in einigen wenigen Schritten audiogesteuerte Animationen erstellen. Dazu wird ein Musikstück im Audionode analysiert und die resultierenden Daten (Amplitude links und rechts, Spektrumdaten, Auswahl von Features) an den Ausgabeattributen des Knotens im Szenengraph zur Verfügung gestellt. Aufgrund der beschränkten Dateiformat-Unterstützung von Maya wird eine Audiolibrary verwendet, mit der alle gängigen Dateiformate problemlos in einen Datenstrom verwandelt werden können, der dann vom Audionode analysiert wird.
Der Benutzer kann dann beliebige Knoten, wie Objekte, Texturen, Lichter, etc. mit den Ausgabeattributen des Audionodes verbinden. Das funktioniert mit allen in Maya zur Verfügung stehenden „Verbindungseditoren"[1]. In Abbildung 5.2 ist eine Szene mit eingebundenem Audionode dargestellt.
Die Möglichkeit, die Analysedaten in eine externe Datei zu schreiben, ist ein

[1] Maya bietet viele Möglichkeiten, um Knoten miteinander zu verbinden → *Hypergraph, Expression Editor, Connection Editor, Command Line, ...*

weiterer Zusatznutzen. So können mehrere Benutzer an einem Projekt arbeiten, ohne die Audiodatei(en) austauschen zu müssen. Diese Datei enthält zwar keine Musik, aber für jeden Frame die Analysedaten. Der Empfänger muss dann nur den Audionode in der Szene auf den Modus *read from file* stellen, den Pfad der Datei im Audionode-Attribut ändern und kann dann mit den Analysedaten – soweit vorhanden – weiterarbeiten.

6.1.1 Initialisierung des Plugins

Wie bereits im Kapitel 5 kurz beschrieben muss eine Maya Linked Library (.mll) dem ausführenden Programm einen Ein- und Ausstiegspunkt bieten. Aus diesem Grund müssen die beiden nachfolgenden Funktionen implementiert sein, damit Maya auf die Klassen des Plugins zugreifen kann. Wird das Plugin in Maya mit dem Plugin Manager geladen, so wird die Funktion `MStatus initializePlugin(MObject obj)` aufgerufen. In dieser Funktion werden sämtliche im Plugin enthaltenen, vom Benutzer implementierten „neuen“ Maya-Befehle und Knoten registriert. Wie aus dem Quelltext zu erkennen ist, wird in diesem speziellen Plugin ein Maya-Befehl `analyzesound` sowie ein Maya-Knoten `Audionode` zur Verfügung gestellt. An dieser Stelle kann auch die Audiolibrary initialisiert werden, sodass beim Laden des Plugins alle benötigten Ressourcen vorhanden sind.

```
//Datei PluginMain.cpp
#include <maya/MFnPlugin.h>
#include <maya/MGlobal.h>

MStatus initializePlugin( MObject obj )
{
   MFnPlugin pluginFn(obj, "Bindreiter", "1.0", "any");
   pluginFn.registerCommand("analyzesound", AnalyzeCmd::creator,
     AnalyzeCmd::newSyntax);
   pluginFn.registerNode("Audionode", Audionode::id,
     Audionode::creator, Audionode::initialize);

   //initialize Audiolibrary
   [...]
}
```

Wird das Plugin wieder entladen, wird die Funktion
`MStatus uninitializePlugin(MObject obj)` aufgerufen, in der alle Registrierungen rückgängig gemacht werden und allokierter Speicher freigegeben werden kann.

```
MStatus uninitializePlugin ( MObject obj )
{
   MFnPlugin pluginFn(obj);
   pluginFn.deregisterCommand("analyze");
   pluginFn.deregisterCommand(Audionode::id);

   //uninitialize Audiolibrary
   //free memory
   [...]
}
```

6.1.2 AnalyzeCmd

In der Klasse `AnalyzeCmd` wird der neue Maya-Befehl `analyzesound` implementiert. Ist das Plugin geladen, kann mit Eingabe des Befehls in die Befehlszeile (Command Line) von Maya der Audionode erzeugt werden. Damit dieser Befehl nahtlos in Maya eingebunden werden kann, müssen einige Kriterien, wie beispielsweise *undo* und *redo*, unbedingt erfüllt werden. Der Maya-Benutzer soll also problemlos den Befehl rückgängig machen oder wieder herstellen können. Zusätzlich ist es üblich, dass auch selbst erzeugte Befehle in verschiedenen Modi aufgerufen werden können. So bietet jeder sauber implementierte Befehl in Maya die Möglichkeit, einen kurzen Hilfetext aufzurufen. Diese Modi können durch Verwendung von Flags[2] aufgerufen werden. Für den Hilfeaufruf sind die Flags `-h` oder `-help` üblich. Alle vorhandenen Flags müssen der Befehlssyntax in der Funktion `MSyntax AnalyzeCmd::newSyntax()` hinzugefügt werden. Dieser Maya-Befehl bietet neben dem „Hilfe-Flag“ auch Flags für das Setzen des Audiodateipfades und das automatische Setzen der Zeitleiste in Maya auf die Länge der Audiodatei.
Die Klasse `AnalyzeCmd` wird von der Maya-Klasse `MPxCommand` abgeleitet. Folgende Funktionen müssen implementiert werden, um die geforderten Grundvoraussetzungen zu erfüllen:

```
virtual MStatus doIt(const MArgList &args);
virtual MStatus undoIt();
virtual MStatus redoIt();
virtual bool isUndoable()const{return false;}

static MSyntax newSyntax();
static void *creator();
```

In der `creator`-Funktion wird der Befehl erzeugt, in der bereits erwähnten

[2] Flags sind (in Maya) Anhängsel eines Befehls, mit denen zusätzliche Informationen abgefragt oder mitgegeben werden können. In Maya können auch mehrere Flags in der Form „-flag1 Wert -flag2“ angehängt werden.

`newSyntax`-Funktion werden alle erlaubten Flags angeführt. Der Quelltext zum Ausführen des eigentlichen Befehls ist in der `doIt`-Funktion enthalten. Um überhaupt auf die Maya-Szene zugreifen zu können, enthält die Klasse ein spezielles Objekt vom Typ `MDGModifier`[3]. Erst dieses Objekt ermöglicht das Verändern des Szenengraphen, also das Erstellen, Löschen und Verbinden von Knoten, und unterstützt die „undo/redo"-Funktionen. Dazu wird in der `redo`-Funktion einfach die Codezeile `MDGModifier dgMod.doIt();` ausgeführt und in der `undo`-Funktion die Codezeile `dgMod.undoIt();` ausgeführt.

Der Audionode wird in der `doIt`-Funktion erzeugt und in die Maya-Szene eingebunden. Das bedeutet, dass der Knoten mit dem Timenode in der Szene verbunden werden muss. Ohne Eingangsattribute wäre der Audionode nicht funktionstüchtig, da der Knoten erst durch eine Änderung der Eingangsparameter angeregt wird, seine Ausgangsattribute neu zu berechnen. Dazu muss nochmals erwähnt werden, dass in Maya getrennte Klassen für Daten und Funktionen verwendet werden. Die (Daten-)Objekte werden von der Maya-Klasse `MObject` abgeleitet, die Funktionsobjekte von der Klasse `MFn`. So ist im nachstehenden Quelltext zu erkennen, dass sowohl für den Audioknoten als auch für den Zeitknoten zuerst ein Datenobjekt erzeugt wird und anschließend ein Objekt, welches die Funktionen enthält. Den Funktionsobjekten werden die Datenobjekte übergeben und anschließend werden mit dem *Dependency Graph Modifier* die beiden Attribute – und somit auch die beiden Knoten – verbunden.

