**Medientechnik Übung**

Prof. Hansjörg Mixdorff, Angelika Hönemann

**Übung 1: Audio-Grundlagen: Abtastung, Aliasing, Quantisierung**

**Einführung:** Zur Verdeutlichung der Vorgänge bei der digitalen Audio-Verarbeitung werden verschiedene Manipulationen auf dem digitalisierten Audiosignal (in Form von Microsoft-WAV Dateien) selbst programmieren.

*Die Wave-Dateien werden von mir nach Email-Meldung jeder Gruppe separat zugeschickt!*

Die Java-Methoden read_wav und write_wav aus der Klasse WavFile werden zum Ein-und Auslesen von Wave-Dateien zur Verfügung gestellt (moodle). Diese werden aus der main-Methode der Klasse wave_io aufgerufen. Wave_io, bzw. main wird daher entsprechend um einen Programmblock erweitert werden, der die genannten Manipulationen realisiert. Die Klasse wave_io in der vorliegenden Form liest lediglich eine Wave-Datei ein und speichert sie gegebenenfalls unter einem anderen Namen wieder ab.

Wir werden die IDE Eclipse nutzen, um wave_io zu erweitern, d.h zunächst muss wave_io in Eclipse als Java Project angelegt und kompiliert werden. Das Programm kann dann unter "Run configurations" mit den folgenden Argumenten ausgeführt werden (s. Eclipse.pdf).

**Argumente:**

a. input.wav (Wave-Datei 'input.wav' wird eingelesen und Header-Daten werden angezeigt)
b. input.wav output.wav (Wave-Datei 'input.wav' wird eingelesen und in die Datei 'output.wav' kopiert)

Zur Darstellung und Bearbeitung der Wave-Dateien wird Audacity empfohlen, aber es können auch andere Wave-Editoren verwendet werden, da ja jeder auf seinem eigenen Rechner arbeitet.

**1. Aufgabe – Audiodateien erzeugen und einlesen**

a. Schneide aus den dir zugeschickten Audio-Files ab dem Zeitpunkt jeweils ein Stück mit der Länge 5 Sekunden und speichere dieses als WAV-Datei ab. Parameter für Musik: fa=44,1 kHz, stereo, für Sprache: fa=8 kHz mono, beide 16 bit Auflösung. Beim Schneiden achtest du darauf, dass der Schnitt am Beginn einer musikalischen Figur bzw. eines Satzes liegt.
b. Erkläre, warum die Audio-Files unterschiedliche Abtastfrequenzen haben.
c. Lies die Musik- und die Sprachdatei mit wave_io ein und erkläre die Angaben im Header!
d. Berechne die Bitrate für die beiden Dateien!

Benenne die Dateien „Musik_NameArbeitsgruppe.wav" und „Sprache_NameArbeitsgruppe.wav" und schicke mir die beiden WAV-Dateien unter Nennung deiner Arbeitsgruppe im Betreff per Mail zu (In den folgenden Übungen wirst du diese Dateien oft als Ursprungssignale verwenden. Sollten sie verloren gehen, kannst du sie notfalls wieder von mir erhalten.). Du bekommst daraufhin zwei WAV-Dateien mit Testsignalen zugesendet, die du bei den folgenden Aufgabenpunkten benötigst, sine_hiXX.wav und sine_loXX.wav.

*Ins Übungsprotokoll: Sourcecod*e (jeweilige Funktion als Text), Musik- und Sprachaufnahmen, Erläuterungen und Berechnungen

**2. Aufgabe - Aliasing**

a. Modifiziere wave_io dahingehend, dass die Samples in der WAV-Datei in eine (lesbare) ASCII-Datei geschrieben werden. Lies die von mir geschickten Sinusdateien (Sampling-Frequenz: 16 kHz) ein und bestimme aus den resultierenden Zahlenfolgen in der ASCII-Datei die Frequenz der Sinus-Schwingungen. Begründe!

b. Überprüfe deine Schätzung mit dem Spektralanalyse-Tool GRAM. (Vorgehensweise: Menüpunkt Analyze File, Einstellungen: Freq Scale: Linear, FFT Size: 512, Time scale: 1 msec)

c. Bei der zeitlichen Diskretisierung eines Analogsignals muss das sogenannte Abtasttheorem eingehalten werden. Wie lautet es und wie lässt sich der Grenzfall, für den es gerade noch gilt, illustrieren? Erstelle hierzu eine Zeichnung und erläutere.

