Interactive Music Playground - Dokumentation Audioverarbeitung

Dirk Löwenstrom (2203730), Max Wiechmann (2171455)

1. Kontext

Der Interactive Music Playground erkennt mittels openCV Bausteine verschiedener Farben und geometrischer Figuren, die im Code von der Klasse `MusicChip` repräsentiert werden. Jeden Frame (jedes Mal, wenn die Methode `process()` im `ImageProcessor` aufgerufen wird) wird für jeden `MusicChip` die Methode `handleAudio()` aufgerufen, die je nach aktuellem Zustand mit dem Signal-Slot-Mechanismus von Qt Nachrichten an das Objekt der Klasse `SoundControl` sendet. Im Folgenden wird für die Slots `play()`, `stop()` und `applyEffects()` von `SoundControl` der Ausführungsablauf beschrieben, wobei besonderes Augenmerk auf der Anwendung der Effekte gelegt wird.

2. Klassenstuktur

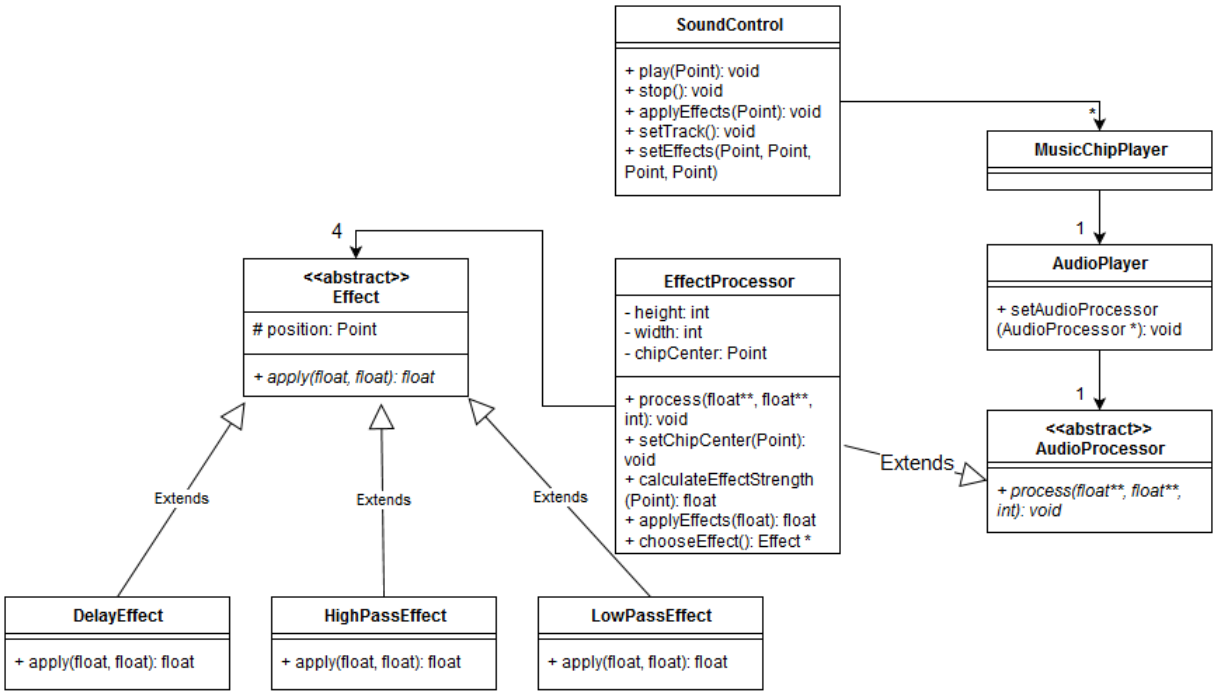

Abbildung 1: UML-Diagramm der wichtigsten Klassen für die Audioerzeugung und Effektanwendung (Unvollständig, erstellt mit draw.io)

