Visualisierung von Spektrogrammen

12.03.2010

Dominik Neumann, Felix Lugauer

Lehrstuhl für Informatik 5 (Mustererkennung)

Universität Erlangen-Nürnberg

Overview

Overview

Aufgabenstellung

Paket 3: Visualisierungskomponenten für Funktionen $f(t) \in \mathbb{R}^2$

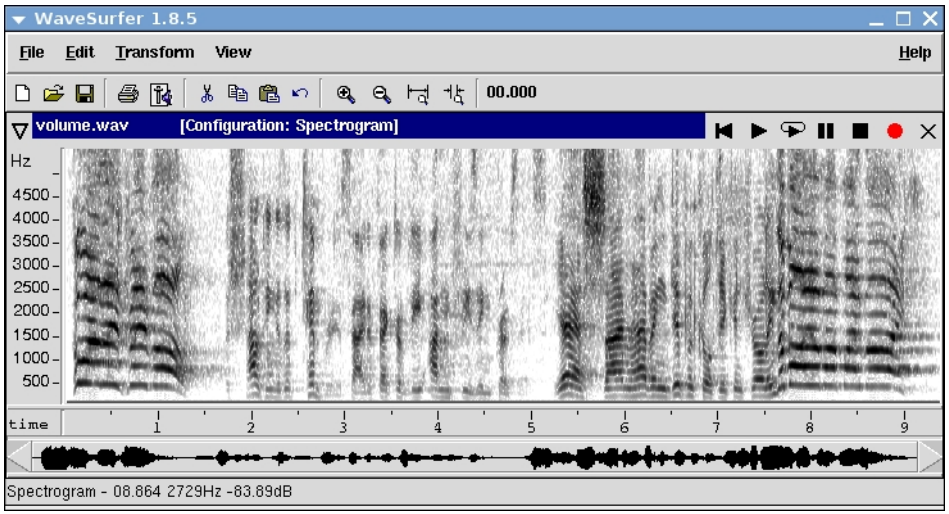
- Komponente zur Darstellung des Spektrogramms von Audiostreams
- Komponente zur Darstellung des Spektrogramms von Audiodateien

Aufgabenstellung

Paket 3: Visualisierungskomponenten für Funktionen $f(t) \in \mathbb{R}^2$

- Komponente zur Darstellung des Spektrogramms von Audiostreams
- Komponente zur Darstellung des Spektrogramms von Audiodateien

Und so könnte es aussehen:

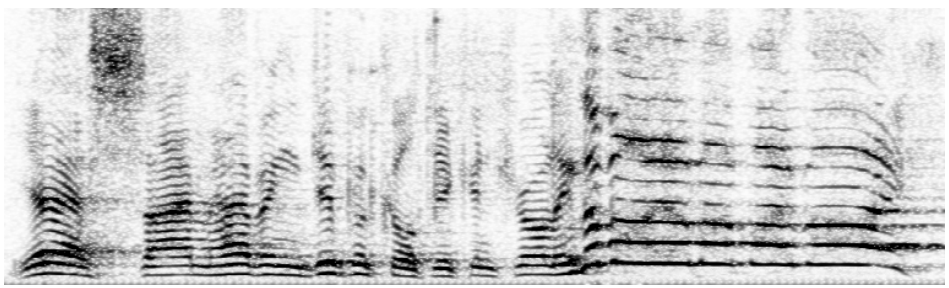


Spektrogramme

Definition

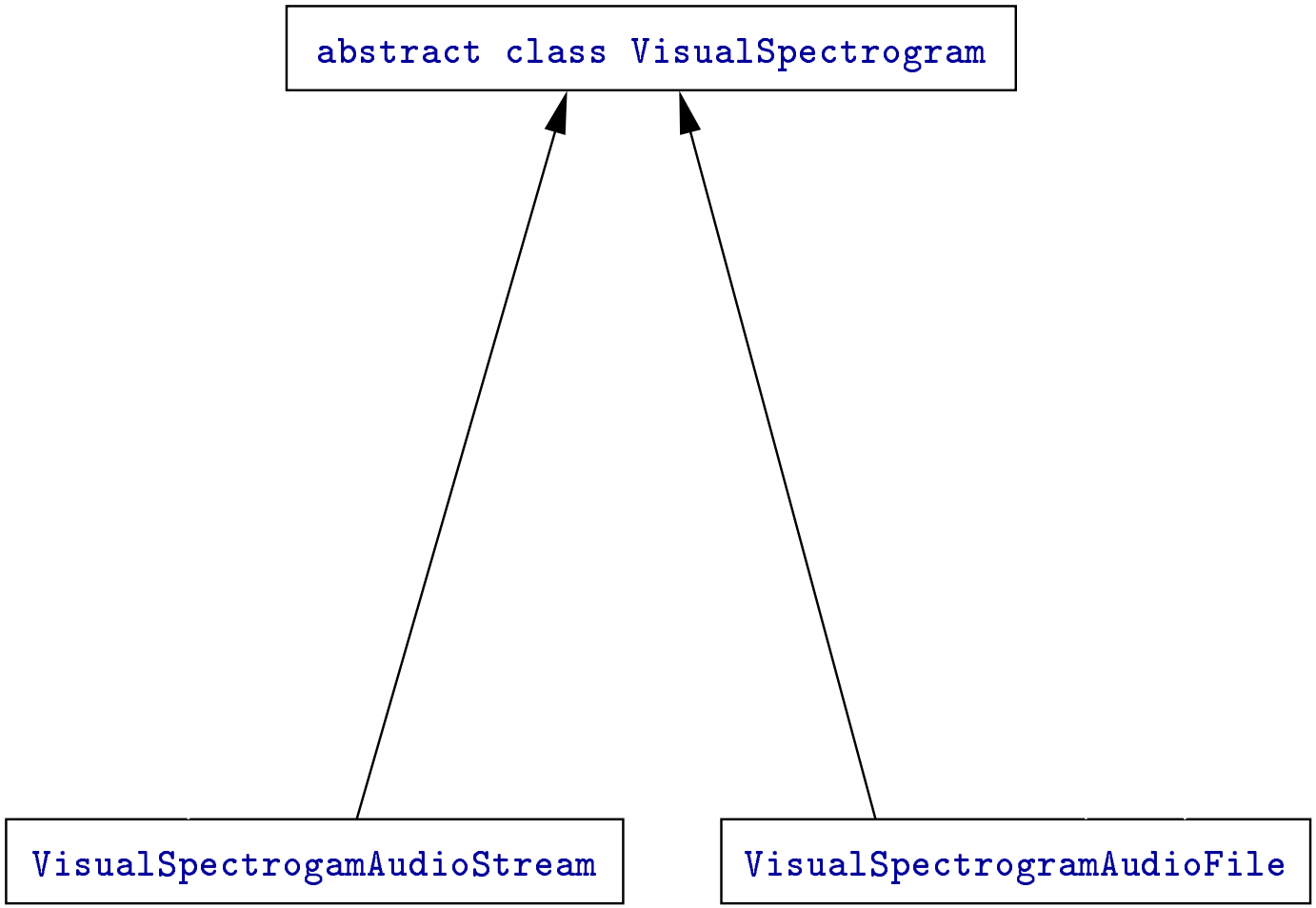
Ein Spektrogramm ist die Darstellung des zeitlichen Verlaufs des Spektrums eines Signals

- x-Achse: Zeit
- y-Achse: Frequenz
- Intensität der Bildpunkte: Amplitude der jeweiligen Frequenz im Spektrum