[3]Dependency Graph Modifier

```
//Klasse AnalyzeCmd
MStatus AnalyzeCmd::doIt(const MArgList &args)
{
   //flags
   [...]

   //create Audionode
   MObject audionodeObj = dgMod.createNode(Audionode::id,&stat);

   //retrieve time node
   MSelectionList selection;
   MObject timeObj;
   MGlobal::getSelectionListByName( "time1", selection );
   selection.getDependNode( 0, timeObj );

   /*use MFnDependencyNode to access
   node plugs ready for connection*/
   MFnDependencyNode depFnTime( timeObj );
   MFnDependencyNode depFnAudio( audionodeObj );

   //connect timenode to audionode
   dgMod.connect( depFnTime.findPlug("outTime"),
   depFnAudio.findPlug("time") );

   //file-open dialog
   [...]
}
```

6.1.3 Audionode

Die Klasse `Audionode` wird von der Maya-Klasse `MPxNode` abgeleitet. Der Audionode ist das Kernstück des Plugins. Die `compute`-Funktion des Audionodes führt bei jeder Änderung der Eingangsattribute alle notwendigen Berechnungen durch und aktualisiert die Ausgangsattribute des Knotens. Neben einer Konstruktor- und Destruktor-Funktion muss jede Klasse, welche die Implementierung eines Maya-Knotens enthält, über eine Initialisierungsfunktion verfügen. In dieser werden alle Eingangs- und Ausgangsattribute sowie die jeweiligen Abhängigkeiten der Ausgangsattribute von den Eingangsattributen erstellt. Wie im ersten der drei nachfolgenden Quelltextausschnitte zu sehen ist, enthält der Audionode einige unterschiedliche Attributtypen. Für einfache Datentypen (int, float, double, ...) werden dem Knoten Attribute vom Typ `MFnNumericAttribute` hinzugefügt, sind die Attribute komplexer, wie beispielsweise Zeichenketten, Kurven, geometrische Objekte usw., dann werden Attribute der Typen `MFnData` und `MFnUnitAttribute` verwendet. In der Initialisierungsfunktion werden im ersten Schritt alle Attribute erzeugt, Kurz- und Langbezeichnung, Typ des Attributs und Standardwerte angegeben. Weiters können zusätzliche Charakteristika des

Attributes (storable, writeable, ...) eingestellt werden. Bei Eingangsattributen ist es nicht notwendig, dass diese speicherbar (storable) sind. Bei Ausgangsattributen wiederum kann darauf verzichtet werden, dass diese vom Benutzer umgeschrieben werden können, da diese ohnehin aus den Eingangsattributen berechnet werden sollen.

```
//Klasse Audionode
const MTypeId Audionode::id(0x00999);
MObject Audionode::a_timeIn;
MObject Audionode::a_fftOutput;
[...]

MStatus Audionode::initialize()
{
   //unit attribute to store time value
   MFnUnitAttribute uAttr;
   /*create a new unit attribute
   of type time and assign it to time*/
   a_timeIn =  uAttr.create( "time", "t",
      MFnUnitAttribute::kTime, 0.0, &stat );
   uAttr.setStorable(false);
   [...]

   //output attribute for fft data -> a_fftOutput
   a_fftOutput = nAttr.create( "fftOutput", "fftOut",
      MFnNumericData::kFloat, 0.0, &stat );
   //indicate that attribute is an array
   nAttr.setArray(true);
   nAttr.setUsesArrayDataBuilder(true);
   //make the attribute read only as it is the output
   nAttr.setStorable(false);
   nAttr.setWritable(false);
   [...]
```

Sind alle Attribute erzeugt, so müssen sie dem Audionode hinzugefügt werden. Weiters müssen die Abhängigkeiten der Attribute definiert werden. Diese geben an, wie sich Veränderungen eines Eingangsattributs auf die Ausgangsattribute auswirken. Jede Verbindung muss angeführt werden, damit der Knoten verlässlich funktioniert. Gezeigt wird nur ein kleiner Ausschnitt des Quelltexts:

```
   //add all attributes
   stat=addAttribute(a_timeIn);
   stat=addAttribute(a_fftOutput);
   [...]

   //set attribute affects
   attributeAffects(a_timeIn, a_fftOutput);
   [...]

   return MS::kSuccess;
} //end initialize
```

In der bereits erwähnten `compute`-Funktion werden die Ausgangsattribute berechnet. Aus dem Datenblock, der die gesamten Daten eines Knotens enthält, werden mittels sogenannter `MDataHandle`, welche die einzelnen Attribute referenzieren, die Eingangsattribute ermittelt. Dabei ist zu beachten, dass beim Setzen oder Holen von Attributdaten die Datentypen übereinstimmen.

```
//Klasse Audionode
MStatus Audionode::compute(const MPlug& plug, MDataBlock &data)
{
   //connect input and output attributes
   MDataHandle timeHnd=data.inputValue(a_timeIn, &stat);
   [...]

   //retrieve current time and other values
   MTime t = timeHnd.asTime();
   currentFrame = (int)t.value();
   [...]
```

Sind die Werte der Eingangsattribute ermittelt, so werden die abhängigen Ausgangsattribute neu berechnet. Dazu wird ebenfalls erst ein `MArrayDataHandle` erzeugt, um auf die Daten zugreifen zu können. Im speziellen Fall der FFT-Daten kommt zusätzlich ein `MArrayDataBuilder` zum Einsatz, ohne den das Schreiben der einzelnen Array-Elemente nicht möglich wäre. Nach Ermittlung der FFT-Daten mit der Funktion `getFFTDataFromStream`[4] werden für jedes Frequenzband neue `MDataHandle` hinzugefügt.

[4]Eine genauere Beschreibung dieser Funktion ist in Abschnitt 6.1.4 nachzulesen.

```
   //fft output
   if(plug == a_fftOutput)
   {
      MArrayDataHandle outputFFTHnd =
         data.outputArrayValue(a_fftOutput);
      MArrayDataBuilder outputArrayBuilder =
         outputFFTHnd.builder(&stat);

      //retrieve logical index of array
      int element = plug.logicalIndex(&stat);

      getFFTDataFromStream(decodechan, 1);
      //get live bass data
      for(int c = 0; c < bands; c++)
      {
         MDataHandle outputDataElem =
            outputArrayBuilder.addElement(c);
         outputDataElem.set(FFT_DATAOUT[c]*scalefft);
      }

      outputFFTHnd.set(outputArrayBuilder);
      is_FFT_performed = true;
      /* if other features need the fft data
       it has not to be recomputed*/

      return MS::kSuccess;
   }//end fftoutput
}
```

Wie bereits erwähnt, ist es möglich, den Audionode in verschiedenen Modi zu betreiben. Neben der Analyse einer Audiodatei können die Analysedaten zusätzlich in eine Datei geschrieben werden oder an Stelle der Analyse aus einer Datei ausgelesen werden. Dies ist in der `compute`-Funktion berücksichtigt und im Quelltext nachzulesen. Das Verbinden und Schreiben der Attribute funktioniert allerdings identisch, es werden nur zusätzlich zur – oder an Stelle der – FFT-Analyse die Daten in eine Datei geschrieben bzw. ausgelesen.

6.1.4 BASS Audiolibrary

BASS ist eine Audiolibrary für die Verwendung in Windows- und Mac OSX-Applikationen. Sie bietet einfache aber leistungstarke Sample-, Stream- und Recorderfunktionen für beinahe alle gängigen Dateiformate. Um bei der Feature Extraction nachvollziehbare Ergebnisse zu erhalten, werden in diesem Plugin allerdings nur Dateien mit einer Abtastrate von 44.1 kHz erlaubt.

In Windows benötigt BASS DirectX 3 oder höher für die Wiedergabe von Audio und nützt – falls vorhanden – Vorteile von DirectSound und Direct-

Sound3D Hardware-Beschleunigungs-Drivern. In Mac OSX verwendet BASS CoreAudio für die Wiedergabe, wobei Mac OSX 10.3 oder höher empfohlen wird. Sowohl PowerPC als auch Intel Macs werden unterstützt.

BASS unterstützt C/C++, Delphi und Visual Basic APIs. Die BASS Library ist für nichtkommerzielle Anwendungen lizenzfrei und gratis verwendbar. Mehr Informationen zum Leistungsumfang und zu weiteren Add-Ons sind unter http://www.un4seen.com/bass.html zu finden.

