

Multimediasysteme - Spektrogrammfilterung

Christopher Holzweber*, Stefan Paukner†, Simon Sternbauer‡, Benedikt Lichtneker§

May 31, 2019

*Matr Nr: 11803108

†Matr Nr: 11808822

‡Matr Nr: 11812499

§Matr Nr: 11809953

Contents

1 Realisierung	4
1.1 Package Application	4
1.1.1 Spektroapp.java	4
1.2 Package View	4
1.2.1 UserFrame.java	4
1.2.2 ImagePanel.java	5
1.2.3 Startingpanel.java	5
1.2.4 Waveform.java	6
1.3 Package Model	6
1.3.1 WAVConverter.java	6
1.3.2 WAVFormatException.java	7
1.3.3 AudioPlayer.java	7
1.3.4 PlayThread.java	7
1.3.5 ImageFilter.java	7
1.3.6 MP3WavConverter.java	8
1.3.7 SpektogramImage.java	9
2 Spektrogram	10
2.1 Spektrogram mittels FFT	10
2.1.1 Auflösungsproblematik	10
3 Projektlogo	12
3.1 Erstellung des Icon	12
3.2 Erstellung des Loadingscreens	12
3.3 Farbschema	12
4 Verzeichnisse	14
5 Referenzen	14

Abstract

In diesem Projekt werden WAV oder MP3 Files zur Bildfilterung benutzt. Dabei werden sogenannte Spektrogramme erzeugt, welche Daten der Audio Files in visualisierter Form darstellen. Aufgrund dieser Spektrogramme werden anschließend Bilder im .jpg oder .png Format gefiltert. Somit werden Eigenschaften verschiedener Musikgenres auf Bilder übertragen. Mit folgender Grafik werden die Projektschritte beschrieben:

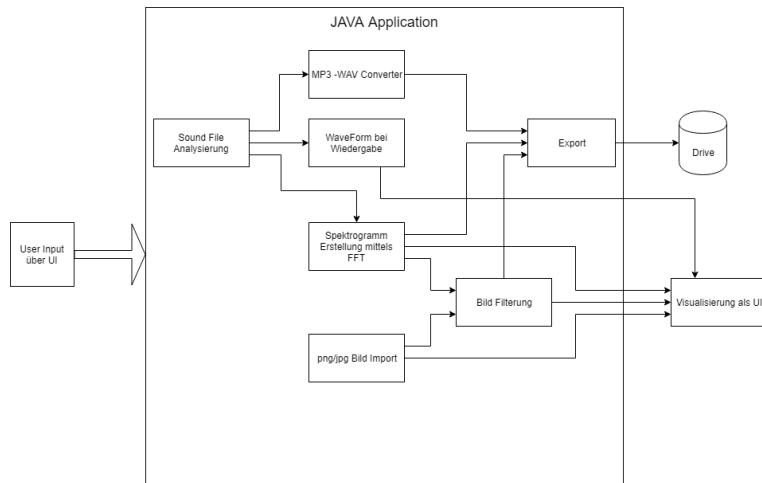


Figure 1: Projektskizze

Das entsprechende Soundfile und das zu bearbeitende Bild können interaktiv vom User ausgewählt werden. Dafür ist ein JAVA UI zuständig. Im ersten Schritt sollen Soundfiles analysiert werden, dies bedeutet, es wird mittels FFT bzw. Short Time FFT ein Spektrogramm des Liedes/der Musik/des Klanges für eine bestimmte Zeitspanne erstellt werden. Als Analysetools wurde hierzu Java verwendet.

Dieses Spektrogramm wird anschließend dem User über die bedienbare UI visualisiert.

Ebenso ist es dem User möglich, ein .png oder .jpeg Bilder auszuwählen. Diese werden anschließend auf ein gewisses Format zugeschnitten.

Sobald eine WAV oder MP3-Datei geladen wurde, ist es möglich ein Spektrogramm zu erstellen. In einem Spektrogramm werden in dem Zeitabschnitt gemessenen Frequenzen, über der Zeit sichtbar gemacht. Je nach Amplitude erhält der Wert zu einem gewissen Zeitpunkt eine andere Farbe. Diese Intensität ändert sich, je stärker eine Amplitude einer Frequenz vorhanden ist.

