


# 4. Audiotechnik und Tonbearbeitung

- 4.1 Grundlagen der Audiotechnik
- 4.2 Analoge Audiotechnik
- 4.3 Raumklang
- 4.4 Digitale Audiotechnik 
- 4.5 Programmierung für Audioverarbeitung

## Literatur:

- Th. Görne: Tontechnik, 2. Auflage, Hanser-Verlag 2008
- H. Raffaseder: Audiodesign, Fachbuchverlag Leipzig 2002
- B. Katz: Mastering Audio – The Art and The Science,  
Focal Press 2002

# Geschichte der digitalen Audiotechnik

1926: Erstes Patent für PCM-Abtastung  
(P.M. Rainey, Western Electric)

1962: PCM-Übertragungssystem  
(Bell Telephone)

1974, Sony: PCM-Audio digital auf Magnetband

ab 1975: Digitale Masterbänder in Tonstudios

1979, Philips: Erste digitale Audiodisc

1981, Philips/Sony: Compact Disc (CD)

1988 Verkaufszahlen von CDs über denen von LPs

1981: Standardisierung von MIDI (Synthesizer)

1990: DAT-Recorder (Digital Audio Tape)

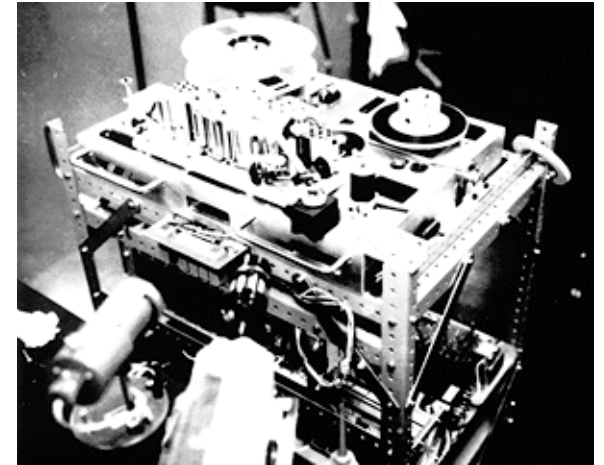
1990: CD-R

1992: Philips DCC und Sony MiniDisc (MD)

1999, Sony/Philips: Super Audio CD (SACD) und Direct  
Stream Digital

1999: DVD-Audio

2000+: Direct-to-disc Recording etabliert sich



Sony digital tape recorder 1974



DAT Tape (1990–2005)

# Analog-/Digital-Wandlung

## Abtastung

Diskretisierung (Abtastrate)

Quantisierung (Wortlänge)

## Abtastraten:

32 kHz (Broadcast, Semi Professional)

44.1 kHz für CDs, Mastering und Aufzeichnung mit Videogeräten  
der 625/50 Norm

44.056 kHz für Aufzeichnung mit Videogeräten der 525/59.94 Norm

48 kHz professionelle Studionorm für Recorder mit feststehenden Köpfen

96 kHz High End Mastering, DVD Audio

2.8224 MHz „Direct Stream Digital“ (z.B. für SACD = Super-Audio CD)

- Wortlängen:

16 Bit (Audio-CD)

24 Bit (Mastering)

Interne Wortlängen von Bussystemen und Effektgeräten oft höher

# Digitale Pegel

Maximaler digitaler Audiopegel: 0 dBFS (dB Full Scale)

*Maximal* möglicher Wert in der gewählten Quantisierung

Prinzipiell unabhängig vom Spannungswert nach Analog-Konversion

Alle real auftretenden Pegelwerte sind negativ

1 Bit Wortlänge = Verdopplung des Dynamik-Wertebereichs = 6 dB

8 Bit Wortlänge = 48 dB (z.B. 00000000 = -48 dBFS)

16 Bit Wortlänge = 96 dB

24 Bit Wortlänge = 144 dB

Digitaler Headroom: Differenz zwischen Arbeitspegel und 0 dBFS

Mindestens -9 dBFS, oft höherer Abstand vorgeschlagen (z.B. -20 dBFS)

„Intersample Peaks“, optimaler Arbeitsbereich für analoge Verstärker

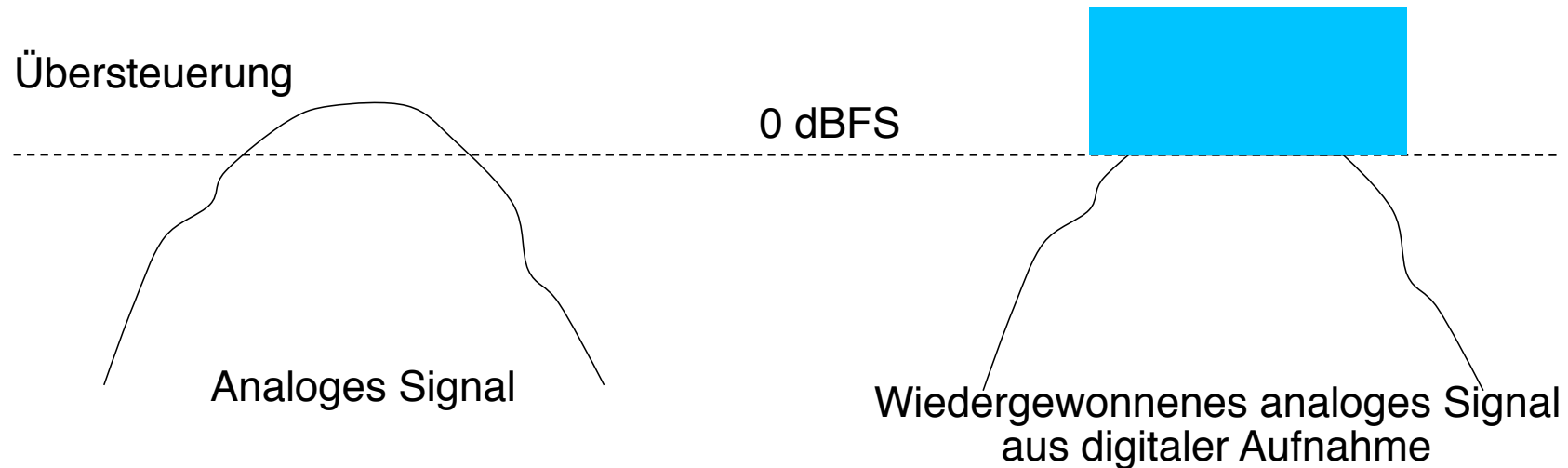
D/A-Wandlerpegel 6 dBu entspricht digitalem Arbeitspegel

Rauschen in der digitalen Bearbeitung:

A/D-Wandler: Rauschen bei ca. -104 dBFS

Digitale Mixer: Rauschen bei ca. -120 dBFS

# Digitales „OVER“?



Signalanteile, die durch Übersteuerung bei der Aufnahme verloren gehen, sind für immer verloren („Clipping“)

„OVER“-Anzeige nur bei Aufnahme, nicht bei Wiedergabe!