In diesem Plugin wird die Audiolibrary verwendet, um die ausgewählten Audiodateien wiederzugeben sowie die Daten zur Verfügung zu stellen, die zur Audioanalyse benötigt werden. Dazu wurden der Klasse `Audionode` die folgenden Funktionen hinzugefügt:

```
//Klasse Audionode
void Audionode::loadAudio(MString filename)
void Audionode::playAudio(DWORD handle, QWORD newpos)
void Audionode::stopAudio(int status)
void Audionode::getRawDataFromStream(DWORD handle)
void Audionode::getFFTDataFromStream(DWORD handle, int windowed)
void Audionode::getLevelsFromStream(DWORD handle)
```

In diesen Funktionen werden mit Hilfe der Audiolibrary die Audiodateien geladen, die rohen Audiodaten ermittelt und die Wiedergabe gesteuert. Die Funktion `loadAudio` wird immer dann aufgerufen, wenn sich das Eingangsattribut für den Dateipfad der Audiodatei ändert. Die beiden Funktionen zur Wiedergabe werden dann aufgerufen, wenn der Benutzer in Maya die Zeitposition verändert. Die letzten drei Funktionen werden bei jeder Zeitänderung von der `compute`-Funktion aufgerufen und ermitteln die Lautstärke des linken und rechten Kanals des Audiosignals sowie das Spektrum mittels FFT und die einfachen Amplitudenwerte des Signals. Die FFT-Blockgröße beträgt 2048 Werte, wobei dann 1024 Werte im Array `float FFT_DATA[1024]` gespeichert werden. Die Erfassung der rohen Sampledaten erfolgt ebenfalls mit einer Blockgröße von 2048 Samples für Mono- und 4096 Samples für Stereosignale, die dann in den Arrays `float AUDIO_DATA_MONO[2048]` oder `float AUDIO_DATA_STEREO[4096]` gespeichert werden. Die Lautstärken der Kanäle werden in den beiden `float`-Werten `LEFT_LEVEL` und `RIGHT_LEVEL` gespeichert. All diese Daten dienen als Grundlage zur Berechnung der Eigenschaften.

Die Befehle und Funktionen, die in der Audiolibrary für diese Aufgaben zur Verfügung stehen, können im Quelltext oder in der Hilfedatei, die beide auf dem beiliegenden Datenträger enthalten sind, nachgelesen werden.

6.2 Feature Extraction

Die Klasse `FeatureExtraction` enthält alle Funktionen zur Audioanalyse. Alle implementierten Eigenschaften wurden bereits in dieser Arbeit theoretisch beschrieben, hier wird genauer auf die Implementierung eingegangen. Durch die unterschiedlichen zeitlichen Auflösungen, die durch die Frameeinstellungen in Maya begründet sind, ist die Audioanalyse unter Umständen nicht ganz exakt. Bei einer Auflösung von 30 Frames pro Sekunde (NTSC) sind die Ergebnisse relativ exakt, da die Datenblöcke einander gut überlappen. Bei einer geringen Auflösung von 15 Frames pro Sekunde (GAME) überlappen die Blöcke nicht mehr so und die Audioanalyse ist daher wissenschaftlich nicht mehr korrekt. Am meisten ist die Beat Detection von diesem Problem betroffen, obwohl die wissenschaftliche Korrektheit bei Animationen in diesem Fall vermutlich ohnehin nicht im Vordergrund stehen wird.

6.2.1 Implementierung der zeitbasierten Eigenschaften

Für die Implementierung der zeitbasierenden Eigenschaften werden die rohen Audiodaten aus den `AUDIO_DATA`-Arrays verwendet.

Short Time Energy

Hier wird aus den Audiodaten für jeden Analysezeitpunkt ein Energiequadrat ermittelt. Bei Stereosignalen wird der Durchschnitt von linkem und rechtem Signal verwendet und aus diesen Werten die *Short Time Energy* berechnet.

```
/** short-time energy
 * out: float square of energy
 * in: float[] audio-data
 */
float FeatureExtraction::shortTimeEnergy(float audiodata[], int size)
{
   float sum = 0.0;
   if(Audionode::getStereoInfo()==1)
   {
      for(int i =0;i<size-1; i+=2)
      {
         sum+=(audiodata[i]+audiodata[i+1])/2;
         //average of left and right channel
      }
   }
   [...] //mono
   sum *= sum;
   return sum;
}
```

Zero Crossing Rate

Diese Funktion gibt die Zahl der Vorzeichenwechsel der Sampledaten für einen bestimmten Zeitraum zurück. Bei Stereosignalen wird nur der linke Kanal für die Analyse verwendet.

```
/** zero crossing rate
 * out: int -> zerocrossings
 * in: float[] audio-data, size of audiodata
 */
int FeatureExtraction::zeroCrossingRate(float audiodata[], int size)
{
   int zcr =0;
   [...] //stereo
   else if(Audionode::getStereoInfo()==0)
      for(int i=1; i<size; i++) //i from 1 end
      {
         /*current value is positive and
          value before was negative*/
         if(audiodata[i]>0 && audiodata[i-1]<0)
            zcr++;
         /*current value is negative and
          value before was positive*/
         else if(audiodata[i]<0 && audiodata[i-1]>0)
            zcr++;
      }
   return zcr;
}
```

6.2.2 Implementierung der spektralen Eigenschaften

Als Implementierungsgrundlage der folgenden Features dienen die Resultate der FFT. Diese spektralen Analysewerte werden zusätzlich in einem Array `float FFT_DATAOUT[32]` in 32 Bändern zusammengefasst, dass dem Plugin-Benutzer in Maya ebenfalls zur Verfügung steht.

Fundamental Frequency Estimation

Im Gegensatz zur bereits beschriebenen Eigenschaft Fundamental Frequency wird nur eine Annäherung der Grundfrequenz ermittelt. Dazu werden aus den FFT-Daten die lokalen Maxima ermittelt und der niedrigste Maxima-Index im Array gesucht. Die Grundfrequenz ergibt sich dann aus der Auflösung der FFT und dem Index. Bei einer Abtastrate von 44.1 kHz und einer FFT-Auflösung von 2048 Werten sind die ersten 1024 Indexwerte, die den Bereich bis 22050 Hz repräsentieren, relevant. Jeder Indexwert steht für ein Frequenzband mit einer Breite von $\Delta f = 22050/1024$, also ca. 22 Hz. Der

Rückgabewert ist vom Typ `int` und stellt eine Annäherung der Grundfrequenz in Hz dar. Der resultierende Fehler hängt also von der Auflösung der FFT ab und ist in diesem Fall $\leq 11\ Hz$. Das Ergebnis könnte optimiert werden, wenn der Phasengang des Spektrums berücksichtigt würde.

```
/** getFundamentalFreqencyEstimation
 * out: int estimation of fundamental frequency
 * in: float[] &fft-data, float threshold
 */
int FeatureExtraction::getFundamentalFrequencyEstimation
(float fftdataIn[], int arraysize, float threshold)
{
   /*bass library liefert bei einer fft mit
    2048 werten ein array mit 1024 zurück
    untersucht wird immer nur die untere hälfte
    des spektrums also fft.length/2
    abtastfrequenz ist bei der initialisierung
    der audio library mit 44.1kHz angenommen*/
   int fftlength = arraysize;
   int fftlength2 = fftlength/2;
   int bin = 44100/fftlength;

   //finds local maxima...
   findLocalMax(fftdataIn, fftlength2);

   std::vector<int> idxthres;
   float max=0.0;
   /*length of array frequencies with
    indices of maxima in spectrum*/
   int freqlength = sizeof(frequencies)/sizeof(int);

   //find max
   for(int i=0; i<freqlength; i++)
      if(fftdataIn[frequencies[i]]>max)
         max=fftdataIn[frequencies[i]];

   for(int i=0; i<freqlength; i++)
      if(fftdataIn[frequencies[i]]-max>-threshold)
      //list all indices with values higher than treshold
         idxthres.push_back(frequencies[i]);

   int maxfftbin = idxthres.at(0);
   /*return estimated fundamental frequency
    of bin with highest value*/
   return maxfftbin*bin;
}
```