Mit Hilfe dieses Spektrogrammes kann man nun das gewählt Bildfile mit „Musik filtern“. Je nach Auswirkung/Genre/... der gewählten Musik, wird somit ein anderes Bild entstehen. Je ruhiger die Musik ist, desto weniger Kontrast soll das Bild bekommen, je mehr Frequenzen unterschiedlicher Amplitude das Musikstück hat, desto bunter wird das Bild und desto lauter das Musikstück, desto mehr Kontrast/Schärfe soll ein Bild bekommen. Somit soll in gewisser Weise „Kunst auf Kunst“ übertragen werden.

Ebenso ist es möglich die gewählte Musik im integrierten Player abzuspielen, die gefilterten Bilder zu exportieren. Ebenso ist eine Konversation von WAV auf MP3 und umgekehrt möglich, diese Audio Files sind ebenso exportierbar.

Eine mögliche Erweiterung wäre noch, das Bild via Sozialmedia zu teilen und diverse Daten über das gewählte Musikfile auszugeben.

1 Realisierung

Die Softwarerealisierung wurde hauptsächlich mittels objektorientierten Entwicklungssprachen getätigt. Java unter anderem bietet wie auch MatLab Möglichkeit zur FFT Filterungen. In den folgenden Abschnitten werden die einzelnen .java Files erläutert.

1.1 Package Application

1.1.1 Spektroapp.java

In dieser Klasse befindet sich die main Methode, um das Programm zu starten. Dabei wird ein neuer Userframe initialisiert.

1.2 Package View

1.2.1 UserFrame.java

Diese Klasse erzeugt die eigentliche UI Umgebung.

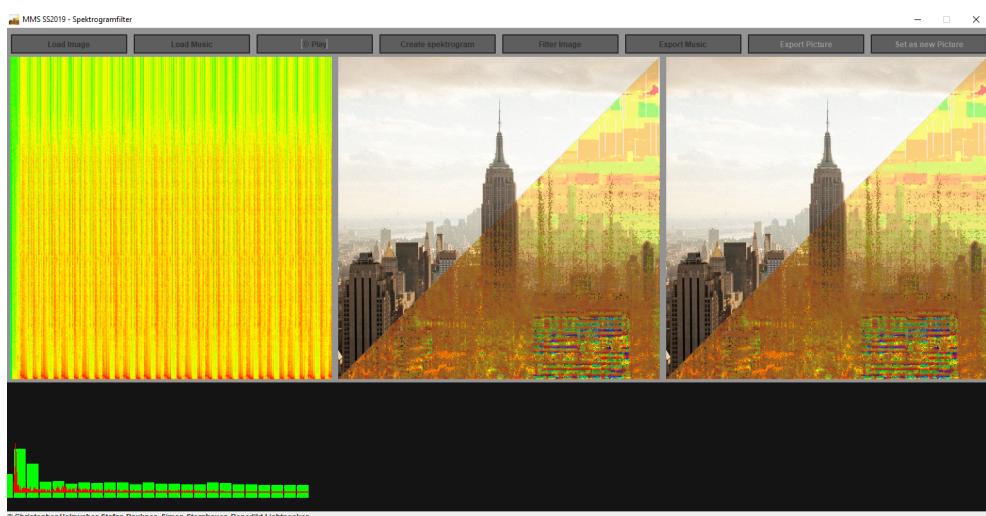


Figure 2: Eine Übersicht der zur Verfügung stehenden Buttons

Die Benutzeroberfläche kann wie folgt bedient werden:

- Load Image - Ladet mit Hilfe eines Filechoosers ein .png oder .jpg Image. Wird ein falsches Format gewählt, so wird das Window neu gestartet.
Sobald ein Image geladen wurde, wird dieses auch nun in der UI angezeigt. In der Mitte des UI befindet sich das Originalimage.
- Load Music - Ladet mit Hilfe eines Filechoosers ein .wav oder .mp3 File. Wird ein falsches Format gewählt, so wird das Window neu gestartet. Nach dem Laden wird vom System automatisch eine Kopie im jeweils anderen Dateiformat erstellt. Für die weitere Verarbeitung sind ausschließlich die .wav Dateien wichtig.