## Digitales Clipping

Akustisch wesentlich unangenehmer als bei analoger Übersteuerung

Rechteck-artige Schwingungen

## OVER-Anzeige bei digitalen Peakmetern:

Folge von mehreren aufeinander folgenden 0 dBFS-Werten (z.B. 3)

# Verzerrungen bei der Digitalisierung

## Quantisierungsfehler:

Differenz zwischen "echter" Amplitude und nächstgelegener Quantisierungsstufe

Kann als additives Rauschen aufgefasst werden

*Quantisierungsrauschen*

## Granularrauschen:

Z.B. einzelner Ton in extrem stiller Umgebung

Kleine Signale werden als Rechtecksignal wiedergegeben

Hochfrequente Obertöne hörbar als Klirrgeräusche

Schwankende Amplitude: "Flanger"-Effekt

Abhilfe: Dithering (siehe später)

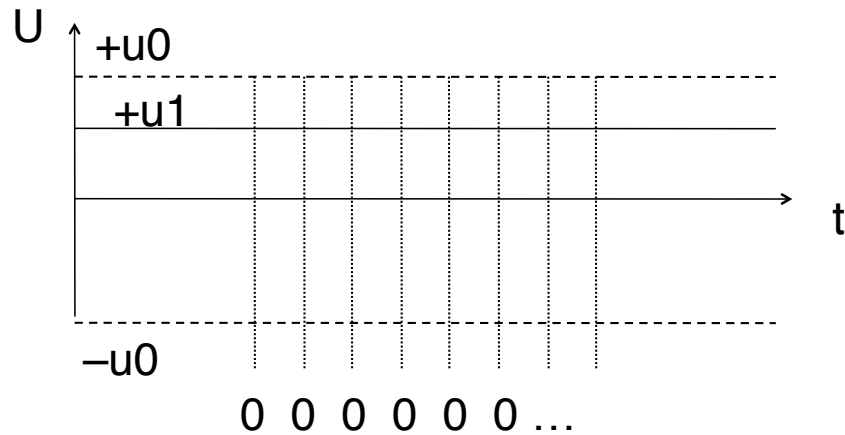
## Rundungsfehler:

Rechnen mit Audio-Daten, z.B. bei Divisionen

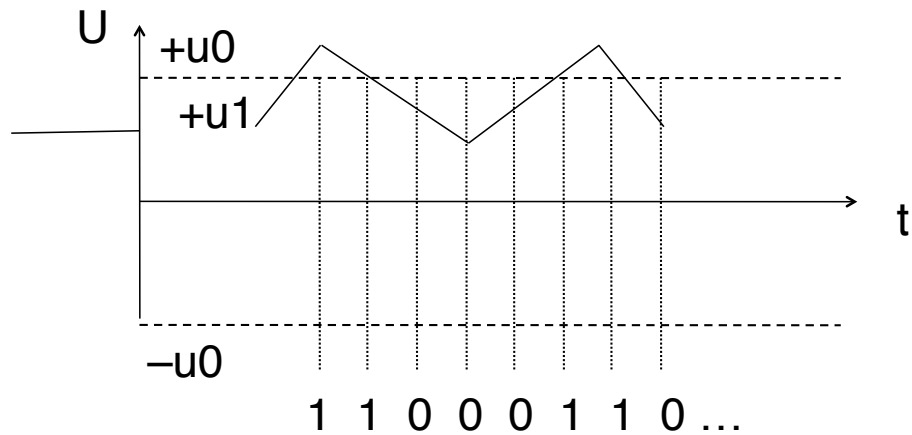
Abhilfe: Wortlängenmanagement (siehe später)

Audio example for Flanging  
(Source Wikipedia)

# Audio-Dithering



Annahme: Quantisierungsschwelle bei  $u_0$   
(LSB, Least Significant Bit)  
Konstante Spannung  $u_1$  unterhalb  $u_0$  (z.B.  $u_0/2$ )  
Sampling liefert Null-Werte



Weißes Rauschen (Zufallsignal) beigemischt  
mit niedriger Amplitude ( $u_0/2$ )  
Sampling liefert zufällige Wertereihe 0 und 1  
*Durchschnitt (bei nachfolgender Glättung)  
entspricht  $u_1$  !*

Hinzufügen von Rauschen steigert Auflösung der Digitalisierung  
Geringere Quantisierungsverzerrungen für extrem niedrige Pegel

# Oversampling bei der A/D-Wandlung

## *Oversampling:*

Vervielfachung der Abtastrate (im Vergleich zu der nach dem Nyquist-Theorem eigentlich ausreichenden Rate)

Meist um eine Zweierpotenz (2-fach, 4-fach, 8-fach)

Sowohl bei der Digitalisierung (A/D-Wandlung) einsetzbar als auch bei der Reproduktion (D/A-Wandlung)

## Oversampling bei der A/D-Wandlung:

Beispiel: CD-Audio (Bandbreite bis 22,05 kHz)

Mindestabtastrate nach Nyquist 44,1 kHz

4-fach Oversampling: Abtastung mit 176,4 kHz

Tiefpassfilterung (Anti-Aliasing Filter)

(analog: vor Abtastung auf 88,2 kHz oder

digital: nach Abtastung auf 22,05 kHz)

Dezimation: Reduktion der Abtastrate (Weglassen von Samples)

## Vorteile:

Analoge Tiefpassfilter mit weniger steilen Flanken ausreichend

Digitales Signal basiert auf Analyse von mehr Information

Quantisierungsrauschen wird auf größeren Frequenzbereich verteilt



# Noise Shaping

## Grundidee:

Quantisierungsrauschen bei großen Frequenzen erhöhen und bei niedrigen Frequenzen verringern

Idealerweise: Anhebung außerhalb des Hörbereichs und Absenkung im Hörbereich

Analogie zu Floyd-Steinberg-Dithering bei Bildern

## Noise Shaping ohne Oversampling:

Kann Quantisierungsrauschen in schlechter hörbare Bereiche verschieben

## Noise Shaping mit Oversampling:

Kann Quantisierungsrauschen in Bereich zwischen höchster Signalfrequenz und Abtastgrenze verschieben, also unhörbar machen

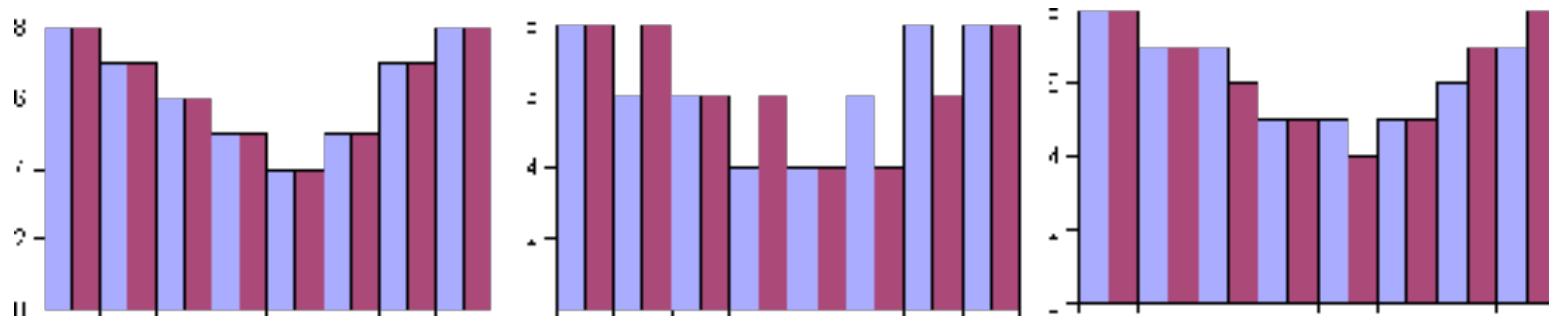
## Realisierung:

Rückkopplung des Quantisierungsfehlers zum Quantisierer-Eingang  
(auf ein oder mehrere Samples bezogen: "Ordnung" des Shapers)

Interne Verwendung höherer Auflösung und Wortbreitenreduktion

Setzt Grundrauschen voraus (kombiniert mit Dithering)

# Oversampling bei der D/A-Wandlung



3-Bit-Wandler, 2 Bit-Wandler und  
2-Bit Wandler mit gleitenden Mittelwerten

[wwwex.physik.uni-ulm.de](http://wwwex.physik.uni-ulm.de)

## Künstliche Erhöhung der Abtastrate:

Erhöhung der Abtastrate auf ein Vielfaches und Belegung der neuen Samples mit Nullwerten

Digitale Tiefpassfilterung zur Bandbegrenzung und Interpolation

Ermöglicht einen Teilschritt der D/A-Wandlung voll digital durchzuführen

"Oversampling DAC":

Zuerst Oversampling und Noise Shaping (durch Feedback),  
dann D/A-Wandlung mit niedriger Bitauflösung (z.B. 1-Bit-DAC)

# Wortlängenmanagement

Digitale Bearbeitung liefert größere Wortlängen

Berechnungsergebnisse in der Signalverarbeitung

Es ist sinnvoll, mit längeren Wortlängen zu arbeiten

Intern in digitalen Audiogeräten

In der digitalen Audio-Bearbeitungskette

Vermeiden der Akkumulation von Rundungsfehlern!