Spectral Centroid

Der `float` Rückgabewert dieser Funktion beschreibt ungefähr den Index des Schwerpunktes des Spektrums. Die Frequenz in Hz kann wie bei der Fundamental Frequency Estimation ermittelt werden.

```
/** spectralCentroid
 * out: float spectral centroid
 * in: float[] fft-data
 */
float FeatureExtraction::
spectralCentroid(float fftdata[], int arraysize)
{
   float summagnitude = 0.0;
   float sum = 0.0;
   for(int i=0; i<arraysize; i++)
   {
      summagnitude += i*fftdata[i];
      sum += fftdata[i];
   }
   return summagnitude/sum;
}
```

Spectral Fluctuation

Diese Funktion gibt einen `float`-Wert zurück, der eine einfache Aufsummierung aller Differenzen der FFT-Daten darstellt:

$$\sum_{i=0}^{1024} |X_t[i] - X_{t-1}[i]|$$

Es werden also die Differenzbeträge addiert.

```
/** spectralFluctuation
 * out: float -> spectral fluctuation
 * in: float[] fft-data
 */
float FeatureExtraction::
spectralFluctuation(float fftdata[], int fftsize)
{
   fft_flux_new = new float[fftsize];
   arrayCopy(fftdata, fftnew);
   float flux = 0.0;
```

```
      for(int i=0; i<fftsize; i++)
      {
         float val = abs(fftnew[i])-abs(fftold[i]);
         if(val>0)
            flux+=val;
         else
            flux+=-val;
      }

      arrayCopy(fftnew, fftold);
      delete [] fft_flux_new;
      fft_flux_new = NULL;
      return flux;
   }
```

Spectral Rolloff

Der Rückgabewert dieser Funktion gibt die Annäherung jener Frequenz in Hz an, bei der die Energie der tieferen Frequenzen 85 % der gesamten Energie des Signals ausmacht. Der Wert ist vom Typ `float`.

```
/** spectral rolloff
 * out: int estimated frequency,
 * where lower frequencies own "percentage"
 * of total energy
 * in: float[] fftdata, int percentage
 */
float FeatureExtraction::
spectralRolloff(float fftdata[], int percentage, int fftsize)
{
   float frequency = 0.0;
   fft_new = new float[fftsize];
   arrayCopy(fftdata, fftnew);
   float totalenergy = 0.0;
   for(int i=0; i<fftsize; i++)
      totalenergy+= abs(fftnew[i]);

   float sum = 0.0;
   int bin=0;
   while(bin<fftsize && sum<totalenergy*(0.01*percentage))
   {
      sum+=abs(fftnew[bin]);
      bin++;
   }

   frequency = (float)bin*44100/fftsize;
   delete [] fft_new; //delete new fft
   fft_new = NULL;
   return frequency;
}
```

Spectral Bandwidth

Die spektrale Bandbreite berechnet sich aus den FFT-Daten und aus dem Spectral Centroid und der Rückgabewert vom Typ bewegt sich zwischen zwischen 0 und $N/2$, wobei N die Länge des FFT-Arrays darstellt. Je kleiner der Wert, desto schmäler ist die Energieverteilung um den Spectral Centroid im Spektrum. Ein großer Wert deutet darauf hin, dass im Spektrum die Energien relativ breit verteilt sind.

```
/** spectral bandwidth
 * out: float bandwith in kHz
 * in: float[] fftdata, float spectrumCentroid
 */
float FeatureExtraction::
spectralBandwidth(float fftdata[], float SC, int fftsize)
{
   float bandwidth = 0.0;
   float divisor = 0.0;
   float dividend = 0.0;
   for(int i=0; i<fftsize; i++)
   {
      divisor+=fftdata[i]*fftdata[i];
      dividend+= (i-SC)*(i-SC) * fftdata[i]*fftdata[i];
   }
   bandwidth = sqrt(dividend/divisor);
   return bandwidth;
}
```

6.2.3 Implementierung der Beat Detection

Die Beat Detection, die in diesem Plugin implementiert wurde, ist eine Variante des in Kapitel 4 beschriebenen „frequency selected sound energy"-Algorithmus. Dabei werden in einem Buffer mit einer Länge von zwei Sekunden die Quadrate der Summen (E_t) der FFT-Werte gespeichert. In einem weiteren Buffer werden die Differenzen der einzelnen Energiequadrate ($D_t = E_t - E_{t-1}$) gespeichert. Positive Differenzen deuten auf eine ansteigende Flanke hin, negative Differenzen werden ignoriert und auf 0 gesetzt. Zwei weitere Werte, die zur Beatdetektion herangezogen werden, sind die mittleren Abweichungen (V_E und V_D) der Bufferwerte vom jeweiligen Durchschnitt (ϕE oder ϕD). Ob nun ein Beat registriert wird oder nicht, lässt sich wie folgt darstellen:

$$BEAT = \{ \begin{array}{ll} 0, & E_t < (\phi E + S_t) || E_t < E_{t-1} || (D_t == 0 \&\& D_{t-1} == 0) \\ 1, & sonst. \end{array} \tag{6.1}$$

wobei durch $S = S_B * V_E/100$ bestimmt wird, wie weit die Abweichung berücksichtigt wird.

Mit den Eingangsattributen `beatlow` und `beathigh` kann der Benutzer den zu analysierenden Frequenzbereich von unten und oben einschränken. Das Attribut `beatsensitivity` (S_B) ermöglicht es dem Benutzer den Wert zu beeinflussen, der bestimmt, wie sensibel der Algorithmus auf neue Energiespitzen reagiert.

```
/** beat detection
 * out: bool -> beat yes/no
 * in: float[] fftdata, int max freq, int min freq, int userpart
 */
int FeatureExtraction::
beatDetection(float fftdata[],int maxfreq, int minfreq, int userp)
{
   [...] //check of inputs
   /*get averages of buffers (beat & derivation)
    get mean variances
    get sensitivity*/
   BEAT_AVERAGE = getBeatAverage(BEAT_BUFFER);
   BEAT_AVERAGED = getBeatAverage(BEAT_BUFFERDERIVE);
   BEAT_MEANVARIANCE =
      getBeatVariance(BEAT_BUFFER, BEAT_AVERAGE);
   BEAT_MEANVARIANCED =
      getBeatVariance(BEAT_BUFFERDERIVE, BEAT_AVERAGED);
   BEAT_SENSITIVITY =
      setBeatSensitivity(BEAT_MEANVARIANCE, BEAT_AVERAGE, userp);
   float lastElem = getBeatLastElement(BEAT_BUFFER);
   float lastElemd = getBeatLastElement(BEAT_BUFFERDERIVE);

   //get actual energy value
   float sum = 0.0;
   for(low; low<high; low++)
      sum+=fftdata[low];

   sum *=sum; //get square of energy
   float derive = sum - lastElem;
   if(derive<0.0) //use only ascending part
      derive = 0.0;
   derive *=derive; //square of derivation

   [...] //update buffers

   int beat = 1;
   if(sum<(BEAT_SENSITIVITY+BEAT_AVERAGE) ||
      sum<lastElem || (derive==0&&lastElemd==0))
      beat = 0;

   return beat;
}
```

6.3 Benutzeroberfläche in MEL

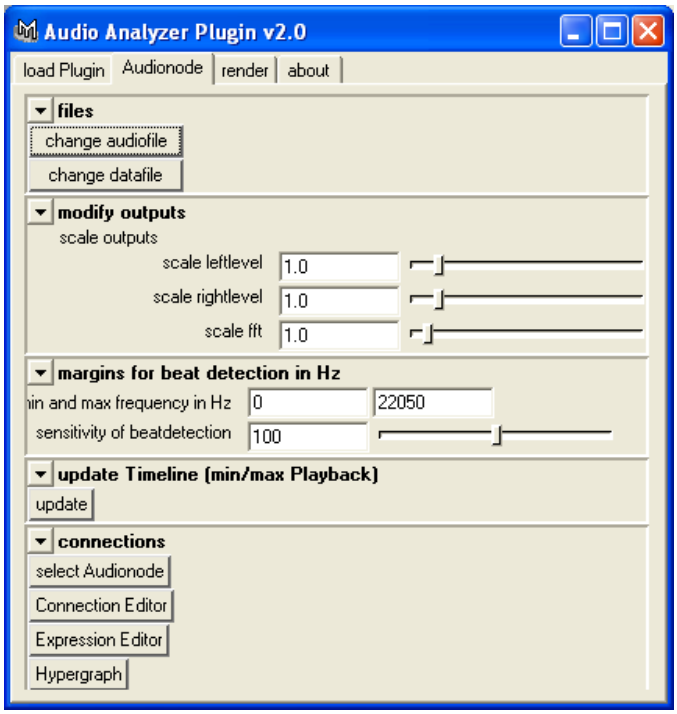


Abbildung 6.1: In dieser Grafik ist die Benutzeroberfläche des Audioanalyse-Plugins zu sehen. Das Plugin wurde mit der Maya C++ API und der Audiolibrary BASS implementiert, das Menü wurde in MEL programmiert.