- Play - Mit Hilfe dieses Buttons ist es möglich das gewählte Soundfile abzuspielen. Wird der Button während dieses Vorganges erneut gedrückt, so stoppt die Wiedergabe, wird der Knopf erneut betätigt, beginnt sie wieder von vorne. Es handelt sich also um einen Start/Stop Button. Siehe Klasse AudioPlayer.java
- Create spektrogram - Sobald ein .wav oder .mp3 File geladen wurde, ist es möglich ein Spektrogram zu erstellen. Das erstellte Spektrogramm erscheint im linken Panel.
- Filter image - Sobald ein Spektrogram UND ein Bild geladen wurden, ist es möglich das Bild mit Hilfe des Spektrograms zu filtern. Das gefilterte Bild erscheint nun im rechten Panel.
- PlayButton - Sobald ein .wav oder .mp3 File geladen wurde, ist es möglich das gewählte File auch abzuspielen.
- Export - Sobald entweder eine .wav oder .mp3 Datei geladen wurde, steht über diesen Button der Export in ein zu wählendes Verzeichnis im jeweils anderen Format zur Verfügung.
- Export Picture - Sobald der erstelle Spektrogramm auf das geladene Bild angewendet wurde, ist es möglich, dass neu kreierte Bild zu exportieren und abzuspeichern. Dabei wird zuerst das Verzeichnis gewählt und der Name des Bildes festgelegt. Exportiert wird es immer als jpeg.
- Set as new Picture - Dieser Button ersätzt das vorhanden ursprüngliche Bild mit dem gefilterten Bild. Somit ist es möglich, eine neu Musik zu laden und auf ein bereits bearbeitetes Bild anzuwenden ohne dies vorher exportieren und neu laden zu müssen.

1.2.2 ImagePanel.java

In dieser Klasse ist es möglich, die angezeigten Bilder zu setzen oder mit Methoden zu bekommen. Für jedes der drei angezeigten Bilder(Spektrogram, Originalbild, gefiltertes Bild) wird ein eigenes solches Panel erstellt.

1.2.3 Startingpanel.java

Erzeugt den anfänglichen Loadingscreen und starten anschließend das UserFrame. Zeitdauer ist mit drei Sekunden bemessen. Damit es mehr wie ein Ladevorgang aus sieht wurde der obere Teil des Frames entfernt. Dies gelingt durch das setzen des Property setUndecorated(true). Ausschnitte aus dem eigentlich Screen:



Figure 3: Starting Panel Auszüge

1.2.4 Waveform.java

Diese Klasse zeigt beim Abspielen des gewählten Files in rot die Spektren passend zum Abspielzeitpunkt mit linearer Darstellung der Frequenz.

Die grünen Balken fassen die Frequenzen in kritische Bänder nach Heinrich Barkhausen zusammen und gibt deren durchschnittliche Amplitude aus.

1.3 Package Model

1.3.1 WAVConverter.java

In dieser Klasse werden die Daten aus dem gewählten File extrahiert. Dabei ist es wichtig zu wissen, dass das WAV File einen sogenannten Header besitzt, welcher alle wichtigen Informationen über das File beinhaltet. Diese Informationen werden über die JavaSound Klasse AudioFileFormat ausgewertet.¹

Entspricht das WAV File dem gewünschten Standard - 2 Channels, 16Bit/Sample - 4Byte Blockalign, Abtastrate 44100Hz, so wird die Konvertierung durchgeführt, ansonsten eine sogenannte WAVFormatException geworfen.

Je ein Samplevorgang wird auf einen double Wert reduziert und anschließend ein Double-Array für die FFT zurückgegeben. Besteht das File aus 2 Channels, und 16 Bit Samplegröße je Kanal, so ergeben sich 4Byte Alignments. Dabei sind 2Byte für den linken und 2Byte für den rechten Kanal gespeichert. Für jeden Kanal werden durch Shiftoperationen die direkten Werte ermittelt und anschließend wird der Mittelwert der beiden Kanäle ermittelt (ergibt ein Monosignal). Dieses Array wird anschließend dem aufrufenden Programm zurückgeben.²

¹https://de.wikipedia.org/wiki/RIFF_WAVE

²<https://stackoverflow.com/questions/39295589/creating-spectrogram-from-wav-using-fft-in-java>

1.3.2 WAVFormatException.java

Wird geworfen, wenn das gewählte WAV nicht dem gewünschten Standard entspricht.

1.3.3 AudioPlayer.java

Diese Klasse startet oder stoppt die Wiedergabe des gewählten AudioFiles. Dies geschieht durch erzeugung eines neuen Thread Objects, welches jederzeit über die Methode stopplay() wieder beendet werden kann.