Rückführung auf Medienwortlänge (Rundung)

In einem einzigen Schritt, nicht mehrfach

Am Ende der Bearbeitungskette

Meist unter Verwendung von Dithering (Re-Dithering)

# Digitale Audiotechnik

## Klassifikationen digitaler Audiobausteine:

### Hardware oder Software

Hardware notwendig für A/D- und D/A-Konversion

Hardware sinnvoll bei hohen Anforderungen an Latenzzeit

Hardware-Bedienelemente für Software-Lösungen  
(z.B. digitales Mischpult)

### Produktion oder Modifikation

Produktion: Synthesizer, Tongeneratoren, MIDI-Eingabegeräte

Modifikation: Mischung, Filter, Effekte

### Live oder Postprocessing

Live: Aufnahmestudio, Live-Bühne, Rundfunk

Postprocessing: „offline“ Nachbearbeitung

Hauptunterschied: Latenzzeiten z.B. für Effekt-Rendering

# Soundkarten



## Bestandteile:

- Digitaler Signal-Prozessor (DSP)

- Digital-Analog-Converter (DAC)

- Analog-Digital-Converter (ADC)

- Speicher (z.B. für Wellenformen bei Klangsynthese und als Puffer)

## Anschlüsse:

- Analoge und digitale Audio-Ein-/Ausgänge

- MIDI-Ein-/Ausgänge

- Raumklang- (Surround Sound-)Ausgänge

Einfache Soundfunktionalität heutzutage meistens "on board"

## Geschichte:

- Xerox *Alto* Prototyp beherrscht einfache Audio-Verarbeitung (1972)

- Erster Audio-fähiger Kleincomputer für den Massenmarkt: Apple Macintosh (1982)

# Digitales Mischpult



Digitales Mischpult  
mit eigenem DSP (Digital Sound Processor)



Steuergerät für Software  
(z.B. Cubase, Logic)  
angeschlossen über FireWire oder USB



Reine Softwarelösung,  
Steuert u.U. Hardware (z.B. spezielle Soundkarten)

# Software: Sequencer

Aufnahme und Erstellung (Komposition) von MIDI-Daten

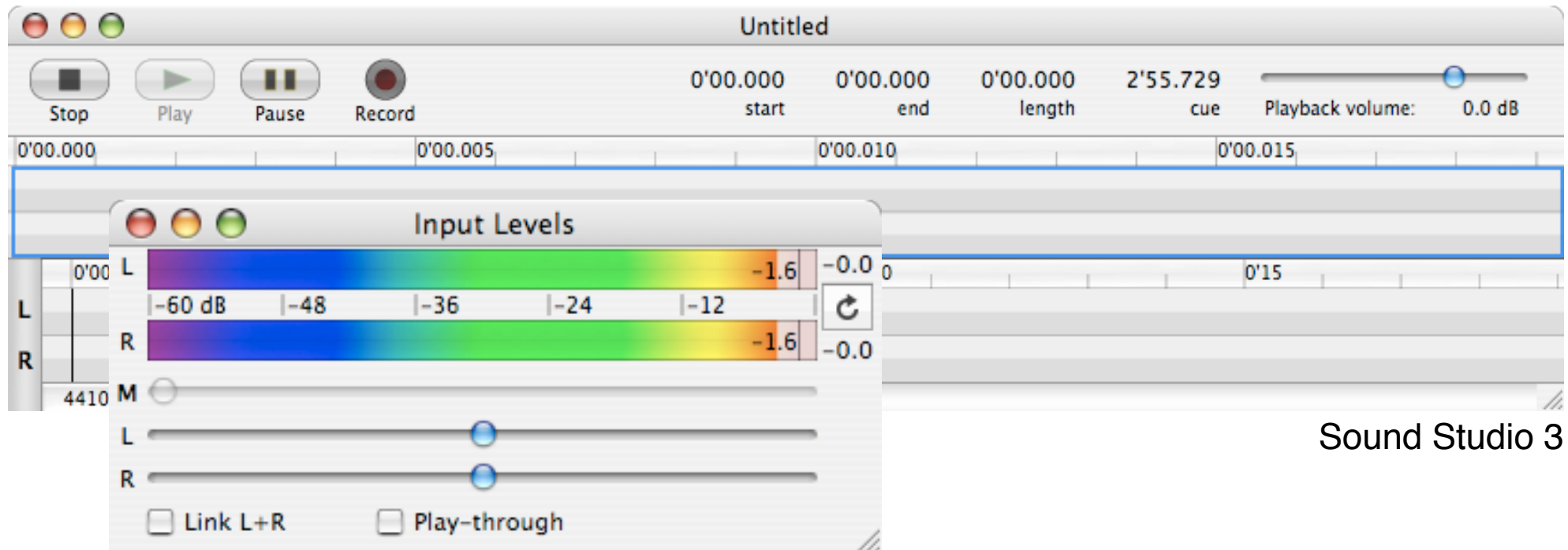
Mischen von  
MIDI-Spuren  
und  
digitalisierten  
Audio-Daten  
(virtuelles  
Mischpult)