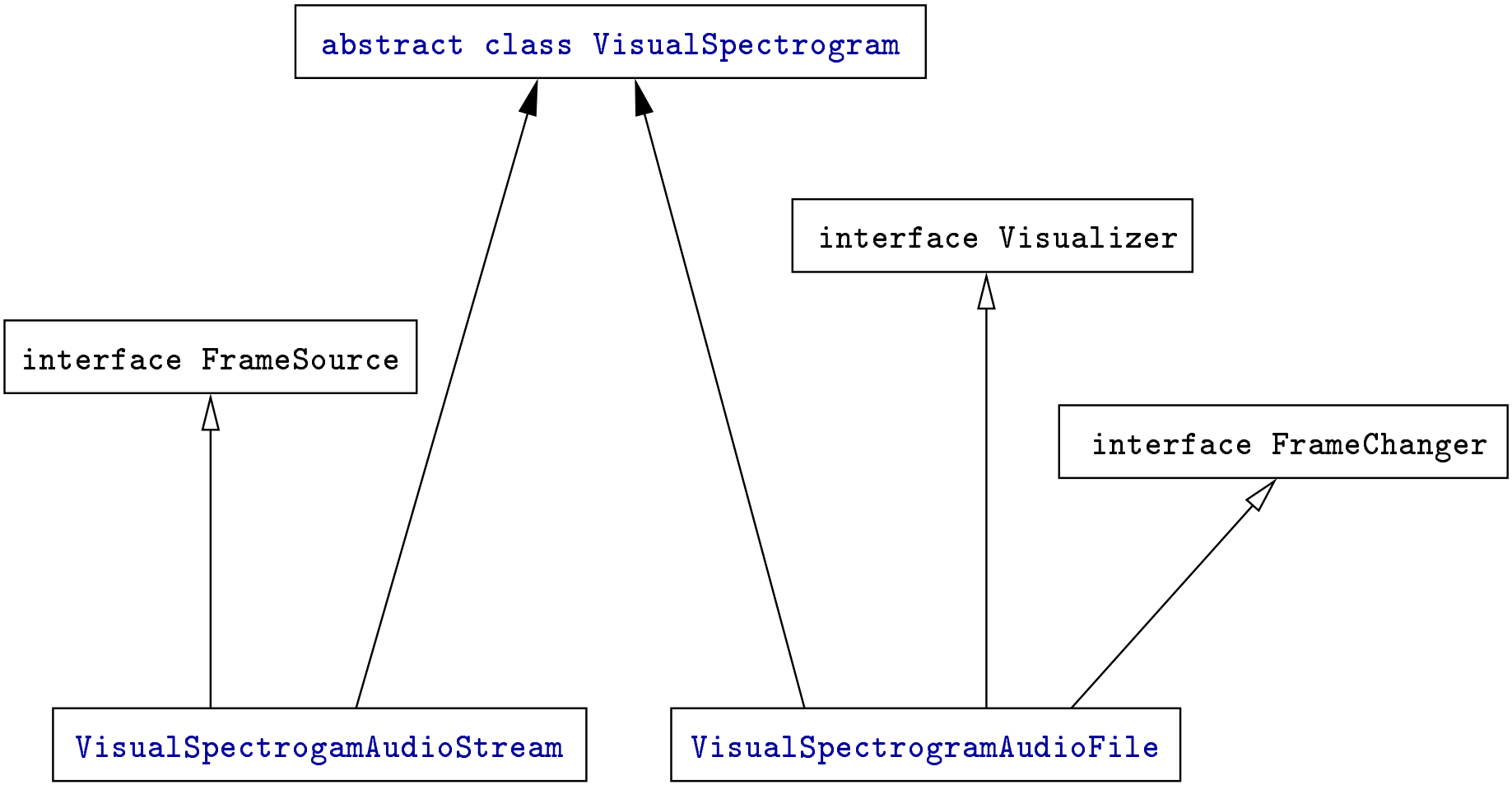


Overview

Klassenhierarchie



Klassenhierarchie



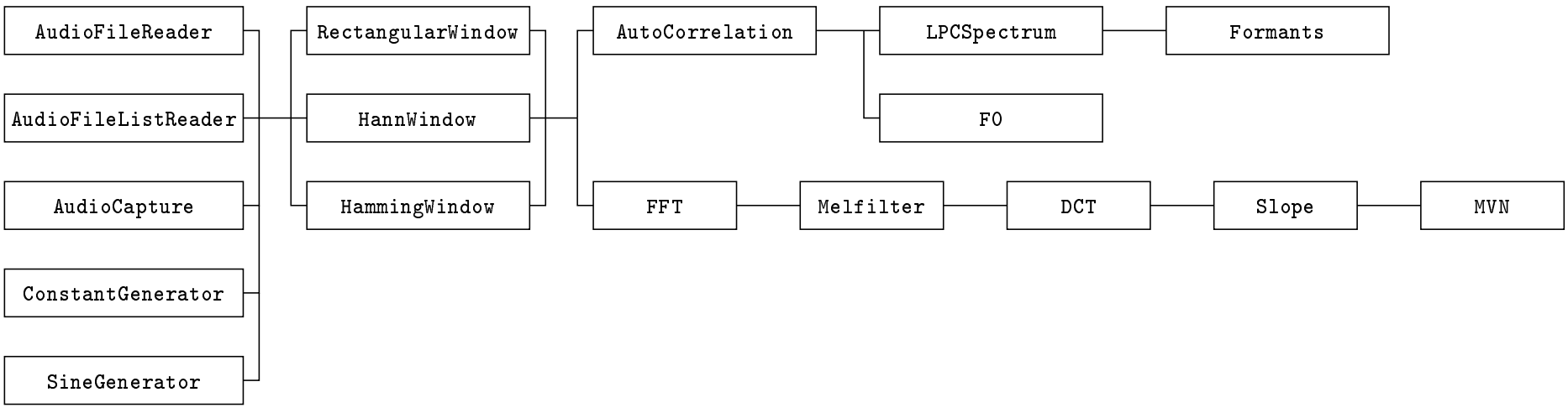
Klassenhierarchie

Relevante Klassen und Interfaces

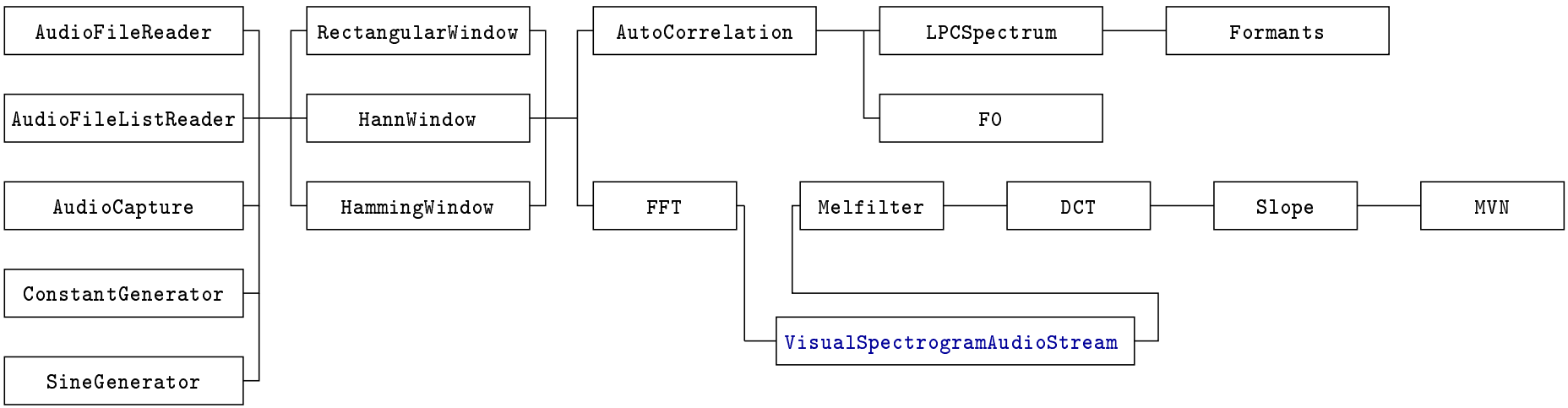
`package visual`

- `interface Visualizer`
  Synchronisation mit anderen Komponenten
- `interface FrameChanger`
  Kommunikation mit Komponenten auf Frame-Ebene
- `abstract class VisualSpectrogram`
- `class VisualSpectrogramAudioStream`
  Visualisierung von Spektrogrammen von Audio-Streams
- `class VisualSpectrogramAudioFile`
  Visualisierung von Spektrogrammen von Audio-Dateien

Eingliederung in das vorhandene Stream-Konzept



Eingliederung in das vorhandene Stream-Konzept