Um dem Benutzer den Umgang mit dem Plugin zu erleichtern, wurde eine Benutzeroberfläche erstellt. Für eine – auch optisch – nahtlose Einbindung des Plugins in Maya bietet die Skriptsprache MEL umfangreiche Möglichkeiten zur Gestaltung von Benutzeroberflächen. Eine Vielzahl von Interaktionsmöglichkeiten (Buttons, Slider, . . .) können in das Interface eingebaut werden und für das Arrangement dieser Objekte werden verschiedene Layouts zur Verfügung gestellt. Der Quelltext für die Benutzeroberfläche in Abb. 6.1 befindet sich in einer Textdatei mit der Dateiendung `.mel`. Befindet sich die Datei im Verzeichnis für Maya-Skripten[5], so können die darin enthaltenen globalen Funktionen von Maya aufgerufen werden. Das Interface dieses Plugins wird in der Funktion `analyzerGUI` erstellt und kann auch mit Eingabe des Funktionsnamen in der Command Line ganz einfach aufgerufen werden.

[5] ../Eigene Dateien/maya/6.5/scripts

Die Benutzeroberfläche selbst besteht aus einem „Tab“-Layout[6]. Im ersten Tab kann der Benutzer das eigentliche Plugin mit dem Plugin-Manager laden, damit die Funktionen des Plugins in den Maya-Szenen zur Verfügung stehen. Dann kann ein neuer Audionode erzeugt oder eine bereits bestehende Datei, die schon einen Audionode enthält, geöffnet werden. Im zweiten Tab sind alle nötigen Bedienungselemente enthalten, die das eigentliche Erstellen der audiogesteuerten Animation erleichtern sollen. Um die Animation rendern zu können, muss zuvor der Befehl *Bake Animation* ausgeführt werden, dessen Einstellungen im dritten Tab zu treffen sind.

In den nachfolgenden Quelltextbeispielen sind sowohl die MEL-Syntax („$ “ vor den Variablennamen, Aufbau von MEL-Befehlen, ...) als auch die Implementierung eines Ausschnitts der Benutzeroberfläche dargestellt.

```
[...]
/** updates timeline -> playbackOptions -minTime & -maxTime
 *
 */
global proc updateTimeline()
{
   string $cmd;
   if(‘objExists Audionode1‘)
   {
      $cmd = "analyzesound -time";
      eval($cmd);
   }
   else{
      warning "no Audionode found";
   }
}

/** creates Audionode and refreshes the UI
 *
 */
global proc createAudionode()
{
   string $cmd = "analyzesound";
   eval($cmd);
   analyzerGUI();
}
```

In diesen beiden MEL-Funktionen, die vom Benutzer über das grafische Interface durch Buttons aufgerufen werden können, ist der im Audioanalyse-Plugin zur Verfügung gestellte „neue“ Maya-Befehl `analyzesound` zu sehen, der im Laufe dieses Kapitels schon beschrieben wurde. Der Quelltextausschnitt zeigt den Aufruf des Befehls mit und ohne Flag, um einerseits die

[6] Anordnung im Karteiformat

Zeitleiste auf die Länge der ausgewählten Audiodatei anzupassen und andererseits den Audionode zu erstellen. In diesem Fall wird der Befehl allerdings von der Funktion `eval()` gekapselt, mit der eventuelle Fehler abgefangen werden können. Alternativ wäre ein Befehlsaufruf ohne diese Kapselung in der Form `analyzesound;` ebenso möglich. Durch die Benutzeroberfläche ist der Benutzer von diesen Details abgeschirmt, sodass die Bedienung des Plugins ohne Kenntnis der Syntax von Maya-Befehlen möglich ist. Das Plugin wäre auch ohne Benutzeroberfläche voll funktionsfähig, allerdings müsste der Benutzer dann die Maya-Befehle kennen und selbst auf die Reihenfolge der ausgeführten Funktionen achten.

```
global proc changeAudiofile(){}
global proc changeDatafile(){}
global proc scaleAudionodeOutputs(int x){}

/** creates the user interface
 *
 */
global proc analyzerGUI()
{

   [...]

      string $audionode = `columnLayout -width 400`;
      frameLayout -collapsable true -label "files" -width 400;
         rowColumnLayout -nc 1;
            button -label "change audiofile"
               -command "changeAudiofile()";
            button -label "change datafile"
               -command "changeDatafile()";
         setParent ..;//rowColumnLayout1
      setParent ..;//frameLayout1

      frameLayout -collapsable true
          -label "modify outputs" -width 400;
         columnLayout -width 250 -co "left" 20;
            text -label "scale outputs" -align "left";
            floatSliderGrp -label "scale leftlevel"
               -min 0 -max 10 -field true
               -fieldMinValue 0 -fieldMaxValue 100
               -changeCommand "scaleAudionodeOutputs(1)"
               -value $valueL scaleleftslider;
            [...]
         setParent ..;//ColumnLayout1
      setParent ..;//frameLayout2
```

```
        frameLayout -collapsable true
           -label "margins for beat detection in Hz" -width 400;
           [...]
        setParent ..;//frameLayout3

        frameLayout -collapsable true
           -label "update Timeline (min/max Playback)" -width 400;
           columnLayout -width 250;
              button -label "update" -command "updateTimeline()";
           setParent ..;//columnLayout3
        setParent ..;//frameLayout4
        [...]

        frameLayout -collapsable true
           -label "connections" -width 400;
           [...]
        setParent ..; //frameLayout5
     setParent ..; //second tab -> columnLayout2 - $audionode

     [...]
}
```

Dieser Quelltextausschnitt zeigt die Gliederung des zweiten Tabs der Benutzeroberfläche und lässt erkennen, dass die verschiedenen Layouts problemlos ineinander verschachtelt werden können. Weiterführende Informationen über das Erstellen von Benutzeroberflächen in Maya und von MEL-Scripts im Allgemeinen sind in [15] nachzulesen.

6.4 Zusammenfassung und Ausblick

Das Implementieren von Maya-Plugins in C++ ist ohne begleitender Literatur teilweise nicht so einfach. Das Erstellen von einfachen Plugins, die lediglich „neue“ Befehle zur Verfügung stellen, geht leicht von der Hand. Bei sogenannten „Dependency Graph“-Plugins, bei denen neue Knoten implementiert werden, sind die Anforderungen ungleich höher. Besonders das Modell der Trennung von Daten und Funktionen in verschiedene Klassen, sowie das korrekte Verbinden der unterschiedlichen Knoten im Dependency Graph ist anfänglich die größte Fehlerquelle. Einen guten Einstieg bei der Entwicklung einfacher Plugins bietet [4]. Leider gibt es keine guten Beispiele für wirklich komplexe Plugins, die auch auf externe Ressourcen (Audiodateien) zugreifen. Die Einbindung der Audiolibrary in das Maya-Gefüge war eine Herausforderung. Debuggen ist in Maya zwar möglich und auch sinnvoll, jedoch programmiert man lange Zeit ohne konkrete Ergebnisse. Leider gab es während der Entwicklungsphase des Plugins auf diesem Gebiet keine vergleichbaren Projekte. Hat man ein funktionierendes Plugin, so sind die Zusammenhänge plötzlich viel besser zu verstehen.