1.3.4 PlayThread.java

Gibt das AudioFile mittels Javax Sound wieder³. Dabei wird der Pfad des gewählten AudioFiles übergeben und anschließend über die Lautsprecher wiedergegeben. Dieser Thread kann jederzeit durch erneutes drücken des Playbuttons beendet werden, bzw. beendet von selbst, sobald das File fertig abgespielt wurde.

1.3.5 ImageFilter.java

In dieser Klasse wird das Bild im rechten Panel erstellt. Dabei werden über die Methode filter() die beiden Bilder(Spekrogramm und Originalbild) übergeben. Als Rückgabewert erhält man ein neues Bild, welches dem Originalbild entspricht, jedoch mit unterschiedlicher Sättigung und Helligkeit. Die Sättigung der einzelnen Pixel des Originalbildes werden nämlich durch das Spekrogramm bestimmt. Dort wo im Spekrogramm starke Amplituden zu erkennen sind, wird die Sättigung angepasst. Dabei ist das Spekrogramm so zu verstehen, dass Grün eine geringe Amplitude bedeutet und Rot eine sehr hohe Amplitude. Dazwischen liegen die Farben grob übergehend von Gelb auf Orange. Die Sättigung beschreibt „wie stark sich ein farbiger Reiz von einem achromatischen Reiz unabhängig von dessen Helligkeit unterscheidet“, [1] also sein Abstand von der Unbunt-Achse (Schwarz-Weiß-Achse). Damit haben alle Farbtöne (Bunntöne) eine Sättigung bis zu 100 %, und Weiß, Grau und Schwarz eine Sättigung von 0.⁴

Gearbeitet wird dabei im HSB Farbraum⁵⁶, in welchem die Sättigung als eigener Parameter festgelegt werden kann. In Java wird somit jedes Pixel mittels 3 Float-Werten (Range 0.0 - 1.0) beschrieben: Der HUE,SATURATION,BRIGHTNESS. Der Wert der HUE wird mit 360 multipliziert, um den Korrekten Wert des Kreises zu erhalten. Zu beachten gilt, dass der Wert V für Brightness B steht. Um zur Korrekten Farbe zu kommen geht man über den Wert H für HUE.

Um das Bild auch optisch besser dazustellen, wurde nicht nur grob nach mathematischen Formel vorgegangen, sondern die Bereiche, ab welche eine Amplitude als hoch zählt und auch deren Art der Filterung wurden von den uns Entwicklern selbst gewählt, so wie es für das Menschliche Auge am besseren zu sehen ist.

³<https://stackoverflow.com/questions/2416935/how-to-play-wav-files-with-java>

⁴<https://de.wikipedia.org/wiki/Farbs%C3%A4ttigung>

⁵<https://de.wikipedia.org/wiki/HSV-Farbraum>

⁶https://en.wikipedia.org/wiki/HSL_and_HSV

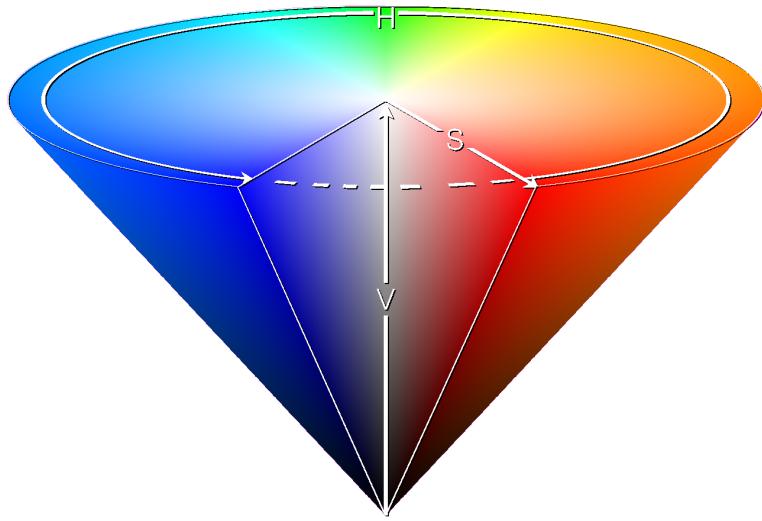


Figure 4: HSB Farbraum

Diese Berechnung erfolgt im Codeabschnitt:

```
// IF GREEN – PIXEL OF ORIGINALPICTURE STAYS THE SAME!

if((spekhsbvals[2]>0.4) && (spekhsbvals[2]<=0.65)) { //MIDDLE amplitude YELLOW
    orighsbvals[1] = (float) 0.5;
}
if((spekhsbvals[2]>0.65 && spekhsbvals[2]<=0.8)) { //NEARLY HIGH amplitude ORANGE
    orighsbvals[1] = (float) 0.8;
    orighsbvals[2] = (float) 0.6;
}
if((spekhsbvals[2]>0.8)) { //HIGH amplitude RED
    orighsbvals[1] = (float) 1.0;
    orighsbvals[2] = (float) 1.0;
}
```