Visualisierer für Spektrogramme von Audio-Streams lassen sich nahtlos in das bereits bestehende Stream-Konzept eingliedern

Problematik bei Audiodateien

Eingliederung eines Audiodatei-Visualisierers in das vorhandenes Stream-Konzept problematisch

Anforderungen:

- Möglichkeit zur Anzeige beliebiger Teile der Audiodatei (von - bis)
- Nutzung unterschiedlicher Fensterfunktionen
- Unabhängigkeit von anderen Visualisierungskomponenten
- ...

Problematik bei Audiodateien

Eingliederung eines Audiodatei-Visualisierers in das vorhandenes Stream-Konzept problematisch

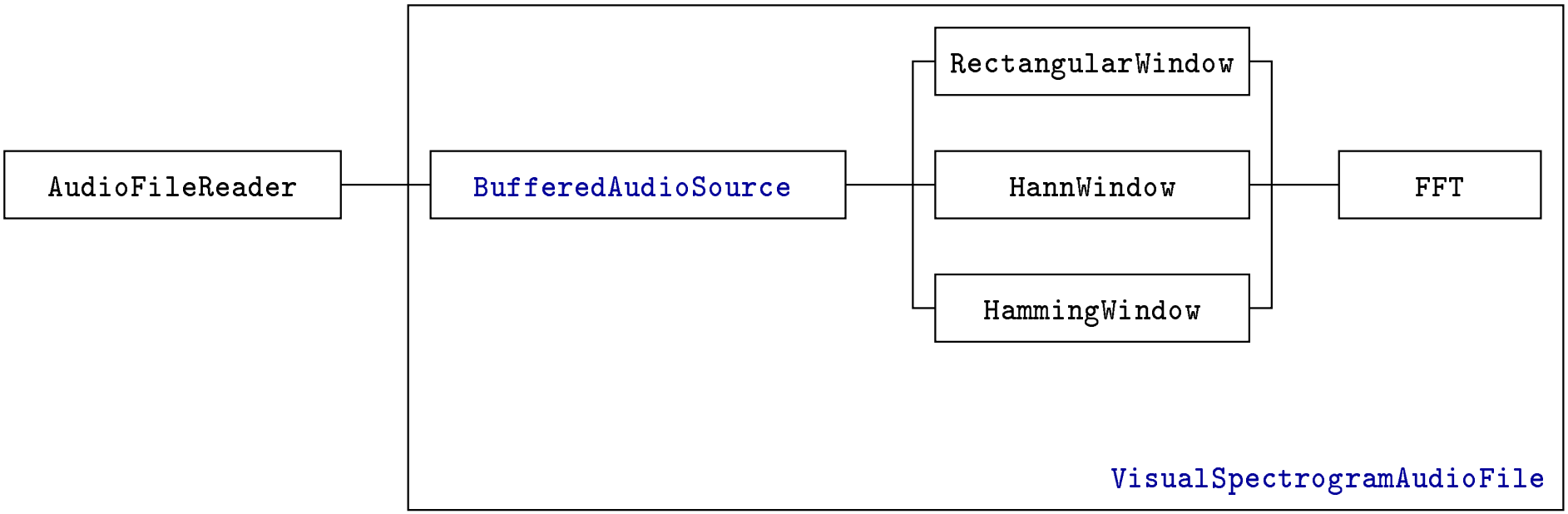
Anforderungen:

- Möglichkeit zur Anzeige beliebiger Teile der Audiodatei (von - bis)
- Nutzung unterschiedlicher Fensterfunktionen
- Unabhängigkeit von anderen Visualisierungskomponenten
- ...