Das Plugin ist sicherlich in einigen Punkten verbesserungsfähig, so wäre beispielsweise die Verwendung einer reinen DSP-Funktion zur Audioanalyse korrekter als die Berechnung der Audioeigenschaften in der `compute`-Funktion. Im Hinblick auf Gesichtsanimationen mittels Spracherkennung wäre eine Formantenerkennung eine sinnvolle Erweiterung des Plugins. Die Umstellung des Plugins auf Audioanalyse mittels DSP-Funktion wird in den nächsten Monaten erfolgen und dann – so wie alle zukünftigen Erweiterungen und Verbesserungen – unter http://work.dabindi.net/audio zur Verfügung stehen.

Anhang A

Glossar

Audionode: Teil des Audioanalyse-Plugins in Maya. Der Audionode ist ein Programmteil, der in jede Mayaszene eingebunden werden kann um dort Analyse-Daten einer beliebigen Audiodatei zur Verfügung zu stellen.

Bark: Die Bark-Skala (nach Heinrich Barkhausen) ist eine psychoakustische Skala für die wahrgenommene Tonhöhe (→ siehe „Tonheit“). Die Skala geht von 0,2 bis 25 Bark. Eine Verdoppelung des Bark-Wertes bedeutet hierbei, dass der entsprechende Ton als doppelt so hoch empfunden wird. Die Bark-Skala ist verknüpft mit der Tonheit in Mel. Es gilt: 1 Bark = 100 Mel. Normiert werden sowohl die Bark- als auch Mel-Skala auf den musikalischen Ton C (131 Hz). Es gilt: 1,31 Bark = 131 Mel = 131 Hz. Ein guter Überblick ist auf http://de.wikipedia.org/wiki/Bark-Skala zu finden.

DFT: Die Diskrete Fourier Transformation ist eine Form der Fourier Transformation für zeitlich begrenzte Signale. Dazu wird das Signal durch ein Fenster der Länge N betrachtet (Multiplikation mit einem Rechteckfenster).

DSP: Digital Signal Processor → Ein digitaler Signalprozessor ist entweder eine spezielle CPU oder ein Algorithmus, welche(r) zur digitalen Signalverarbeitung eingesetzt wird. DSPs ersetzen nicht nur aufwändige analoge Filtertechnik, sondern können darüberhinaus Aufgaben ausführen, die analog nur schwer oder überhaupt nicht lösbar sind:

- Frequenzfilter hoher Ordnung
- Dynamikkompression und Rauschunterdrückung
- Implementierung von Effekten wie Echo, Hall, ...
- Datenkomprimierung zur digitalen Weiterverarbeitung
- ...

Faltung: Eine Faltung im Zeitbereich kommt einer Multiplikation im Frequenzbereich gleich und umgekehrt. Mehr dazu in [9].

Fensterfunktionen: ... werden beim Entwurf digitaler Filter verwendet. Außerdem werden Fensterfunktionen verwendet, um aus Signalen Ausschnitte zu entnehmen. Dazu wird das Signal mit dem jeweiligen Fenster mit der Länge N multipliziert. Es bleibt dann ein zeitdiskretes Signal mit einer Länge von N-Werten ($\rightarrow$ Fensterlänge) über. Das Rechteckfenster verändert das Signal nicht, die anderen Fensterformen (Hann oder Hanning, Hamming, Blackman, Kaiser, Bartlett, ...) gewichten das Signal und sind – abgesehen vom Barlett-Fenster – alle mehr oder weniger glockenförmig. Das Bartlett-Fenster hat die Form eines Dreiecks. Mehr dazu in [9, Kap. 7 und Kap. 11].

FFT: Die Fast Fourier Transformation ist eine schnelle Variante der DFT. Dabei wird die Fenstergröße N so gewählt, dass N als Potenz von 2 darstellbar ist.

Formant: Bei Musikinstrumenten oder der menschlichen Stimme gibt es Frequenzbereiche, bei denen die Lautstärke angehoben ist. Diese Bereiche werden als Formanten bezeichnet und stellen Maxima im Spektrum dar. Die Lage dieser Formanten ist unabhängig von der erzeugten Tonhöhe. Anhand der Lage der Formanten können bei der menschlichen Sprache bestimmte Laute erkannt werden. Durch die Lage der ersten beiden Formanten lassen sich alle Vokale voneinander unterscheiden. Natürlich gibt es Unterschiede der Formantlagen von Mensch zu Mensch, besonders groß ist der Unterschied der Lage zwischen Männern, Frauen und Kindern. Ein Überblick ist auf http://de.wikipedia.org/wiki/Formant zu finden.

Fourier Transformation: ... auch kontinuierliche Fourier Transformation. Ist eine Möglichkeit kontinuierliche, nichtperiodische Signale in ein kontinuierliches Spektrum zu zerlegen – Analysegleichung. Bei der FT wird von einem zeitlich unbegrenzten Signal ausgegangen.

Hanning-Fenster: ... oder auch Hann-Fenster wurde nach seinem Erfinder, dem österreichischen Meteorologen Julius von Hann benannt ($\rightarrow$ Fensterfunktionen).

Harmonien: Harmonien im musikalischen Sinne, sind (ganzzahlige) Vielfache einer Grundfrequenz.

IDFT: Umkehrung der DFT – Synthesegleichung. Berechnung des Signals im Zeitbereich ausgehend vom Frequenzbereich.

IFFT: Umkehrung der FFT – Synthesegleichung. Berechnung des Signals im Zeitbereich ausgehend vom Frequenzbereich.

loudness: ... ist eine subjektive Größe für die empfundene Lautheit eines Audiosignals. Die Lautheit ist nicht zu verwechseln mit der messbaren Lautstärke eines Signals (→ perceptual Features).

Mel, Mel Skala: Mel ist die Maßeinheit für die psychoakustische Größe „Tonheit“ und beschreibt die wahrgenommene Tonhöhe. Die Mel-Skala wurde 1937 von Stanley Smith Stevens, John Volkman und Edwin Newmann vorgeschlagen. Basis für die Definition der Mel-Skala ist der musikalische Ton C. Diesem Ton mit der Frequenz f=131 Hertz wird die Tonheit Z=131 Mel zugeordnet. Ein Ton, der doppelt so hoch wahrgenommen wird, erhält den doppelten Tonheitswert, ein Ton, der als halb so hoch wahrgenommen wird, den halben Tonheitswert. Mit Hilfe psychoakustischer Versuche kann so die Tonheitsskala bestimmt werden. Detailliertere Informationen sind auf http://de.wikipedia.org/wiki/Mel zu finden.

MEL: Maya Embedded Language → Skriptsprache in Maya

perceptual Features: ... sind subjektive Ausdrücke, welche die menschliche Wahrnehmung von Audiosignalen beschreiben. So kann man beispielsweise Eigenschaften wie „Rauhigkeit“, „Klangfarbe“, „Lautheit“, ... als *perceptual Features* bezeichnen.

Rauschen, Weißes oder Rosa: Rauschen entsteht, wenn im Spektrum über einen breiten Bereich viele Frequenzen mit konstanter Leistung auftreten. Zwei besondere Formen des Rauschens werden in der Tontechnik als Referenzsignale verwendet, um bei Lautsprecheranlagen oder Aufnahmegeräten eine möglichst naturgetreue Wiedergabe zu erreichen:

Weißes Rauschen entsteht, wenn über den gesamten Frequenzbereich jedes Teilband die gleiche Rauschleistung enthält. Für den Menschen klingt das Weiße Rauschen scharf, hell und sehr unnatürlich – als ob die hohen Frequenzen größere Energien hätten. Tiefe Frequenzen scheinen zu fehlen.

Beim Rosa Rauschen enthält jede Oktave (Verdoppelung der Tonhöhe) die gleiche Rauschleistung. D. h., dass die Oktave von 20 bis 40 Hz die gleiche Rauschleistung enthält wie die Oktave zwischen 10000 und 20000 Hz. Rosa Rauschen wird aus Weißem Rauschen durch einen Filter mit einem Höhenabfall von 3 dB pro Oktave erzeugt. Für das menschliche Ohr kling Rosa Rauschen natürlicher, als ob über alle Frequenzen eine gleichmäßige Lautstärke vorhanden wäre.