Das in Abbildung 1 dargestellte UML-Diagramm zeigt die wichtigsten Klassen und jeweils die wichtigsten Felder und Methoden für die Audioverarbeitung (Für ein vollständigeres UML-Diagramm siehe Github-Repository). In der Klasse `SoundControl` sind fünf Slots zu sehen, die, wie oben beschrieben, aus der Klasse `MusicChip`, aber auch aus der Klasse `ImageProcessor` emittiert werden. `SoundControl` hat einen Vektor, der `MusicChipPlayer` hält. Diese Klasse dient als Wrapper für den `AudioPlayer`, um die Handhabung zu vereinfachen. Zu jedem `MusicChip` gibt es einen korrespondierenden `MusicChipPlayer`. Jeder `AudioPlayer` hat (neben der nicht im UML Diagramm eingezeichneten `AudioSource`) einen `AudioProcessor`, der den Audioinput erhält, verarbeitet, und als Audiooutput zurückgibt. Die tatsächliche Implementierung des Ganzen findet in der Klasse `EffectProcessor` statt, die von `AudioProcessor` erbt und die Methode `process()` überschreibt. Der `EffectProcessor` enthält die wichtige Logik, um die

Effektanwendung auf den Audioinput zu managen. Die tatsächlichen Effekte sind über eine abstrakte Klasse `Effect` mit der Methode `apply()` und entsprechenden Unterklassen `DelayEffect`, `HighPassEffect` und `LowPassEffect` implementiert. Hier wird Polymorphie voll ausgenutzt.

3. Fall 1: Ein `MusicChip` wird erkannt und `play()` wird aufgerufen

```
void SoundControl::play(Point position){
    MusicChip* sender = qobject_cast<MusicChip*>(QObject::sender());
    MusicChipPlayer* player = getPlayer(sender->objectName());
    player->getEffectProcessor()->active();
    player->getEffectProcessor()->setChipCenter(position);
}
```

Die Methode `play()` wird immer dann ausgeführt, wenn ein `MusicChip` das Signal `on()` emittiert. Dabei wird als Parameter der Mittelpunkt des `MusicChip` übergeben. Nun werden einige von Qt angebotene Funktionen genutzt. Über die Methode `QObject::sender()` lässt sich herausfinden, welches Objekt das Signal gesendet hat. Dies ist notwendig, um über die Methode `getPlayer()` den entsprechenden `MusicChipPlayer` herauszufinden. Eine weitere Eigenschaft von Qt kommt zum Tragen: Sowohl `MusicChip` als auch `MusicChipPlayer` erben von `QObject` und haben somit einen `objectName`. Bei der Erzeugung bekommt der `MusicChipPlayer` den `objectName` passend zum `MusicChip`, sodass durch Vergleich der Objekt-Namen in `getPlayer()` bestimmt wird, welcher `AudioPlayer` Sound abspielen soll. Das Handling der Lautstärke findet komplett im `EffectProcessor` statt. Um Synchronität zu der einzelnen Audiotracks und ihrer BPM (beats per minute) gewährleisten, werden alle `AudioPlayer` zu Beginn gleichzeitig gestartet. Allerdings ist der `gain` standardmäßig auf null gestellt, sodass es nicht zur Klangerzeugung kommt. Um einen Audiotrack tatsächlich abzuspielen, wird durch `active()` im `EffectProcessor` der Zustand (Repräsentiert durch ein Enum) auf `FADEIN` gesetzt. Die Methode `setState()` kümmert sich darum, dass der `gain` im Status `FADEIN` ansteigt, im Status `ON` konstant auf eins bleibt, im Status `FADEOUT` abfällt und im Status `OFF` auf null gesetzt ist.