... und das alles zur Laufzeit

Verwendung des vorhandenen Stream-Konzepts

Realisierung: Die Komponente selbst übernimmt einen Teil des Stream-Konzepts



Funktionsumfang

Features der Visualisierungskomponenten

- Visualisierung von Funktionen $f(t) \in \mathbb{R}^2$
- Logarithmische Skala
- Regulierung von Helligkeit und Kontrast
- Möglichkeit zur Kolorierung (Hitzeskala)
- Farbige Bereichsmarkierungen
- ...

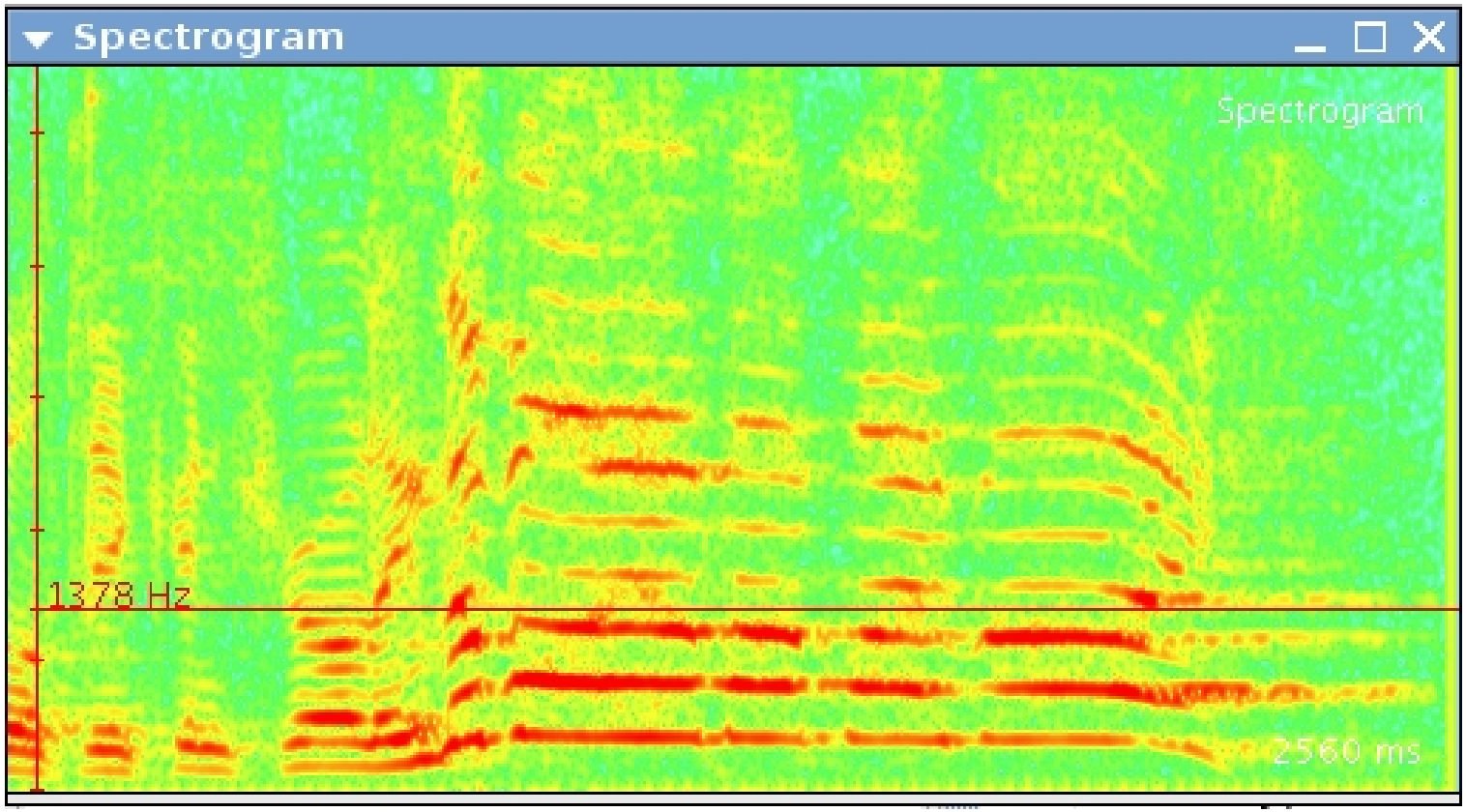
Funktionsumfang (Forts.)

Weitere Features

- Anpassung der Fensterfunktion zur Laufzeit
- Export des Spektrogramms als Bilddatei
- Zusatzinformation über das Spektrogramm darstellen
- Optimierung: Überflüssige Berechnungen vermeiden
- Rechenintensives von eigenen Threads ausführen lassen
- Ausführliche Code-Dokumentation mittels JavaDoc
- ...

