Universiteit van Amsterdam

Multimedia project

Individueel verslag

26 augustus 2018

Student:
R.A.J. Wacanno
11741163

Practicumgroep:
EQFix

Cursus:
Multimedia

1 Introductie

In het volgende document staat beschreven aan welke onderdelen van de applicatie persoonlijk werkzaamheden zijn verricht en welke keuzes er zijn gemaakt met betrekking tot de implementatie van deze onderdelen.

2 Implementatie

2.1 Integratie binnen Android

In de eerste instantie had de applicatie een duidelijke opzet. Er zou een analyse met behulp van de microfoon van de telefoon plaatsvinden, waarna parameters voor een filter bepaald konden worden. Met dit filter zou vervolgens het onderschepte geluid van het *Android* systeem gemanipuleerd worden alvorens het weer naar de luidsprekers of koptelefoonuitgang van de telefoon gestuurd zou worden.

Hiervoor zou het nodig zijn het systeemgeluid op te vangen en weer terug te geven aan *Android*. Na grondig te hebben rond gezocht naar een manier om dit voor elkaar te krijgen bleek dat dit, in ieder geval voor onze toepassing, niet mogelijk was. Het opnemen — dan wel onderscheppen — van het systeem is geluid is om meerdere redenen binnen Android niet mogelijk. Allereerst zijn er natuurlijk veiligheids- en privacy-overwegingen; je zou immers niet willen dat er een applicatie continue zit mee te luisteren. Dit probleem had echter verholpen kunnen worden met het implementeren van speciale bevoegdheden. Een andere en in zekere zin onoplosbare reden dat het onmogelijk is gemaakt zijn overwegingen met betrekking tot auteursrecht. Mocht het namelijk mogelijk zijn om toegang te krijgen tot het systeemgeluid, dan zou ook het geluid van een muziekapplicatie als *Spotify* kunnen worden opgenomen.

Er is uiteindelijk gekozen voor het implementeren van een eigen muziekspeler binnen de applicatie op wiens geluid het berekende filter toegepast kan worden.

2.2 Frequentie-responsanalyse

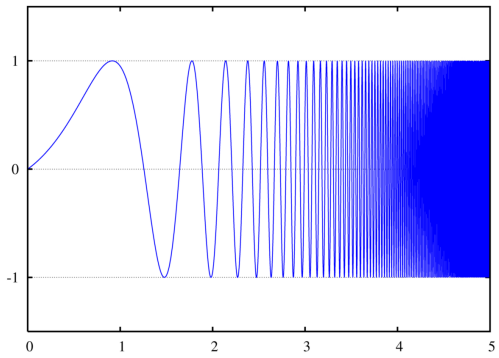
Een van de onderdelen van de analyse die binnen de applicatie plaatsvindt, is de frequentie-responsanalyse. Bij deze analyse wordt bepaald in welke mate bepaalde frequenties aanwezig zijn binnen een gegeven geluidsfragment. In het geval van de applicatie moet bepaald worden of iedere frequentie door de even luid wordt waargenomen op de plek van analyse.

2.2.1 Testmethoden

Voor het testen van de frequentie-respons van alle frequenties zijn meerdere methoden. Hierbij kan ofwel gebruik gemaakt worden van een *frequency sweep* of witte ruis.

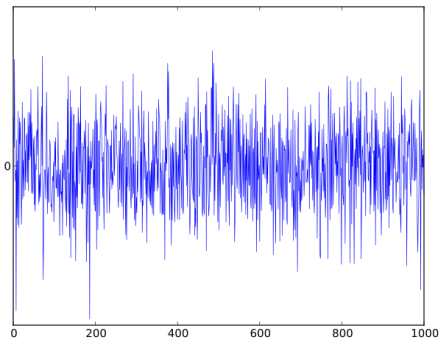
Bij het gebruik van een *frequency sweep* (een toon met een steeds toenemende frequentie) wordt deze afgespeeld en tegelijkertijd opgenomen. De golf van een dergelijke *sweep* is in figuur 1 te zien. Na het opnemen wordt de verkregen opname-*PCM* geanalyseerd en wordt deze met de origineel uitgezonden audio vergeleken om zo te welke frequentie hoe luid te horen was.