4. Fall 2: Ein `MusicChip` wird nicht länger erkannt und `stop()` wird aufgerufen

```
void SoundControl::stop(){
    MusicChip* sender = qobject_cast<MusicChip*>(QObject::sender());
    MusicChipPlayer* player = getPlayer(sender->objectName());
    player->getEffectProcessor()->off();
}
```

Entsprechend wird, wenn das Signal für die Methode `stop()` gesendet wird, durch die Methode `off()` von `EffectProcessor` der State auf `FADEOUT` gesetzt. Wird ein `MusicChip` nicht länger erkannt, klingt sein Audiotrack langsam aus.

5. Fall 3: Ein `MusicChip` wird bereits erkannt und seine Position ändert sich

```
void SoundControl::applyEffects(Point position){
    MusicChip* sender = qobject_cast<MusicChip*>(QObject::sender());
    MusicChipPlayer* player = getPlayer(sender->objectName());
    player->getEffectProcessor()->setChipCenter(position);
}
```

Wird ein `MusicChip` bereits erkannt, aber seine Position ändert sich, wird lediglich die entsprechende `chipCenter`-Variable im `EffectProcessor` neu gesetzt.

6. Die Effektberechnung

```
void EffectProcessor::process(float **input, float **output, int
numFrames){
    //[…]
    gain = min(gain * gainChange, 1.0f);
    for(int i = 0; i < numFrames; i++){
        output[0][i] = applyEffects(input[0][i]) * gain;

    }
    //[…]
}
```

Die Methode `process()` wird vom `AudioPlayer` in der Methode `readData()` permanent für jedes Input-Array aufgerufen und das Output-Array wird dann ausgegeben. Zunächst wird der aktuelle `gain` berechnet, danach wird für jedes Sample die Methode `applyEffects()` aufgerufen und das Ergebnis mit dem `gain` multipliziert im Output-Array gespeichert. Die Methode `applyEffects()` findet zunächst heraus, welcher der vier möglichen Effekte (ein Effekt pro Ecke des Kamera-Inputs) angewandt werden soll. Dazu berechnet sie, welchem Effekt der aktuelle `chipCenter` am nächsten ist. Dann wird die Effektstärke berechnet. Je näher sich der `MusicChip` zur Ecke des Effekts befindet, desto stärker soll der Effekt auf das zugehörige Audiosignal angewandt werden.

```
float EffectProcessor::calculateEffectStrength(Point effect){}
    Point p = Point(chipCenter.x, chipCenter.y * ratio);
    int dx = abs(effect.x-p.x);
    int dy = abs(effect.y-p.y);
    int max_distance = std::max(dx, dy);
    float scaled_distance = min((float)(max_distance)/
((float)(width)/2),1.0f);
    float strength = max(min(1.4*(1 - scaled_distance-0.2),1.0),0.0);
    return strength;
}
```

Die Methode `calculateEffectStength()` ist für die Berechnung der Effektstärke zuständig. Wie in Abbildung 2 zu sehen, ist die Fläche in vier gleichgroße Ebenen geteilt. Am Rand jeder