Demoapplikation Audiostream

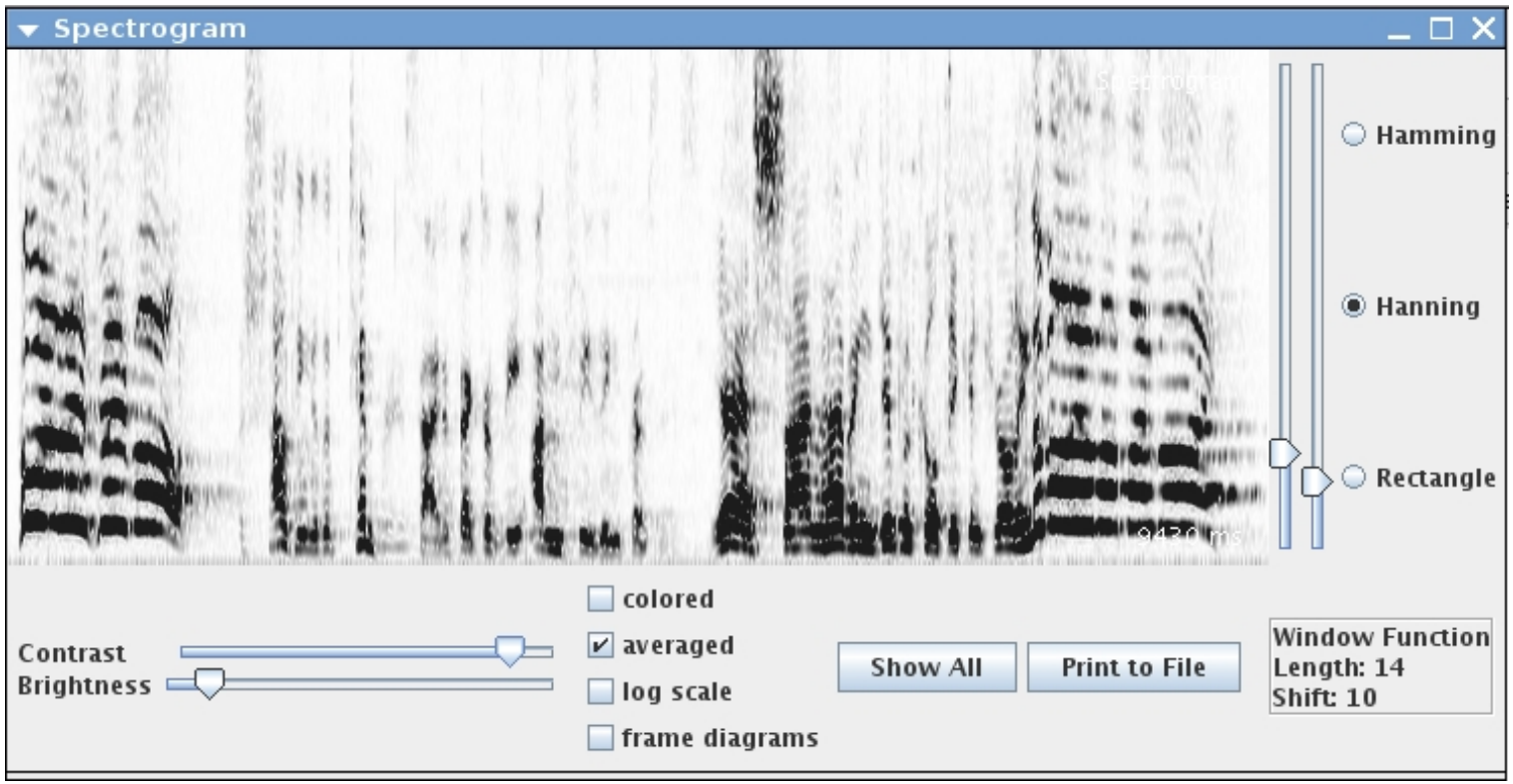
Implementierung einer GUI zur Demonstration des Funktionsumfangs



`visual.DemoSpectrogramStream`

Demoapplikation Audiodatei

Implementierung einer GUI zur Demonstration des Funktionsumfangs



`visual.DemoSpectrogramFile`

Overview

Einsparung von unnötigen Berechnungen

Ziel: Wiederverwendung von bereits berechneten Spektrogrammdaten

Realisierung:

- zunächst werden alle Daten zur Anzeige des aktuellen Spektrogramms in einem Puffer gespeichert
- bei Änderung des anzuzeigenden Bereichs:
  - falls neuer Bereich innerhalb des alten Bereichs liegt: benutze Werte aus Puffer, keine Neuberechnung
  - sonst: Berechnung der neuen Werte nötig
- bei Änderung der Fensterfunktion: Neuberechnung zwingend erforderlich

Optimierte Darstellung der Spektrogrammdaten

Ziel: Berücksichtigung aller Daten durch Mittelwertbildung

- Einpassung der Spektrogrammdaten aus dem Puffer auf die gewünschte Bildgröße
- Entscheidendes Verhältnis $ratio = \frac{bufferLength}{imageWidth}$
  - $ratio < 1.0$ mehrere Pixel bekommen den gleichen Wert
  - $ratio > 1.0$ mehrere Pufferwerte kommen auf einen Pixel
- Optisch *angenehmeres* Aussehen, Glättung von Rauschen

Kommunikation mit Komponenten auf Frame-Ebene

Synchronisation der Anzeige mit Spektrum, MFCC-Diagramm, ...