Digitale Effekte



Intuem

# Software: Audio-Aufnahme

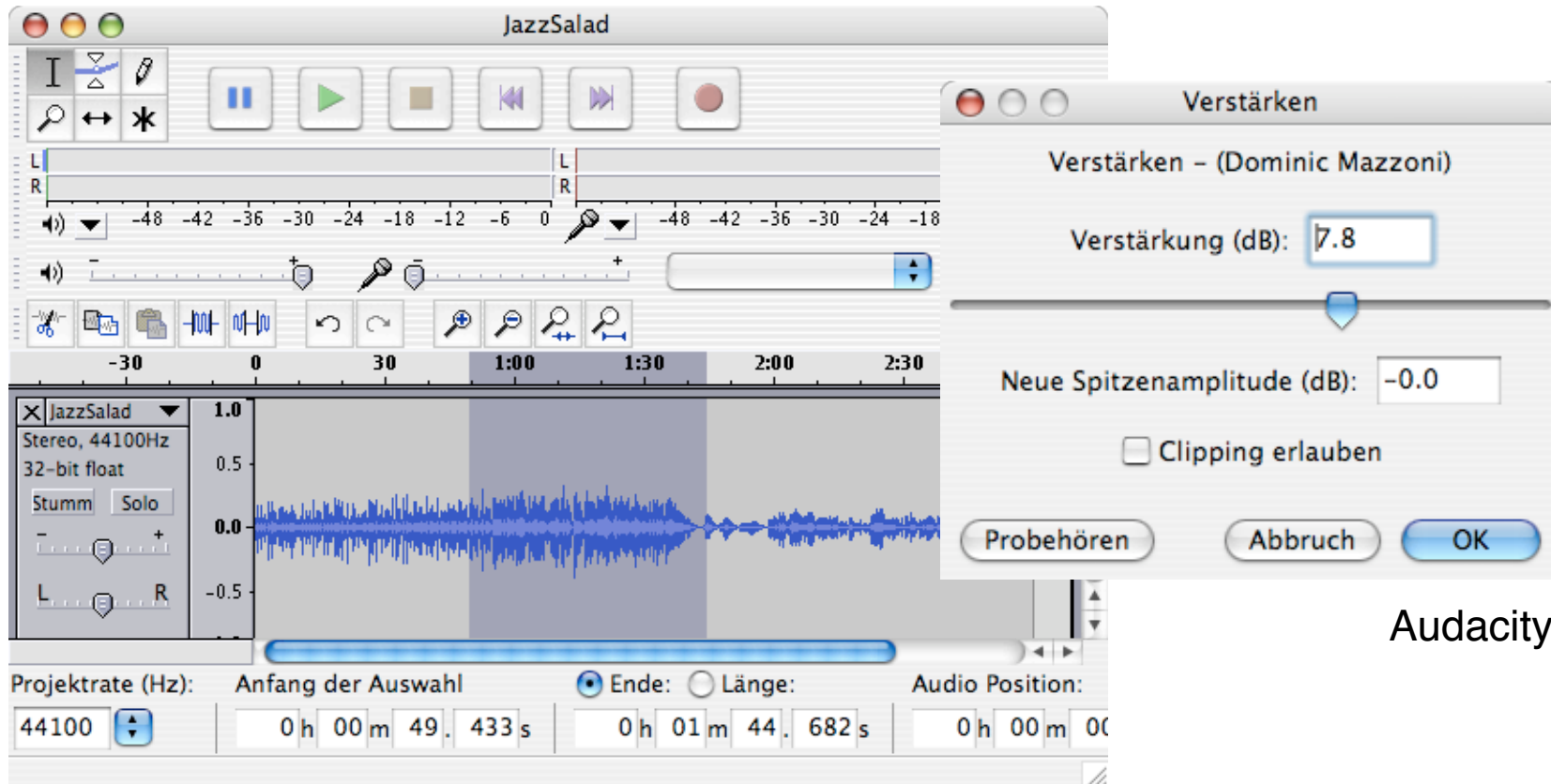


Sound Studio 3

- Zugriff auf Eingabegeräte (Mikrofone, Mischpult)
- Fortschrittsanzeige
- Aussteuerungsanzeige und -Kontrolle
- Speicherung in diversen Formaten
- Oft kombiniert mit anderen Funktionen (z.B. Postprocessing)



# Software: Audio-Bearbeitung



Audacity

Import diverser Audio-Formate

Misch- und Schnittfunktionen

Nachbearbeitung von Klangspektrum und Pegel

Anwendung von Audio-Effekten

# Bearbeitung der Amplitude (1)

Verstärken, Dämpfen, "Normalizing":

nachträgliche Anpassung des gesamten Signalpegels

Z.B. Einstellen auf Standard-Durchschnittspegel („Normalizing“)

bei Verstärkung werden auch unerwünschte Effekte  
(z.B. Rauschen, Nebengeräusche) mitverstärkt!

Normalizing:

Relativ zu welchem Standard?

Bsp.: Abmischen eines Albums:

Auf keinen Fall einzelne Titel auf Standard-Pegel „normalisieren“

Sinnvoller: Normalizing des Gesamtalbums  
(Charakter der Einzelsongs bleibt erhalten)

# Bearbeitung der Amplitude (2)

Bearbeiten der Hüllkurve:

Ein- und Ausblenden (*fading*)

Typen von Fadern:

linear

logarithmisch

exponentiell

diverse Zwischenformen

Kompensation von "DC-Offset"

Signal genau auf die gewünschte Null-Linie einstellen

Unerwünschten "Gleichspannungsanteil" abziehen

Vorsicht beim Zusammenmischen verschiedener Signale:

Gesamtpegel beachten! Signale werden meistens additiv  
zusammengeführt

Digitale Übersteuerung führt zu *Clipping*, welches nachträglich nicht  
mehr korrigiert werden kann

Abhilfe: Einzelne Signale/Spuren vorher entsprechend abschwächen

# Dynamik

Unterschied zwischen lauten und leisen Passagen

Höchster minus kleinster Amplitudenwert in einem Zeitfenster

Mikrodynamik

Kleine Zeitfenster (Sekundenbruchteile)

Eigenschaften von Instrumenten

Z.B. Schlagzeug hat hohe Mikrodynamik

Bearbeitung der Dynamik eines Einzelkanals vor der Abmischung

Makrodynamik

Große Zeitfenster (Sekunden bis Minuten)

Eigenschaften von Songs bzw. Szenen

Z.B. pianissimo bis fortissimo

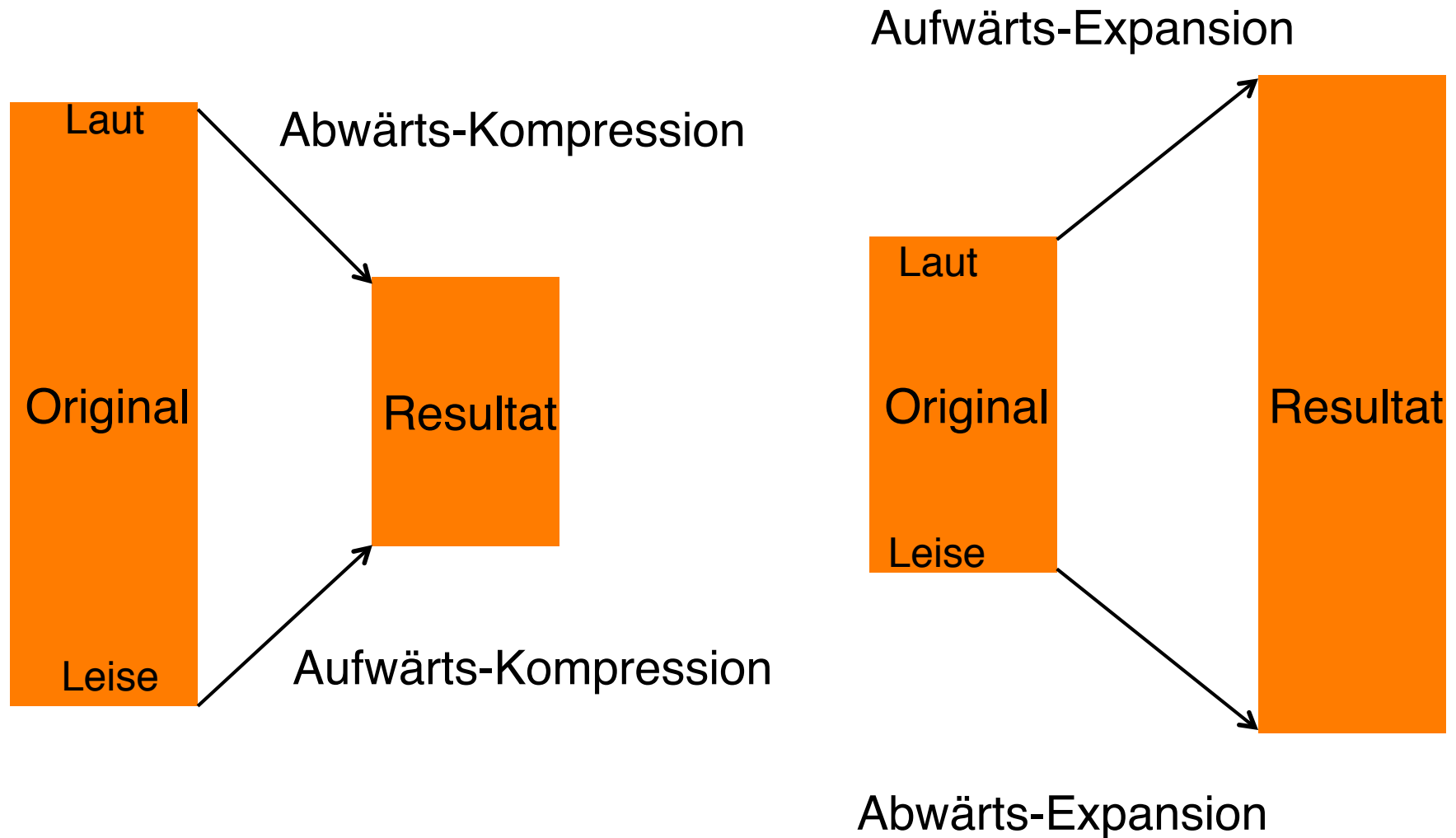
Bearbeitung der Gesamtdynamik eines längeren Abschnitts

Probleme mit Dynamik:

Übersteuerung bei Wiedergabe

Unhörbarkeit durch Hintergrundgeräusche

# Varianten der Dynamik-Modifikation



# Bearbeitung der Dynamik (1)

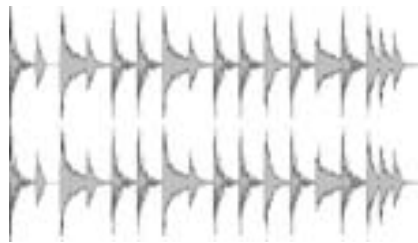
Kompressor:

Schwelle (*threshold*): Amplitudenwert, ab dem die Reduktion einsetzt

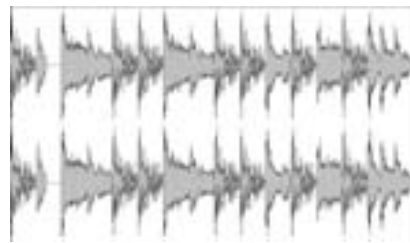
Faktor (*ratio*): Stärke der Reduktion

Einsatz (*attack*) und Ausklingen (*release*): Übergangszeiten

Beispiel:



unkomprimiert



komprimiert  
Threshold -10dB  
Ratio 10.0  
Attack 10 ms  
Release 100 ms

# Bearbeitung der Dynamik (2)

## Multiband-Kompressor

Aufteilung des Signals in Frequenzbänder

Jedes Frequenzband einzeln komprimiert

Geeignet zur selektiven Veränderung des Klangbilds

## Expander:

Umkehrfunktion zum Kompressor (Dynamikerhöhung)

## Limiter:

Schneidet Signal oberhalb der Schwelle auf festen Pegel ab

Vermeidet Übersteuerung

## Gate:

Schneidet Signal unterhalb des Schwellwertes ab

Kann zur Rauschunterdrückung oder zum Ausschalten von Nebengeräuschen dienen

# Bearbeitung des zeitlichen Verlaufs (1)

## Schneiden von Audio-Material

Früher wirklich mit der Schere (am Tonband)!

Heute mit Cut, Copy und Paste im Audio-Editor

Möglichst in Pausen schneiden

Immer im Nulldurchgang schneiden (sonst Knacken hörbar)

Schnitte zu ähnlichem Signal sind relativ problemlos

Schnitte von leisem zu lautem Signal sind relativ problemlos

Schwierig sind Schnitte innerhalb eines kontinuierlichen Signals

## Überblenden:

Bearbeiten der Hüllkurven und Zusammenmischen

Kreuzblende (langsam) und Sturzblende (rasch)

## Schleifen (*loop*):

Verlängern eines akustischen Ereignisses durch Wiederholen

Schnittregeln an der Nahtstelle zu beachten



# Bearbeitung des zeitlichen Verlaufs (2): Resampling

Andere Wiedergabegeschwindigkeit als Samplingrate

Ändert die Zeitdauer und gleichzeitig alle Tonhöhen

Beispiel:

Original

Resampling \* 0,7

Resampling \* 1,4

Beispiel: Erzeugung neuer Töne

Kugel auf Teller

1/16 Resampling

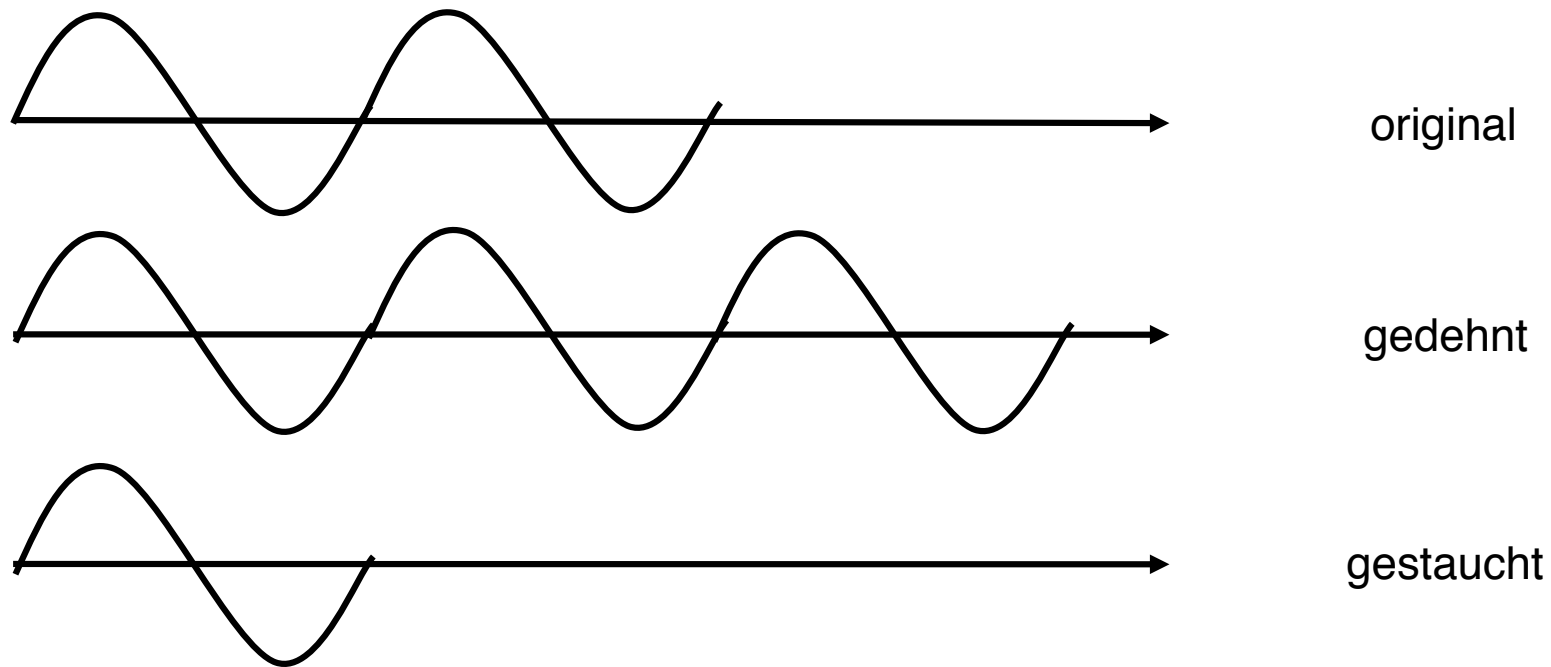
1/64 Resampling

# Bearbeitung des zeitlichen Verlaufs (3): Timestretching

Zeitkorrektur ohne hörbare Tonhöhenveränderung

nur in begrenztem Umfang möglich

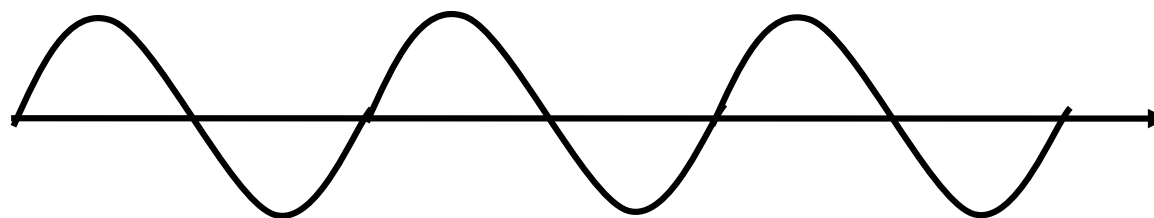
Idee: Wiederholung kleiner periodischer Abschnitte



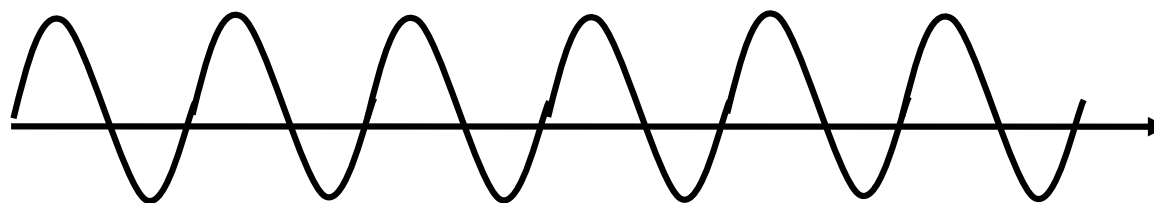
# Bearbeitung des zeitlichen Verlaufs (4): Pitchshifting

Tonhöhenkorrektur bei gleicher Spieldauer

Zusammensetzbar aus Timestretching und Resampling



Original



höher

tiefer

# Raumorientierte Bearbeitung: Echo und Hall

Echo:

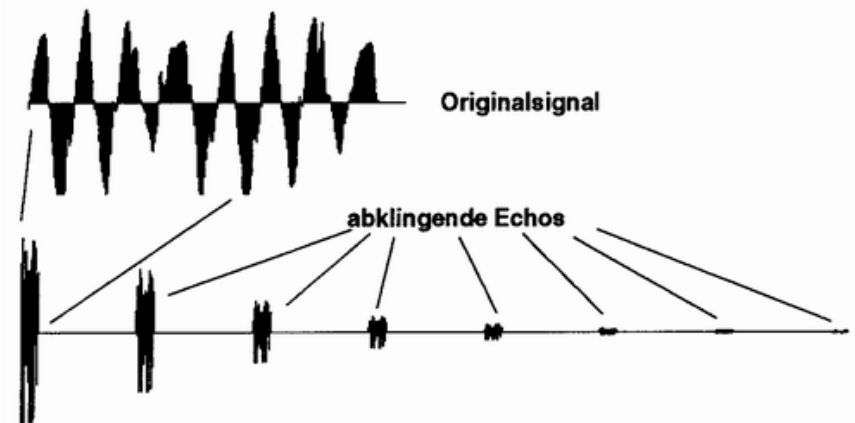
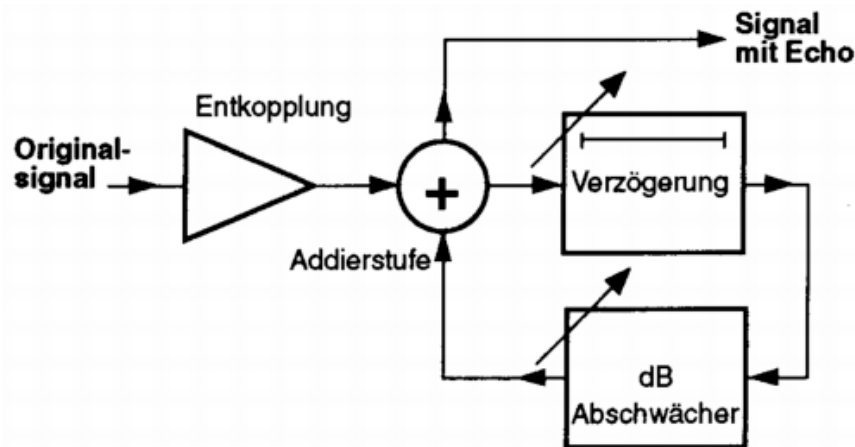
Signal einmal verzögert und abgeschwächt zurückgeführt

Parameter: Verzögerung, Abschwächung

Hall:

Signal vielfach (unendlich oft) mit verschiedenen Verzögerungszeiten zurückgeführt

Halleinstellungen können sehr komplex sein



Echoprinzip

# Schallausbreitung im geschlossenen Raum

## Direktschall:

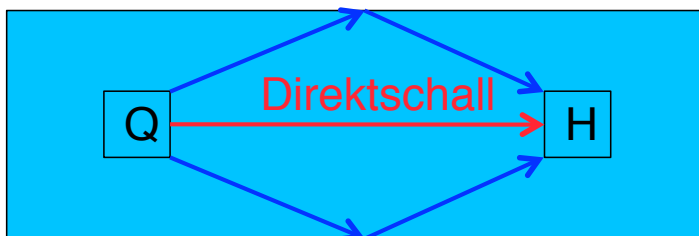
gerade Linie zwischen Quelle und Hörer  
kürzester Weg

## Erstreflexionen:

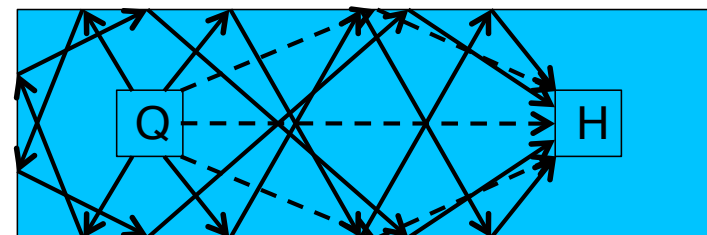
längerer Weg, längere Laufzeit  
als unterschiedliches Signal wahrnehmbar (Echos)

## Mehrfachreflexionen:

als einheitliches "Hall-" Signal wahrgenommen  
klingt mit der Zeit ab



Erstreflexionen



Mehrfachreflexionen

# Impulsantwort

Verhältnis von Direktschall, Erstreflexionen und Nachhall kann experimentell ermittelt werden:

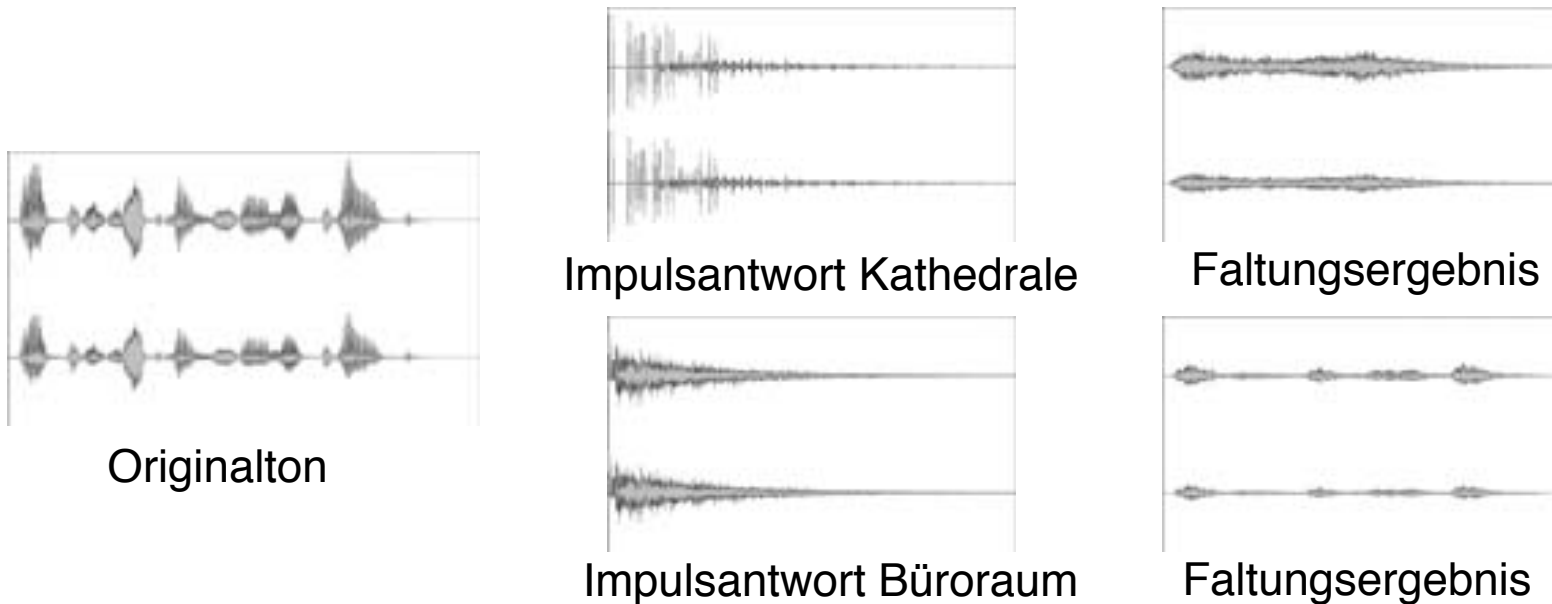
Kurzer Impuls bestimmter Frequenz (Klatschen, Schuss)

Spektrum des reflektierten Signals beschreibt Übertragungseigenschaften des Raums (*Impulsantwort*)

Mathematische Berechnung der Raumwirkung:

– *Faltung (convolution)* des Eingangssignals mit Impulsantwort

Beispiel:



# Phasenorientierte Bearbeitung

Sehr kurze Verzögerungen (unter 30 ms) werden als Phasenveränderungen wahrgenommen und beeinflussen den Gesamtklang

Chorus:

Sehr schnelle, minimal in der Höhe veränderte Signalarückführung  
Lässt Klang voller erscheinen

Flanging:

Noch kleinere Verzögerungszeit (8 ms)  
Tonhöhe konstant - Überlagerung mit Originalsignal  
Feedback: Effektsignal wird an Eingang zurückgeführt  
Verwendung z.B. bei Gitarrenklängen

Phasing:

ähnlich zu Flanging, aber ohne Feedback  
synthetischer Klang

# Restauration

Fehler auf alten Tonträgern:

Rauschfehler (*Noise, Hiss*)

Clickfehler (*Clicks*)

Knistern (*Crackles*)

- **Denoising:**

"Fingerprint" (Spektrum) des Rauschens wird bestimmt  
dann exakt diese Frequenzen ausgefiltert

- **Declicking:**

Signallücke durch Interpolation (oder zweiten Stereokanal) ersetzen

- **Decracking:**

Wiederholtes Declicking, auch automatisch ausgeführt



# 4. Audiotechnik und Tonbearbeitung

4.1 Grundlagen der Audiotechnik

4.2 Analoge Audiotechnik

4.3 Raumklang

4.4 Digitale Audiotechnik

4.5 Programmierung für Audioverarbeitung



Literatur:

<http://java.sun.com/products/java-media/sound/>

<http://www.jsresources.org>

# Java Sound API

## "Low-level" API

zur Steuerung der Ein- und Ausgabe von Tonmedien  
umfasst Funktionen für digitale Audioinformation und für MIDI-Daten  
erweiterbare Basis, keine ausgefeilten Editor-Funktionen o.ä.

## Verwandte Java-Technologien:

Java Media Framework (JMF)

auf höherer Ebene angesiedelt

einfachere Lösung für Abspielen von Tonmedien

Synchronisation mit anderen Medien (v.a. Video)

## Pakete des Java Sound APIs (in Standard-Java-Installation enthalten):

`javax.sound.sampled`

`javax.sound.midi`

# Digitales Audio in Java Sound

Hauptfokus in `javax.sound.sampled`:

- Transport von Audiodaten (Aufnahme und Wiedergabe)

- Verwaltung von Puffern

- Mischen von Daten

- Steuerung: Start, Anhalten, Aussetzen

Prinzipiell mit der Schnittstelle möglich:

- Direkte Bearbeitung von Sample-Information

  - (z.B. selbstgeschriebene Synthesizer- und Filter-Funktionen)

Default-Implementierung eingeschränkt in Funktionalität:

- z.B. Einlesen von WAV-Dateien, aber nicht von MP3- oder MPEG4-AAC-Dateien

- Funktionalität erweiterbar durch „Service Provider Interface“

  - Plug-Ins mit zusätzlicher Codec-Funktionalität (teilweise) verfügbar

# Audio-Formate

Java Sound befasst sich mit formatierten Audio-Daten

Ein Audio-Format wird durch ein Objekt der Klasse **AudioFormat** beschrieben und enthält die folgenden Angaben:

Codierungsverfahren (normalerweise Pulscode-Modulation (PCM))

Zusammenhang zwischen Klangdruck und Signal:  
(PCM/linear,  $\mu$ -Gesetz, a-Gesetz)

Bei PCM: Samplewerte mit/ohne Vorzeichen (*signed/unsigned*)

Anzahl von Kanälen (z.B. 2 für Stereo)

Abtastrate (samples/second)

Auflösung (bits/sample)

Paketgrösse (*frame size*) (bytes)

Paketrate (*frame rate*) (frames/s)

Byte-Lesereihenfolge (*byte order*) (big-endian/little-endian)  
(Wo steht das höchstwertige Bit?)

# AudioInputStream

`javax.sound.sampled.AudioInputStream`

Abgeleitet von der abstrakten Oberklasse `java.io.InputStream`

Byteweise lesbare Datenströme

`read()`, `skip()`, `close()`

`markSupported()`, `mark()`, `reset()`:

Springen zu markierter Position

Methoden zum Öffnen einer Audiodatei:

`javax.sound.sampled.AudioSystem`

`AudioInputStream getAudioInputStream(File file)`

Es werden ggf. mehrere Parser für unterstützte Audio-Formate eingesetzt

Konversion von Streams in andere Formate möglich

Ein `AudioInputStream` hat immer ein festgelegtes **AudioFormat**

Das Format wird z.B. beim Öffnen einer Audio-Datei festgelegt und im Streamobjekt gespeichert.

`AudioFormat getFormat()`

# Beispiel: Öffnen einer Audio-Datei

```
import javax.sound.sampled.*;
...

public static void main(String[] args) {

    String strFilename = args[0];
    File soundFile = new File(strFilename);
    AudioInputStream audioInputStream = null;
    try {
        audioInputStream =
            AudioSystem.getAudioInputStream(soundFile);
    }
    catch (Exception e) {};
    AudioFormat audioFormat =
        audioInputStream.getFormat();
    System.out.println(audioFormat);
    ...
}
```

# Line, Port und Mixer

Ein *Port* beschreibt einen Ein- oder Ausgang für einen Strom von Audio-Daten zu einem (virtuellen oder physikalischen) Gerät

Eine *DataLine* beschreibt einen Datenstrom, der steuerbar ist (Start/Stop).

Ein *Mixer* ist die Java-Sound-Abstraktion für alle tonverarbeitenden Komponenten mit Ein- und Ausgabeports

Kann einem realen physikalischen Gerät entsprechen oder rein in Software realisiert werden

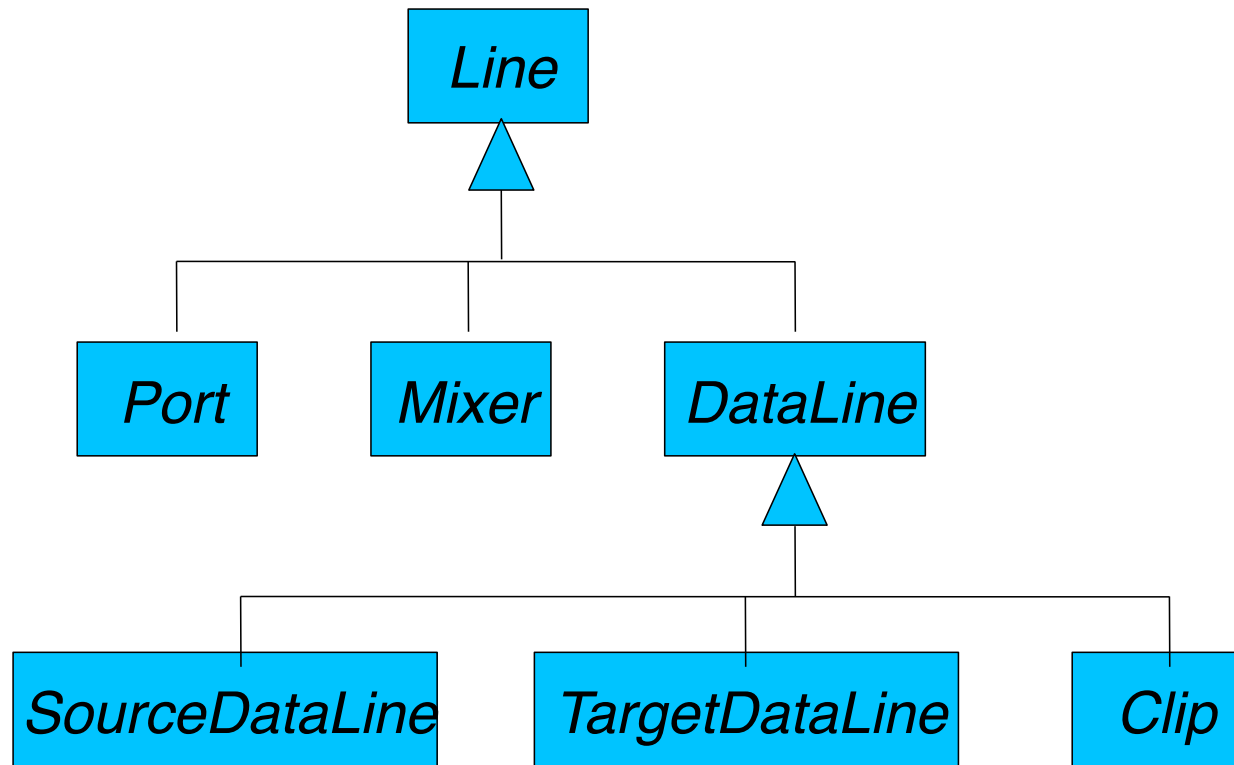
Wichtiger Spezialfall eines Mixer-Objekts: Mischpult-Funktion