d. Bei herkömmlichen Soundkarten tritt systembedingt kein Aliasing auf, weil das Audiosignal stets geeignet vorbehandelt wird. Wie sieht diese Vorbehandlung aus?

e. Mit einem kleinen Trick lässt sich Aliasing jedoch nachweisen. Diese auch als Down-Sampling bekannte Methode besteht darin, dass man bei einer WAV-Datei z.B. jeden zweiten Abtastwert wegwirft. Man erhält so eine Wellenform, die genau die Hälfte der ursprünglichen Abtastfrequenz aufweist. Wenn man das Signal nicht vorher bandbegrenzt hat, können Aliasing-Verzerrungen hörbar werden.

   Modifiziere wave_io dahingehend, dass vom eingelesenen Audiosignal jeder zweite Abtastwert verworfen wird und das resultierende Signal abgespeichert wird. Der Header muss natürlich entsprechend verändert werden!

f. Wende das erstellte Programm auf die von mir geschickten Sinusdateien an (sine_hiXX.wav und sine_loXX.wav) an. Welche Frequenzen erscheinen nach dem Down-Sampling? Was würde passieren, wenn man geeignet bandbegrenzen würde?

*Ins Übungsprotokoll: Sourcecode (jeweilige Funktion als Text), originale und modifizierte Sinusfunktionen, Screenshot GRAM, Zeichnung, Erläuterungen und Berechnungen, Listung der Abstastwerte für mindestens eine Periode für die beiden Sinusschwingungen*

**3. Aufgabe - Bitreduzierung**

a. Die herkömmlichen PC-Soundkarten arbeiten meist entweder mit 16 oder 8 bit-Auflösung. Wie groß ist die Anzahl bei diesen beiden Werten darstellbaren Amplitudenwerten?

b. Modifiziere wave_io dahingehend, dass die Bitanzahl reduziert wird. Dazu werden alle Samples durch eine Potenz von 2 geteilt (Integer-Division ohne Rest). Damit das resultierende Signal nicht leiser wird als das Original, wird die Operation durch Multiplikation mit derselben 2er Potenz kompensiert. Zu beachten: Der Datentyp hat nach wie vor 16 bit!

c. Mit dem entstandenen Programm sollen nun die in Aufgabe 1 erzeugten Wave-Dateien (Sprache und Musik) bitreduziert werden. Ab welcher Bitanzahl tritt eine hörbare, also deutliche Verschlechterung der Qualität ein? Bei wie viel Bit ist das Sprachsignal noch verständlich?

d. Was charakterisiert das entstehende Quantisierungsgeräusch bei der Bitreduzierung und macht es besonders störend?

e. Modifiziere dein Programm noch einmal so, dass auch das Differenzsignal zwischen Original und bitreduziertem Signal, d.h. der Quantisierungsfehler ausgegeben werden kann. Dabei musst du bedenken, dass z.B. bei der 1 Bit Reduzierung das Quantisierungsrauschen nur von -1 bis +1 verlaufen würde. Dieser Wertebereich wäre viel zu klein, als dass man das Rauschen beim Abspielen als 16bit-Wert noch hören könnte. Daher muss das Rauschen durch Multiplikation mit einer 2er Potenz verstärkt werden. In anderen Worten: Hat man vorher durch 2^n geteilt, sollte man das Differenzsignal vor dem Abspeichern mit 2^(16-n-1) multiplizieren. So ist sichergestellt, dass der Verstärkungsfaktor mit der Anzahl der gelöschten Bits kleiner wird.

f. Welchen Charakter hat das Rauschen bei einer Reduktion um 1bit und wie verändert es sich bei zunehmender Bit-Reduktion?

*Ins Übungsprotokoll: stets Sourcecode (jeweilige Funktion als Text),*

*Für 3c: bitreduzierte Sprach- und Musikaufnahmen, bei denen jeweils das Quantisierungsrauschen hörbar wurde mit Angabe der verbleibenden Bitzahl, Spektrogramme GRAM, Erläuterungen*

*Für 3e: Differenzsignale, d.h. Quantisierungsfehler bei **1 bit Reduzierung** für die Sprach- und Musikaufnahmen, außerdem Differenzsignale, d.h. Quantisierungsfehler für die Sprach- und Musikaufnahmen, bei denen in 3c das Quantisierungsrauschen hörbar wurde. Spektrogramme GRAM zu allen Aufnahmen, Erläuterungen*