Somit ergeben sich derartige Filterungen:

1.3.6 MP3WavConverter.java

In dieser Klasse stehen zwei statische Methoden zur Verfügung, welche eine verfügbare MP3 oder WAV-Datei in das jeweils andere Format exportieren und in einem Ordner converted im aktueller User Directory ablegen. Diese Datei kann über die GUI nachfolgend mittels "Export" an einen beliebigen Ort kopiert werden.

Für die Umwandlung wird die Java-Bibliothek JAVE-1.0.2 von Sauronsoftware[9] verwendet. Jene bietet neben einer Vielzahl an diverse Umwandlungen von Audio- und Videodaten auch eine einfache Konvertierung von MP3 nach WAV und umgekehrt an.

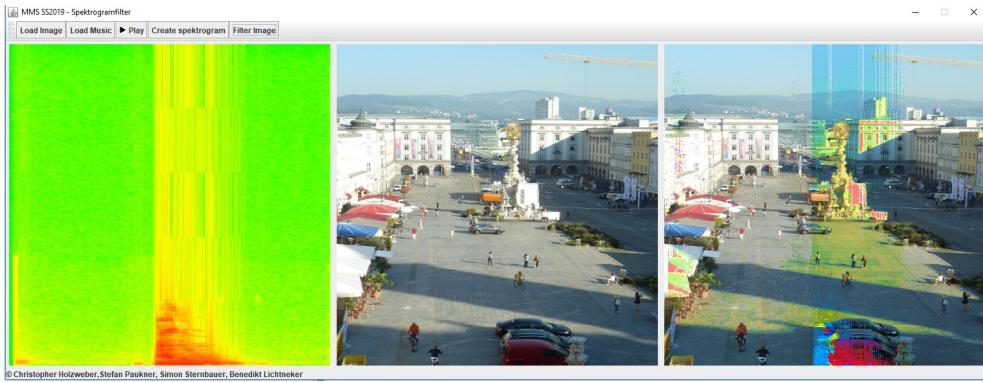


Figure 5: Spektrogramm mit sehr vielen geringen Amplituden

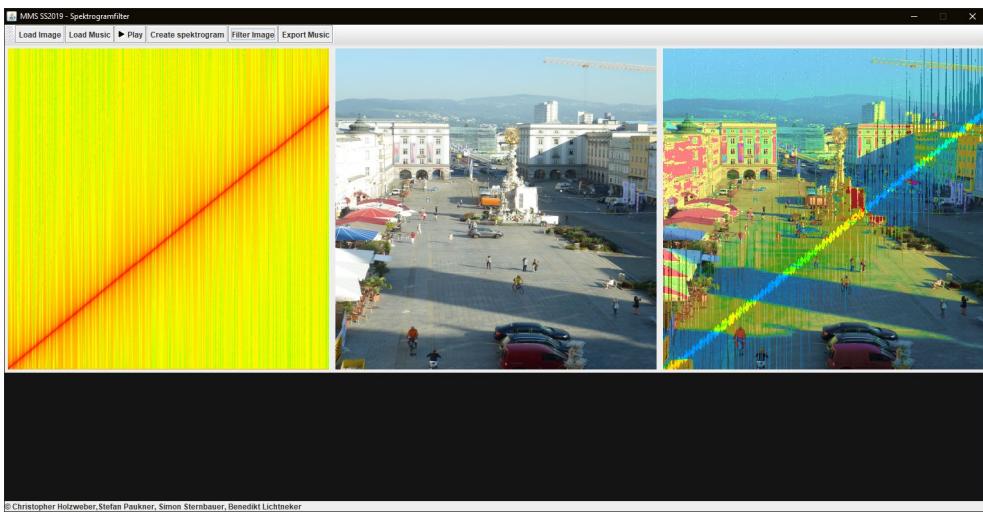


Figure 6: Spektrogramm mit sehr vielen hohen Amplituden

1.3.7 SpektogramImage.java

Diese Klasse kapselt die Bilderstellung des linken Spektogramm. Sie erhält vom Aufrufer die Daten des PCM Signals aus der WAV Datei. In der Klasse FFT [10] werden diese dann transformiert. Die Abmessungen des transformierten Ergebnisses und des darzustellenden Bildes passen erwartungsgemäß nicht zusammen, somit ist eine Stauchung der Ergebnisse notwendig welche auch hier gemacht wird. Die FFT wird mit einer Windowsize von 2048 eine Overlap-factor 8 und einem Zeropadding von 2 durchgeführt. siehe hierzu Abschnitt 2.1.1. Mit diesen Einstellungen sind bei einem Audiofile von 10 Sekunden mit der Abtastrate von 44,1 kHz ohne Overlap und Zeropadding 215 Transformationen notwendig. Mit dem eingestellten Overlap und gleichbleibender Windowsize erhöht sich die Zahl auf 1712 Transformationen. Mit diesen Werten erhält man eine gute Frequenzauflösung von 1024 Schritten und eine gute Zeitauflösung von welche bei der Stauchung einen einfachen Algorithmus zulässt. Es werden von der Klasse 2 Funktionen angeboten die 2 verschiedene Bilder zurückgibt. Eine wird für die Darstellung am Bildschirm verwendet, hierbei wird die Amplitude als H wert im HSV Farbraum eingesetzt und anschließend in den RGB Farbraum gewechselt. Die 2. Funktion bildet die Amplitude als