Anwendungsprogramme können Daten entweder auf Eingabe-Ports abliefern oder auf Ausgabeports abholen (streaming)

Alternativ können Daten für Eingabeports aus Dateien gelesen werden und für Ausgabeports in Dateien geschrieben werden (audio files)

Das Konzept der *Line* verallgemeinert Ein- und Ausgabeports und Mixer zu beliebigen Bestandteilen einer "Audio-Pipeline"

# Schnittstellen-Hierarchie für Audio-Pipelines



Gepufferte Wiedergabe  
einer Audioquelle

Gepufferte Aufnahme  
einer Audioquelle

Wiedergabe einer  
Audioquelle, die  
komplett in den Speicher  
vorgeladen wird



# Audio-Ressourcenverwaltung

Typischerweise ist Audioverarbeitung an Systemressourcen gebunden

Installierte Geräte

Pufferspeicher in Systembereichen

Erzeugung eines Line-Objekts ausschließlich über zentrale Verwaltung:

```
javax.sound.sampled.AudioSystem
```

Anfragefunktionen, welche Ressourcen vorhanden sind

Es gibt immer einen Standard-Mixer, der auf das eingebaute Hardware-Audio-System abgebildet wird

Erzeugung eines speziellen Line-Objekts

Zur Beschreibung von Line-Objekten:

```
javax.sound.sampled.Line.Info mit Unterklassen:
```

```
    DataLine.Info, Port.Info
```

```
javax.sound.sampled.Mixer.Info
```

# Erzeugung eines Line-Objekts

Anfrage an AudioSystem, unter Angabe von LineInfo-Information für das gewünschte Line-Objekt

welcher Typ:

*SourceDataLine* (Wiedergabe)

*Clip* (wiederholte Wiedergabe)

*TargetDataLine* (Aufnahme)

welches Audioformat

Beispiel (Ausgabe von Audioinformation):

```
AudioFormat audioFormat =  
    audioInputStream.getFormat();  
DataLine.Info info = new DataLine.Info  
    (SourceDataLine.getClass(), audioFormat)  
AudioSystem.getLine(info);
```

# Operationen zur Nutzung eines Line-Objekts

Bisher nur die Verfügbarkeit des gewünschten Typs von `Line` geprüft.

Nächste Schritte:

- Reservierung der Ressource mit `open ()`

  - Reservieren des benötigten Pufferbereichs

  - kann bei gleichzeitigen Anforderungen anderer Programme an Ressourcenmangel scheitern

  - bei `SourceDataLine` Angabe eines Audioformats möglich

- Bei `DataLine` (d.h. steuerbarem Strom):

  - verschiedene Zustände:

    - `running = true/false` (Übertragung ein/aus)

  - Bei Erzeugung: `running = false`

  - Starten durch `start ()`

Die meisten folgenden Beispiele basieren auf [www.jsresources.org](http://www.jsresources.org) !

# Beispiel: Öffnen einer Audio-Line zur Wiedergabe

```
public static void main(String[] args) {
    String strFilename = args[0];
    File soundFile = new File(strFilename);
    AudioInputStream audioInputStream = null;
    try {
        audioInputStream =
            AudioSystem.getAudioInputStream(soundFile);
    }
    catch (Exception e) {};

    AudioFormat audioFormat = audioInputStream.getFormat();
    SourceDataLine line = null;
    DataLine.Info info =
        new DataLine.Info(SourceDataLine.class, audioFormat);
    try {
        line = (SourceDataLine) AudioSystem.getLine(info);
        line.open();
    }
    catch (Exception e) {};
    line.start();
    ...
}
```

# Beispiel: Audiowiedergabe aus Datei

...

```
byte[] abData = new byte[EXTERNAL_BUFFER_SIZE]; //128k
```

```
int nBytesRead = 0;
```

```
while (nBytesRead != -1) {
```

```
    try {
```

```
        nBytesRead =
```

```
            audioInputStream.read(abData, 0, abData.length);
```

```
    }
```

```
    catch (Exception e) {};
```

```
    if (nBytesRead >= 0)
```

```
        line.write(abData, 0, nBytesRead);
```

```
    }
```

...

SourceDataLine.write()

Puffer

Startindex

Endindex

# Nebenläufigkeit in Java: Threads

Häufig laufen in der Medientechnik mehrere Verarbeitungsketten nebeneinander ab

Beispiel: Aufnahme eines Audiosignals || Warten auf Ende-Befehl

Java-Threads:

"leichtgewichtige Prozesse" – innerhalb eines Betriebssystemprozesses

Jeder Thread implementiert die Schnittstelle `java.lang.Runnable`

Einzige Methode in `Runnable`:

```
public void run()
```

Standard-Implementierung von `Runnable`:

```
java.lang.Thread implements Runnable
```

Starten eines Threads - d.h. der `run()`-Methode:

```
t.start()
```

# Beispiel: Ein nebenläufiger Java-Audio-Recorder

```
public class SimpleAudioRecorder1 extends Thread {  
    private TargetDataLine          m_line;  
    private AudioFileFormat.Type    m_targetType;  
    private AudioInputStream        m_audioInputStream;  
    private File                    m_outputFile;  
  
    public SimpleAudioRecorder1(TargetDataLine line,  
        AudioFileFormat.Type targetType, File file) {  
        m_line = line;  
        m_audioInputStream = new AudioInputStream(line);  
        m_targetType = targetType;  
        m_outputFile = file;  
    }  
    ...  
    public void run() {  
        try {  
            AudioSystem.write(  
                m_audioInputStream, m_targetType, m_outputFile);  
        }  
        catch (Exception e) {}  
    } ...  
}
```

# Starten und Anhalten der Aufnahme

```
public class SimpleAudioRecorder1 extends Thread {  
    private TargetDataLine          m_line;  
    private AudioFileFormat.Type    m_targetType;  
    private AudioInputStream        m_audioInputStream;  
    private File                    m_outputFile;  
  
    ...  
  
    public void start() {  
        m_line.start(); // Startet die Aufnahme im Audio-System  
                        // (TargetDataLine)  
        super.start(); // Startet den Datentransfer-Thread  
                      // d.h. self.run()  
    }  
  
    public void stopRecording() {  
        m_line.stop();  
        m_line.close();  
    }  
}
```



# Beispiel: Direkte Bearbeitung von Samples

```
public class SineOscillator extends AudioInputStream {
    public SineOscillator
        (float fSignalFrequency, float fAmplitude,
         AudioFormat audioFormat, long lLength) {
        super(new ByteArrayInputStream(new byte[0]),
              new AudioFormat(AudioFormat.Encoding.PCM_SIGNED,...), lLength);
        ...
        m_abData = new byte[nBufferLength];
        for (int nFrame = 0; nFrame < nPeriodLengthInFrames; nFrame++) {
            float fPeriodPosition =
                (float) nFrame / (float) nPeriodLengthInFrames;
            float fValue =
                (float) Math.sin(fPeriodPosition * 2.0 * Math.PI);
            int nValue = Math.round(fValue * fAmplitude);
            int nBaseAddr = (nFrame) * getFormat().getFrameSize();
            m_abData[nBaseAddr + 0] = (byte) (nValue & 0xFF);
            m_abData[nBaseAddr + 1] = (byte) ((nValue >>> 8) & 0xFF);
            m_abData[nBaseAddr + 2] = (byte) (nValue & 0xFF);
            m_abData[nBaseAddr + 3] = (byte) ((nValue >>> 8) & 0xFF);
        }
        m_nBufferPosition = 0;
    }
}
```