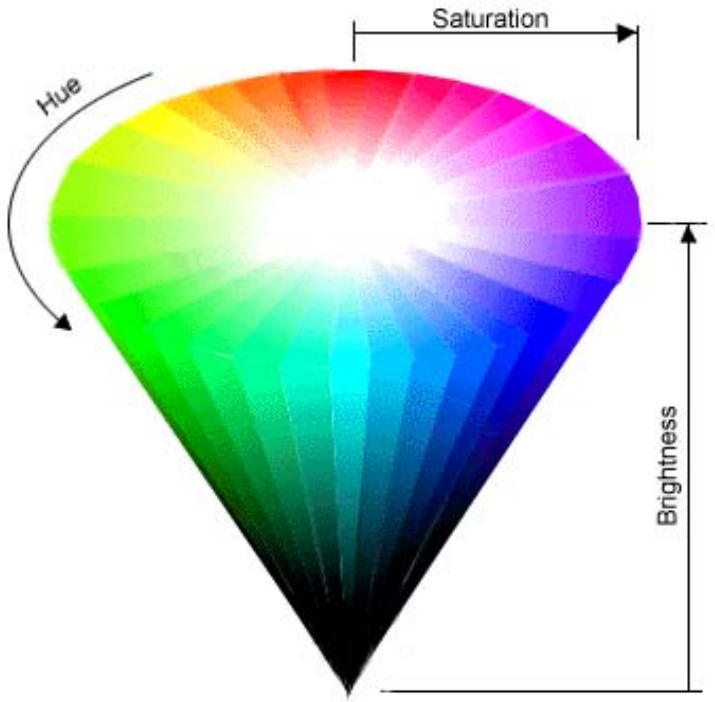
- GUI-Entwickler meldet die Komponenten der Frame-Ebene beim Spektrogramm an
- Realisierung durch Liste: Speichern der Referenz auf zu synchronisierende Komponenten
- Ereignis lässt die Komponenten bestimmtes Frame analysieren
  `FrameListener.show(int startSample)`

- Interfaces garantieren einheitliche Kommunikation:
  - Spektrogramm Visualisierer: `visual.FrameChanger`
  - Frame-Ebene Visualisierer: `visual.FrameListener`

Farbiges Spektrogramm

Hitzeskala über HSB-Farbmodell

- HSB-Farbmodell
  - Farbton
  - Sättigung
  - Helligkeit
- Grauskala: keine Sättigung
  - Weiß: `brightness = 1.f`
  - Grau: `0.f < brightness < 1.f`
  - Schwarz: `brightness = 0.f`

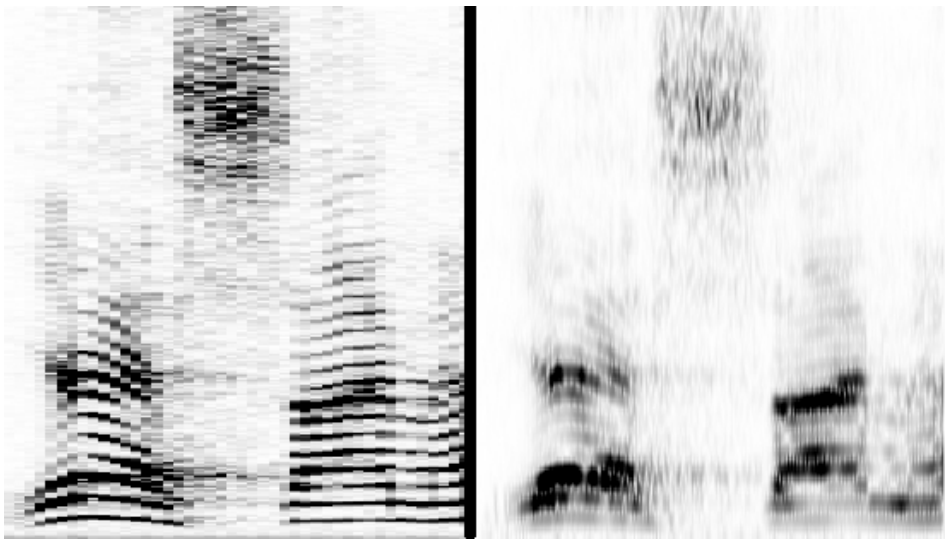


- `static int java.awt.Color.HSBtoRGB(float hue, float saturation, float brightness)`

Möglichkeit zur Änderung der Fensterfunktion

Ziel: Veranschaulichung von Auswirkungen der verwendeten Fensterfunktion auf das Spektrogramm

- Fenstereigenschaften
  - Typ: `Hamming` | `Hanning` | `Rectangle`
  - Größe: Anzahl der Abtastwerte pro Fenster
  - Shift: Sprungweite
- Anpassung aller genannten Eigenschaften zur Laufzeit möglich