Rot wert im RGB Farbraum ab. Diese Funktion wird weiters für die Filterung verwendet.

2 Spektrogram

2.1 Spektrogram mittels FFT

In einem Spektrogramm werden Spektren (bei uns Frequenzspektren der DFT/FFT) über der Zeit dargestellt, damit man deren Verläufe erkennen kann. Es wird für einen bestimmten Zeitabschnitt eine DFT gemacht und dann an diesem aufgetragen.

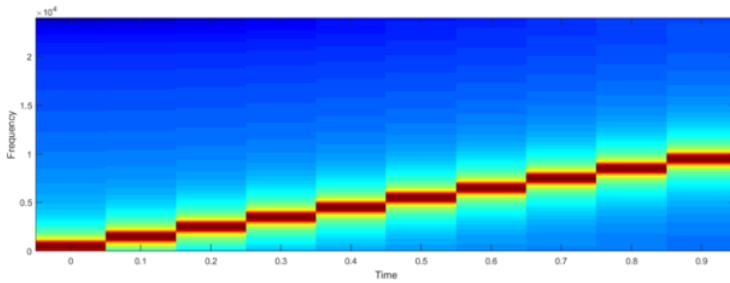


Figure 7: Spektrogram ohne Überlappung und Zeropadding

2.1.1 Auflösungsproblematik

Man möchte natürlich eine gute Zeit und Frequenzauflösung bei einem Spektrogramm. Das ist aber nicht ohne weiteres möglich, da der DFT folgende Formel zugrunde liegt:

$$\Delta f = \frac{1}{T_{Fenster}} = \frac{f_{Samples}}{N_{Sampleanzahl}}$$

Vergrößert man das Messfenster (also die Messdauer), so wird zwar die Frequenzauflösung besser, die Zeitauflösung aber schlechter (da die Größe des Messfensters die Zeitauflösung festlegt). Das gilt natürlich auch umgekehrt: verkleinert man das Messfenster um eine bessere Zeitauflösung zu erhalten, so sinkt die Frequenzauflösung der DFT.

Es gibt zwei Wege wie man, trotz der beschriebenen Problematik ohne Verluste die Auflösung von Frequenz und Zeit verbessern kann:

- Überlappen (Zeitauflösungsverbesserung):

Es werden zur selben Zeit mehrere DFT's versetzt gemacht. Immer die fertig ist, wird dann ausgegeben. Da man nun öfter mit einer DFT fertig wird, werden die Balken der einzelnen DFT's viel dünner -> Verbesserung der Zeitauflösung: Schaut man sich Daten in Echtzeit an, so ist man immer um die Dauer einer ganzen DFT später (das ist aber auch ohne Überlappen so). Je höher die Überlappung, desto besser die Zeitauflösung. Nachteil hier ist jedoch der steigende Rechenaufwand, da für ein Zeitstück dann mehrere DFT/FFT's nötig sind!