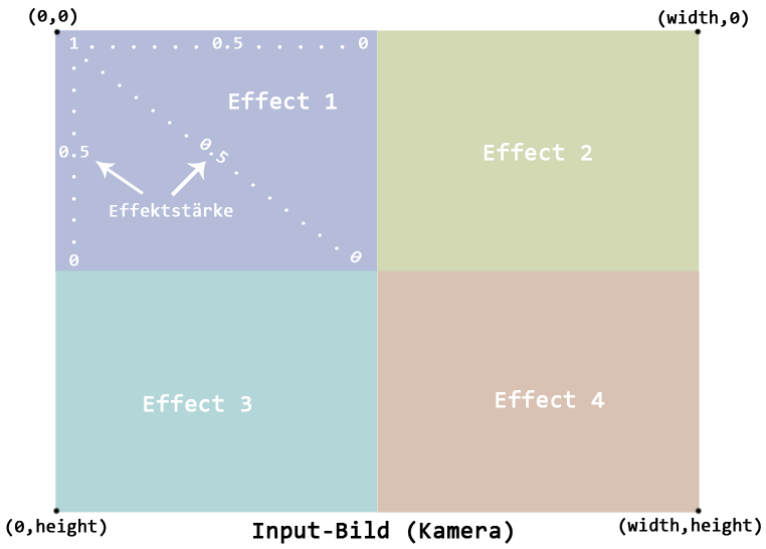

Abbildung 2: Aufteilung des Input-Bildes und Effektstärke

Teilfäche soll die Effektstärke null betragen, in der Ecke eins. Um die korrekte Stärke zu berechnen, muss zunächst das Koordinatensystem gestreckt werden, damit ein Effekt in y-Richtung bei der Hälfte ebenfalls null ist. Deswegen wird sowohl `chipCenter` als auch `Point effect` in y-Richtung um das Seitenverhältnis hochskaliert, sodass die Fläche nun praktisch quadratisch ist. Dann wird

sowohl in x-, als auch in y-Richtung die Entfernung vom Punkt zum Effektpunkt berechnet. Da es nur darauf ankommt, in welche Richtung der `MusicChip` weiter weg ist, wird das Maximum von beiden Werten genommen und das Ganze auf eine Zahl zwischen Null und Eins skaliert. Die Stärke berechnet sich dann aus einer linearen Funktion, die in Abbildung 3 als Graph dargestellt ist, wobei die noch invertiert wird, sodass eine Entfernung von Eins eine Stärke Null bedeutet.

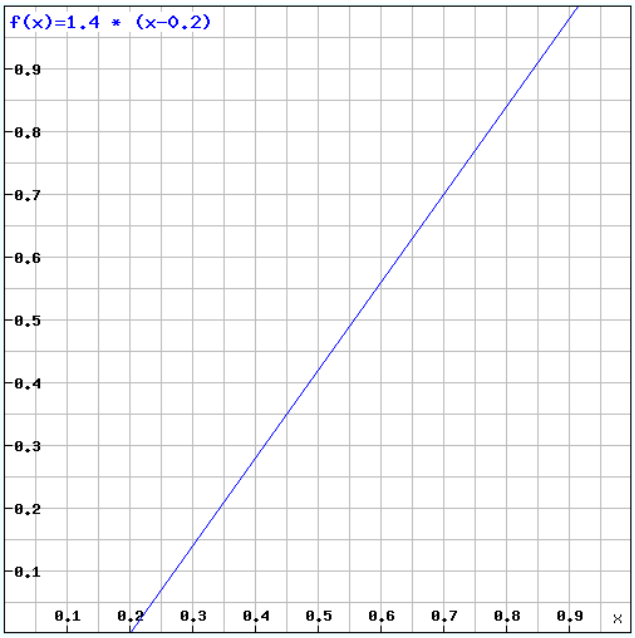

Abbildung 3: Graph der Funktion zur Effektstärken-Berechnung (Erstellt auf rechneronline.de/function-graphs/)

Die berechnete Effektstärke wird dann der Methode `apply()` des ausgewählten Effekts übergeben.

```
float HighPassEffect::apply(float input, float strength){
    lastOutput = 0.9f * (lastOutput + input - lastInput);
    float output = lastOutput * strength + input * (1-strength);
    lastInput = input;
    return output;
}
```

Beispielhaft für die Effekte ist hier die `apply()`-Methode des `HighPassEffect` dargestellt. Zunächst wird der Hochpassfilter mit voller Stärke berechnet und dann der tatsächliche Output durch Einrechnen der Stärke berechnet und zurückgegeben. Dieses Ergebnis wird nun in das Output-Array geschrieben und letztlich als Sound über die Lautsprecher abgespielt. Verändert sich die Position eines `MusicChip`, verändert sich entsprechend die `strength`, oder, wenn der `MusicChip` aus einem Effektbereich in den anderen wechselt (siehe Abbildung 2), der angewandte Effekt.