Rechteckfenster: siehe Fensterfunktionen

rhythm: Der Rhythmus ist keine „messbare“ Signaleigenschaft, sondern bildet Muster aus betonten und unbetonten Schlägen. Er wird oft mit

dem *Metrum* verwechselt. Das Metrum gibt an, wie die zugrundlegenden Zeiteinheiten eines Stückes in Gruppen organisiert sind. Diese Gruppen werden als Takte bezeichnet. Das Metrum ist sozusagen das „übergeordnete Zeitmaß“, in welches verschiedene Rhythmen sich einfügen können (vgl. [5, Kap. 7]).

richness: . . . ist eine Größe für die „Fülle“ eines Audiosignals. Ist viel Energie über einen großen Frequenzbereich verteilt, so klingt das Audiosignal voller, ist die Energie auf nur wenige Frequenzen verteilt, klingt das Signal nicht so voll (→ perceptual Features).

pitch: Tonhöhe in Hz

sample: Zeitdiskrete Signale werden mathematisch als Folge von Zahlen dargestellt. In der digitalen Signalverarbeitung wird ein Wert in so einer Zahlenfolge als Sample bezeichnet.

short time energy (function): Sie zeigt die durchschnittliche Energie im Signal pro Zeitausschnitt.

S/N, SNR, Signal Noise Ratio: Das Signal-Rausch-Verhältnis – auch Signal-Rausch-Abstand – ist definiert als das Verhältnis der vorhandenen mittleren Signal-Leistung zur vorhanden mittleren Rauschleistung, wobei der Ursprung der Rauschleistung nicht berücksichtigt wird. Je größer das SNR, desto besser kann das eigentliche Signal vom Rauschen abgegrenzt werden.
Wichtig bei Datentransfer, Audiobearbeitung, . . .

SRA, Signal Rausch Abstand: siehe Signal Noise Ratio

timbre: Klangfarbe - sie ermöglicht dem Hörer zwei oder mehrere Instrumente zu unterscheiden, selbst wenn diese den gleichen Ton spielen.

Tonheit: Die Tonheit ist eine psychoakustische Größe für die empfundene Tonhöhe (→ Bark und Mel).

Verrauschtheit: Die Verrauschtheit eines Signal ist höher, wenn sich die Energie im Signal auf sehr, sehr viele unterschiedliche Frequenzen verteilt. Ist die Energie auf wenige Frequenzen verteilt, so erscheint der Klang klarer.

Video Jockey(VJ): Video Jockeys sind „Musikvideo-Clip-Ansager“, wie sie bei Fernsehsendern, wie beispielsweise MTV und VIVA, täglich zu sehen sind. Die Begriffe Video Jockey und Visual Jockey werden aufgrund der gemeinsamen Abkürzung VJ oftmals verwechselt, wobei die Moderatoren-Tätigkeit der Video Jockeys nicht mit der künstlerischen Performance eines Visual Jockeys nicht zu vergleichen ist. → Visual Jockey

Visual Jockey(VJ): Ein Visual Jockey ist im Allgemeinen für die Visualisierung von Musik mit Hilfe von Bildern, Videoclips und Videoeffekten zuständig. Dabei werden bei Veranstaltungen visuelle Effektshows vorgeführt. In der Regel erweitert ein Visual Jockey dabei eine Audioperformance (z. B. die eines DJs) um eine visuelle Komponente. Aus vielen kurzen Videoclips und Bildern wird live zur Musik ein neues Video kreiert. Idealerweise passt das Video zur Musik oder zum Rhythmus. Meist werden ganz kurze Videoclips synchron zur Musik wiederholt, miteinander gemischt und mit Effekten bearbeitet.

VJ-Tool: Das VJ-Tool ist die Software, welche den Visual Jockey in seiner Arbeit unterstützt. Mit diesem Programm können Videos in verschiedenen Geschwindigkeiten abgespielt, mit Effekten bearbeitet und miteinander gemischt werden.

waveform: Die Waveform stellt das Audiosignal im Zeitverlauf dar. Die Wellenform des Audiosignals.

$\mathbf{x(n)}$ vs. $\mathbf{X(\omega)}$: Für Formeln in diesem Dokument gilt: Signale im Zeitbereich werden durch Kleinbuchstaben dargestellt, Signale im Frequenzbereich werden durch Großbuchstaben dargestellt.

ZCR, Zero Crossing (Rate): Ein Zero Crossing tritt dann auf, wenn aufeinander folgende Samples in einem zeitdiskretem Signal unterschiedliche Vorzeichen aufweisen.

Anhang B

Inhalt des beiliegenden Datenträgers

B.1 Sounds

Dateien im .wav-Format, 44.1 kHz, mono, Dauer jeweils $\tilde{5}$s
Dateiem im .mp3-Format, 44.1 kHz, stereo

Pfad: /sounds

audio1.wav	Sprachdatei
audio2.wav	Ludwig van Beethoven - Violin Concerto in D major, Op. 61
audio3.wav	Public Domain - Operation Blade

B.2 Maya-Plugin

Pfad: /MayaPlugin

maya65.rar	Plugin für Maya 6.5, Windows XP
maya7.rar	Plugin für Maya 7.0, Windows XP
/doku	Plugin .html Dokumentation
/analyzer	Plugin Projektdateien

B.3 Diplomarbeit

Pfad: /print

StefanBindreiter.dvi . .	Diplomarbeit (DVI-File, ohne Grafiken)
StefanBindreiter.pdf . .	Diplomarbeit (PDF-File)
StefanBindreiter.ps . . .	Diplomarbeit (PostScript-File)

Pfad: /latex-source

StefanBindreiter.tex	Hauptdatei
...	weitere *LaTeX*-Dateien
literatur.bib	Quellenverzeichnis
/images	Bilder im EPS-Format

B.4 Online-Quellen

Pfad: /documents

AuditoryToolbox.pdf	Slaney, Malcolm
bass.chm	BASS Audiolibrary Dokumentation
BeatDetection.pdf	Patin, F.
classification.pdf	Klapuri, Anssi
dmx_r3.pdf	Entwurf DMX512-A Protokoll
forum_highend3d.html	Forumausschnitt Highend3D
logan_paper.pdf	Logan, B.
merkmale_fuer_.pdf	Niemann, H.

Anhang C

Benutzerdokumentation des Maya-Plugins

C.1 Installation

Kopieren der Plugin-Dateien Die beiden Dateien analyzer.mll und bass.dll müssen gemeinsam in einem Ordner liegen und können in das Plugin-Verzeichnis (../maya_home/bin/plugins) kopiert werden.

Benutzeroberfläche Um das Plugin über die Benutzeroberfläche in Maya steuern zu können, muss die Datei analyzerGUI.mel in das Skript-Verzeichnis von Maya kopiert werden (c:/documents and settings/user/maya/6.5/scripts). Wird Maya 7 verwendet, so muss natürlich anstelle des Ordners „6.5“ der Unterordner „7.0“ ausgewählt werden.
Nach Durchführung der beiden Installationsschritte kann das Plugin bereits verwendet werden. Allerdings muss Maya neu gestartet werden.

C.2 Bedienung

Start Ist Maya gestartet, so kann durch das Eintippen des Befehls `analyzerGUI` entweder in die Befehlszeile oder in den Skript-Editor von Maya die Benutzeroberfläche aufgerufen werden. Natürlich kann für diesen Befehl ein eigener Button in Maya erstellt werden, was in der Maya-Hilfe gut beschrieben ist. Durch Drücken der Eingabetaste erscheint dann die Benutzeroberfläche. Ist dies nicht der Fall, so ist zu überprüfen, ob die Datei analyzerGUI.mel im oben angeführten Verzeichnis liegt und ob Maya auch wirklich nach der Installation neu gestartet wurde.

Laden/Entladen des Plugins Der erste Schritt – egal ob eine neue Szene oder eine bereits bestehende mit Audionode bearbeitet wird – ist immer das Laden des Plugins. Über den Button „Plugin Manager“ im ersten Tab der

Benutzeroberfläche wird der Plugin Manager aufgerufen. Hier kann muss die Datei analyzer.mll ausgewählt werden.