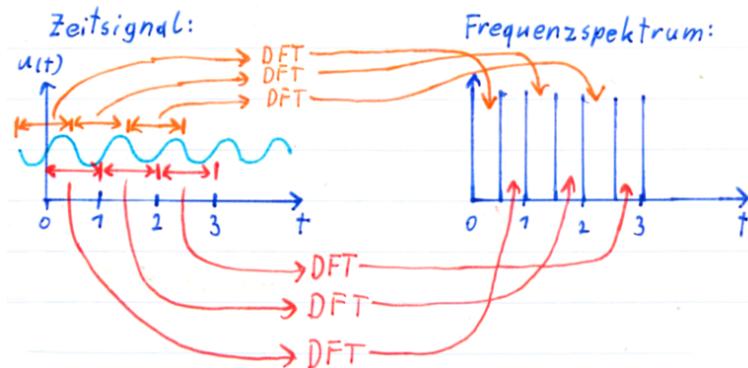


Figure 8: 50% Überlappung bei FFT

- ZeroPadding (Frequenzauflösungsverbesserung):

Bei Zero-Padding wird versucht, das Messsignal "virtuell/künstlich" zu vergrößern, dass bedeutet, dass von einem beispielsweise 10 Sekunden Signal, 1 Sekunde als Messsignal verwendet wird. Die Fenstergröße ist hierbei jedoch nicht 1 Sekunde, sondern 10 Sekunden, da die restlichen 9 Sekunden eine konstante Amplitude von 0 gemessen wird. Der Zeitbereich wird also mit 0en ausgefüllt. Durch einbringen eines Faktor N_{ZP} , werden die Messpunkte vervielfacht. Diese sind alle 0 und haben somit keine Auswirkung bei der DFT, außer dass eben TFenster größer wird und somit df kleiner. Somit ergibt sich eine bessere Frequenzauflösung.

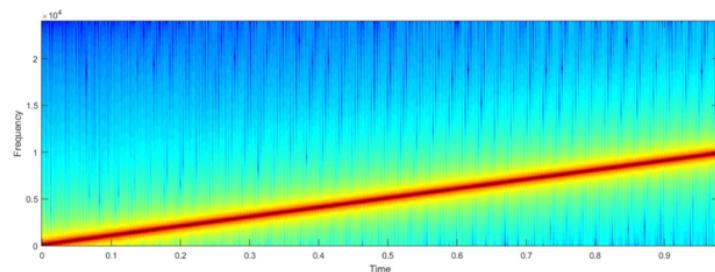


Figure 9: Spektrogramm mit Überlappung 90% und 8-fach Zeropadding

3 Projektlogo

3.1 Erstellung des Icon

Das Icon wird mit Hilfe unserer eigenen Applikation erstellt. Das Bild und eine entsprechend repräsentative Musik wird geladen. Ziel ist es einen vorher-nacher Effekt zu zeigen. Anschließend werden beide Bilder mithilfe des Programms Gimp2 zusammengefügt. Mittels des Werkzeuges „Pfade“ wird die Diagonale erzeugt um den Effekt herzustellen. In Figure 10 sind die zwei Ebenen des Bildes ersichtlich.

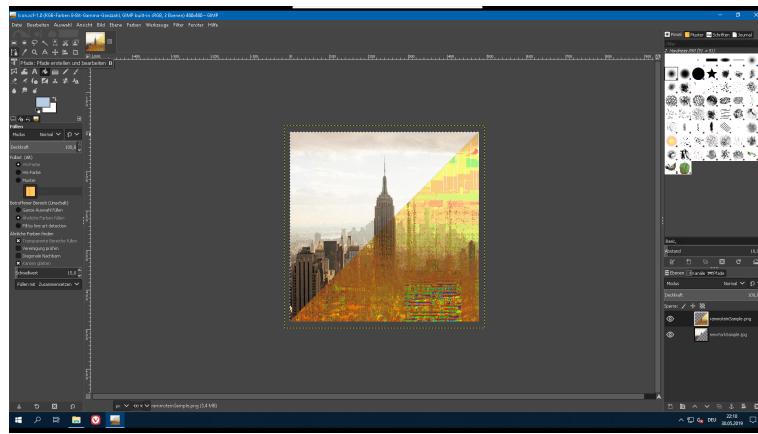


Figure 10: Logoerstellung

Das fertige Bild wird anschließend als .png exportiert und in die Applikation mit Hilfe des folgenden Code eingefügt:

```
 ImageIcon img = new ImageIcon("icon.png");
new StartingPanel(img).setIconImage(img.getImage());
```

Das nun gesetzte Logo wird auch in der TaskBar angezeigt.