Implementierung des internen Datenstroms

1. Auslesen des gepufferten Audiosignals
2. Fensterung liefert Werte für FFT-Berechnung
3. FFT liefert Frames, die im internen Puffer gespeichert werden

Implementierung

```
public void computeAndRepaintSpectrogram() {

    BufferedAudioSourceReader basr = bas.getReader(viewFromSample,viewNSamples);
    window = Window.create(basr, windowType+","+windowLength+","+windowShift);
    FrameSource spec = new FFT(window);

    samplesPerShift = ((basr.getSampleRate() * windowShift) / 1000);
    int size = (viewNSamples / samplesPerShift);
    spectrogramBuffer = new double[size][spec.getFrameSize()];

    for(int i = 0; i < size && spec.read(spectrogramBuffer[i]); i++) { }
    repaintAndUpdateImage();
}
```

Flexible Darstellungsmöglichkeiten

Anpassung des Spektrogramms an die Komponentengröße

- Funktion zur Größenänderung des internen `BufferedImages`
- Listener überwacht Änderung an der Komponente
- Interface `java.awt.event.ComponentListener`

Implementierung

```
public void componentResized(ComponentEvent e) {
     setImageDimension(getWidth(), getHeight());
     repaintAndUpdateImage();
}
```

Normierung und Kontrastspreizung der Darstellung

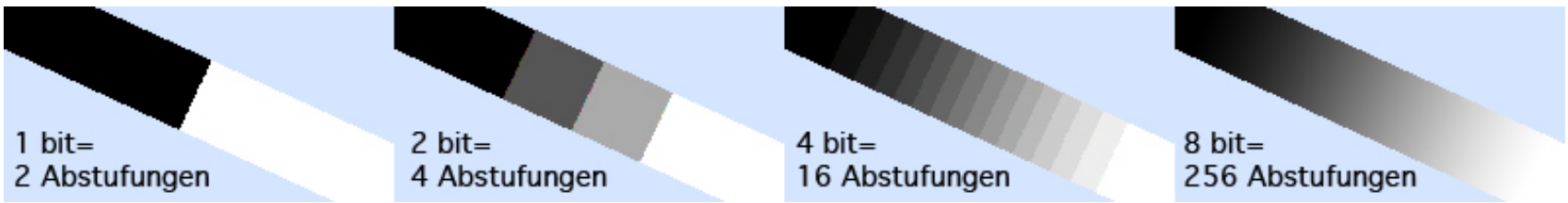
Ziel: Verbesserung der Sichtbarkeit des Spektrogramms

- Frequenzbänder die stark ausgeprägt sind sollen umso dunkler im Spektrogramm sein
- Abbildung von Werten aus der FFT auf Grauskala
- Formel zur Anpassung der Abstufungen im Wertebereich:

$$pixelWert = \frac{FFTWert}{Kontrast}, \quad Kontrast \geq FFTWert$$

Umsetzung der Kontrastanpassung

- Kontrastdifferenzierung erhöht bzw. verringert ’Lesbarkeit’ des Spektrogramms
- Realisierung durch Einschränkung der Werteskala:
  - kleiner Wertebereich: deutlichere Abstufungen zu benachbarten Graustufen
  - großer Wertebereich: kaum Unterschiede zu benachbarten Graustufen



Entkopplung von Anzeige und Berechnung

Ziel: Einsparung von Rechenzeit bei Neuzeichnung der Komponente

- Zusatzinformationen bei jeder Mausbewegung in Komponente
- Information nicht ins Bild, sondern auf Komponente zeichnen
- Darstellung von Informationen ohne Neuberechnung des Bildes
- Transformation der Pufferdaten auf Bilddaten nur, wenn sich am Spektrogramm etwas ändern soll: Kontrast, Farbe,...

- Entkopplung der Zeichenfunktionen:
  - `paint()`
  - `paintOverlay()`
  - `createSpectrogramImage`

Overview

Was würden wir jetzt anders machen?

Think before you ink!

- Mehr Vorüberlegungen zu Projekt und Struktur
- Code-Leichen und ’hingehackte’ Codestücke früher über Board werfen ;-)
- Kommentare nicht erst am Ende

Ausblick

Wenn wir noch mehr Zeit hätten:

- Weitere Verbesserungen der Speicherverwaltung
- Einzelne Features verbessern
  - Vielfältigere Möglichkeiten zur Exportierung des Spektrogramms
  - Weitere Maßnahmen zur Synchronisation mit anderen Komponenten
- Berechnungen parallelisieren

Ausblick

Wenn wir noch mehr Zeit hätten:

- Weitere Verbesserungen der Speicherverwaltung
- Einzelne Features verbessern
  - Vielfältigere Möglichkeiten zur Exportierung des Spektrogramms
  - Weitere Maßnahmen zur Synchronisation mit anderen Komponenten
- Berechnungen parallelisieren
- ...eine noch schönere Demo-GUI schreiben :-)

Overview

Ende

Vielen Dank für die Aufmerksamkeit!