Erstellen des Audionodes Ist das Plugin erfolgreich geladen, so kann mit dem Button „create Audionode“ ein neuer Audionode erzeugt werden. Es darf allerdings nur *ein* Knoten in der Szene enthalten sein. Ist bereits ein Audionode vorhanden, oder das Plugin nicht geladen kommt es zu einer Fehlermeldung. Ist ein neuer Knoten erzeugt worden, so öffnet sich sofort ein Dialog-Fenster, in dem die zu analysierende Audiodatei ausgewählt werden kann. Neben .mp3-Dateien können auch .wav-Dateien mit einer Abtastrate von 44.1 kHz geöffnet werden. Die Audiodatei kann natürlich jederzeit geändert werden.

Bearbeiten der Szene Die Elemente zum eigentlichen Bearbeiten der Szene mit dem *Audionode* befinden sich im gleichnamigen Tab. Mit den Buttons „change audiofile“ und „change datafile“ können die zu analysierende Audiodatei sowie die optionale Datendatei mit den Analysedaten ausgewählt werden. Mit den verschiedenen Schiebereglern können die Analysedaten skaliert werden. Bei der Beaterkennung wird der Algorithmus durch niedrigere Werte sensibler, bei höheren Werten werden nur ganz intensive Beats erkannt. Mit dem Button „select Audionode“ wird der Knoten ausgewählt, damit man beispielsweise im Attribut-Editor auf die einzelnen Parameter direkt zugreifen kann. Der Button “update“ richtet die Zeitleiste nach der Länge der ausgewählten Audiodatei aus. Die anderen Buttons öffnen die diversen Verbindungseditoren, in denen der *Audionode* mit anderen Elementen in der Szene verbunden werden kann. Sind Verbindungen hergestellt, so sind die Auswirkungen durch Abspielen der Szene oder Hin- und Herklicken in der Zeitleiste zu sehen.
Um die Analysedaten in eine externe Datei zu schreiben, muss der „Betriebsmodus“ des Audionodes im Attribut-Editor geändert werden.

Wichtig! Bevor eine Datei geöffnet wird, die bereits einen Audionode enthält, *muss* das Plugin zuvor geladen werden. Ebenso *muss* vor Erzeugen des Audionodes das Plugin geladen werden.

C.3 Hinweise

Um eine audiogesteuerte Animation rendern zu können, *müssen* zuvor jene Teile, die mit dem Audionode verbunden sind, „gebaked“ werden. Maya kann leider während des Renderns nicht auf die Audiodaten zugreifen. Aus diesem Grund wird die Animation vorberechnet. Über den Button „Bake Animation“ der Benutzeroberfläche gelangt man zu den Einstellungen für das „Baken“. Es müssen nicht alle Objekte vorberechnet werden, sondern

es reicht aus jene Objekte zu markieren und zu „baken“, welche Daten vom Audionode empfangen.

Es ist nicht zwingend notwendig, dass sich die beiden Plugin-Dateien (analyzer.mll und bass.dll) im Plugin-Verzeichnis von Maya befinden. Solange die beiden Dateien im *selben* Verzeichnis liegen, ist es egal, wo sie sich befinden. Für ein rasches Laden des Plugins ist es allerdings von Vorteil, die Dateien in den Plugin-Ordner zu kopieren.

Es kann sein, dass sich das Plugin im Plugin Manager *nicht* durch einfaches Anklicken der Checkbox laden lässt. In diesem Fall muss über den „Browse“-Button die Datei analyzer.mll nochmals ausgewählt werden.

Eine technische Dokumentation ist sowohl auf dem beiliegenden Datenträger als auch online unter http://work.dabindi.net/audio zu finden. Die Dokumentation zur Audiolibrary ist ebenfalls am Datenträger enthalten sowie unter http://www.un4seen.com auch online vorhanden.

Literaturverzeichnis

[1] BV, T.: *DMX512*. URL, http://www.theater-technisch-lab.nl/dmxdu.htm, April 2000. letztes Mal online besucht: 10.06.2006.

[2] Doblinger, G.: *Signalprozessoren*. J. Schlembach Fachverlag, Wilburgstetten, Deutschland, 2004.

[3] ESTA, U.: *USITT DMX512-A*. URL, http://www.soundlight.de/techtips/dmx512/dmx_r3.pdf, Oktober 2000. Kopie auf CD-Rom (documents/dmx_r3.pdf).

[4] Gould, D. A. D.: *Complete Maya Programming*. Morgan-Kaufmann, San Francisco, CA, 2003.

[5] Hall, D. E.: *Musikalische Akustik - Ein Handbuch*. Schott Musik International, Deutschland, 1997.

[6] Klapuri, A.: *Audio signal classification*. URL, http://www.iua.upf.es/mtg/ismir2004/graduateschool/, Oktober 2004. Kopie auf CD-ROM (documents/classification.pdf).

[7] Kosina, K.: *Music Genre Recognition*. Diplomarbeit, Fachhochschule Hagenberg, Medientechnik und -design, Hagenberg, Austria, Juni 2002.

[8] Logan, B.: *Technical Report on Auditory Toolbox*. URL, http://ismir2000.ismir.net/papers/logan_paper.pdf, Oktober 2000. Kopie auf CD-Rom (documents/logan_paper.pdf).

[9] Oppenheim, A. V. und R. W. Schafer: *Zeitdiskrete Signalverarbeitung*. Oldenbourg Verlag, 1999.

[10] Patin, F.: *Beat Detection Algorithms*. URL, http://www.yov408.com/html/articles.php?page=1, Juni 2003. Kopie auf CD-Rom (documents/BeatDetectionAlgorithms.pdf).

[11] Rabiner, L. und R. Schafer: *Digital Processing of Speech Signals*. Prentice-Hall International, Inc., 1978.

[12] Raffaseder, H.: *Audiodesign*. Fachbuchverlag Leipzig im Carl-Hanser-Verlag, Leipzig, Deutschland, 2002.

[13] Slaney, M.: *Technical Report on Auditory Toolbox*. URL, http://www.speech.cs.cmu.edu/comp.speech/Section1/HumanAudio/auditory.t%lbx.html, Oktober 1998. Kopie auf CD-Rom (documents/Auditory Toolbox TechReport.pdf).

[14] Uppuluri, Jyoti und Verret, R.: *Beat this!*. URL, http://www.owlnet.rice.edu/~elec301/Projects01/beat_sync/beatalgo.html%, Dezember 2001. letztes Mal online besucht: 11.06.2006.

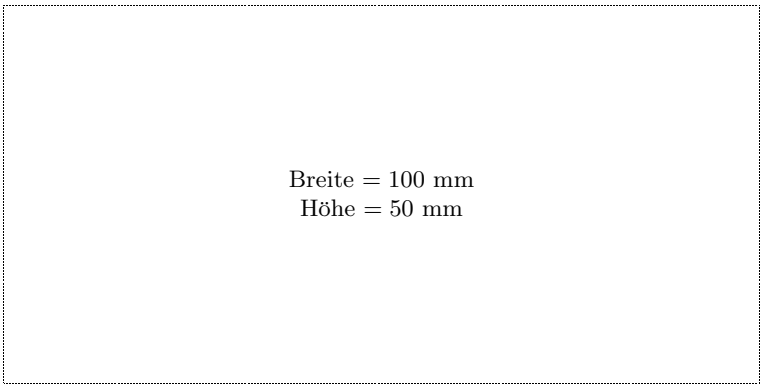
[15] Wilkins, M. R. und C. Kazmier: *MEL-Scripting for Maya Animators*. Morgan-Kaufmann, San Francisco, CA, 2003.

[16] Zhang, T. und C.-C. J. Kuo: *Content-based Audio Classificatoin and Retrieval for Audiovisual Data Parsing*. Kluwer Academic Publishers, Norwell, Massachusetts, 2001.

[17] Zölzer, U.: *DAFX - Digital Audio Effects*. John Wiley Sons, Ltd, England, 2002.

Messbox zur Druckkontrolle

— Druckgröße kontrollieren! —



— Diese Seite nach dem Druck entfernen! —