Figure 11: Icon in TaskBar

3.2 Erstellung des Loadingscreens

Dieser wurde mit Hilfe von Powerpoint erstellt um einen animierten Übergang zwischen den zuvor erstellten Bildern zu schaffen. Es wurde anschließend als .mp4 Video exportiert. Um zu vermeiden, dass ein Videoplayer angezeigt wird, wurde es zu einem GIF um konvertiert.

3.3 Farbschema

Die Applikation verfolgt das neue und moderne Schema „Dark mode“. Alle Buttons erhielten das neue Design. Insgesamt gibt es vier verschiedene Farben:

- Die Button-Hintegrundfarbe

Hierbei handelt es sich um die ganz normale Farbe eines Buttons welche er nach dem Start besitzt. (#5e5e5e)

- Die Button-Hoverfarbe

Die Maus befindet sich über einem Button (#5e9bff)

- Die Button-Klickedfarbe

Die linke Maustaste wird gepresst

- Hintergrundfarbe der gesamten Applikation (#939393)



Figure 12: Buttons

Zu beachten gilt es noch, dass der Play Button ein icon beinhaltet. Hierfür wurden drei Bilder angefertigt, die mit den jeweiligen Hintergrundfarben exportiert wurden:

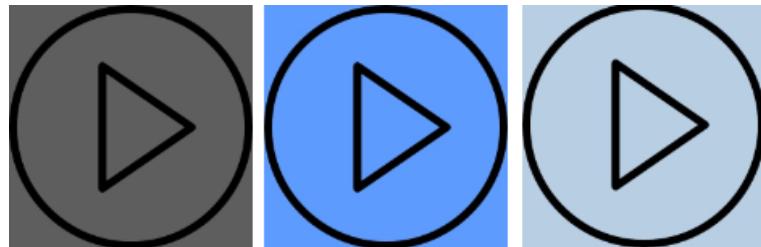


Figure 13: Play Buttons

4 Verzeichnisse

List of Figures

1	Projektskizze	3
2	Eine Übersicht der zur Verfügung stehenden Buttons	4
3	Starting Panel Auszüge	6
4	HSB Farbraum	8
5	Spektrogramm mit sehr vielen geringen Amplituden	9
6	Spektrogramm mit sehr vielen hohen Amplituden	9
7	Spektrogramm ohne Überlappung und Zeropadding	10
8	50% Überlappung bei FFT	11
9	Spektrogramm mit Überlappung 90% und 8-fach Zeropadding	11
10	Logoerstellung	12
11	Icon in TaskBar	12
12	Buttons	13
13	Play Buttons	13

List of Tables

5 Referenzen

Spektrogramm aus WAV File erstellen in Java[1]

- FFT-Erstellung in Java[2]
- Aufbau eines WAV-Files[3] [4]
- Farbsättigung [8]
- HSB Farbraum[7]
- Audiofile Format[5]
- WAV File abspielen[6]
- MP3 und WAV-Converter[9]

References

- [1] <https://stackoverflow.com/questions/39295589/creating-spectrogram-from-wav-using-fft-in-java>
- [2] <https://stackoverflow.com/questions/3287518/reliable-and-fast-fft-in-java>
- [3] https://de.wikipedia.org/wiki/RIFF_WAVE
- [4] <https://joenord.com/audio-wav-file-format>

- [5] [https://stackoverflow.com/questions/5766445/
how-to-split-a-wav-file-into-channels-in-java](https://stackoverflow.com/questions/5766445/how-to-split-a-wav-file-into-channels-in-java)
- [6] [https://stackoverflow.com/questions/2416935/
how-to-play-wav-files-with-java](https://stackoverflow.com/questions/2416935/how-to-play-wav-files-with-java)
- [7] <https://de.wikipedia.org/wiki/HSV-Farbraum>
- [8] <https://de.wikipedia.org/wiki/Farbs%C3%A4ttigung>
- [9] <http://www.sauronsoftware.it/projects/jave/manual.php>
- [10] [http://www.ee.columbia.edu/~ronw/code/MEAPsoft/src/com/
meapsoft/FFT.java](http://www.ee.columbia.edu/~ronw/code/MEAPsoft/src/com/meapsoft/FFT.java)