Het gebruik van *frequency sweep* heeft een vervelende bijkomstigheid en dat is de afhankelijkheid van *timing*. Op een bepaald moment is namelijk een bepaalde frequentie te horen en hiermee moet worden rekening gehouden bij het bepalen van de amplitude van de betreffende frequentie.



Figuur 1: Een plot van de golf van een sweep

Wanneer witte ruis gebruikt wordt, wordt deze net als de *sweep* afgespeeld en tegelijkertijd opgenomen. Deze techniek is gestoeld op het principe dat in witte ruis theoretisch gezien iedere frequentie even sterk aanwezig is. Als er dus in de opname een frequentie sterker of minder sterkt voorkomt, kan gesteld worden dat dit een gevolg is van de omgeving of het reproductiemedium.



Figuur 2: Een plot van witte ruis

Binnen de applicatie is gekozen om gebruik te maken van de witte-ruismethode vanwege zijn gebrek afhankelijkheid van *timing*. De originele witte ruis en het opgenomen fragment worden samen naar een *deconcolver* gestuurd waar vervolgens een filter-*kernel* uit wordt teruggegeven. Deze filter-*kernel* wordt door de *convolver* gebruikt om de muziek te manipuleren.

2.2.2 Ruis

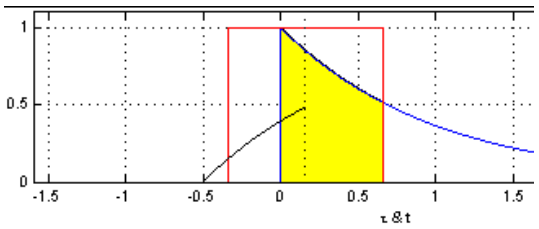
Voor het analyseren van de frequentie-respons met behulp van witte ruis moet deze ruis op de een of andere manier gegenereerd worden. Gezien de willekeurige aard van de ruis is dit mogelijk door een *PCM-array* te vullen met willekeurige getallen. Dit geeft echter in het geval van geluid, en dan in het bijzonder met het ook op het produceren van theoretisch iedere frequentie, niet het juiste effect [Smith et al., 1997]. Er is daarom gebruik gemaakt van de volgende formule:

$$X = \sqrt{(-2 \log R_1)} \cdot \cos(2\pi R_2)$$

Hier is X het getal dat in de *PCM* zal worden geplaatst en zijn R_1 en R_2 twee door de computer gegenereerde willekeurige getallen.

2.3 Convolutie

Voor het daadwerkelijke manipuleren is gekozen om gebruik te maken van convolutie. Dit is een proces waarbij twee functies worden gecombineerd. Figuur 3 is hier een illustratie van. In feite wordt de rode functie steeds over de blauwe gehaald en wordt de onderliggende oppervlakte berekend. Dit resulteert in de zwarte functie.



Figuur 3: Convolutie op twee functies

Binnen de applicatie is convolutie bruikbaar als geluidssignalen gezien worden als functies. Neem bijvoorbeeld de onderstaande vergelijking:

$$f(x) * g(x) = h(x)$$

Hier is $f(x)$ het oorspronkelijke geluidsignaal en $h(x)$ het signaal dat de luisteraar opvangt. $g(x)$ is een vervorming die door de omgeving en het reproductiemedium ontstaat en deze laat signaal $h(x)$ van $f(x)$ afwijken. Om dit te voorkomen kan een convolutie worden toegepast op het oorspronkelijke signaal $f(x)$ en de inverse van $g(x)$. Dit is geïllustreerd in onderstaande vergelijking.

$$f(x) * g(x)^{-1} = \hat{h}(x)$$

Gesteld kan worden dat deze $\hat{h}(x)$ dichter bij $f(x)$ ligt dan $h(x)$, ergo:

$$f(x) \approx \hat{h}(x)$$

De $g(x)^{-1}$ is binnen de applicatie de filter-*kernel* welke tijdens analyse verkregen wordt. Volgens het volgende algoritme vindt de convolutie plaats van het oorspronkelijke signaal en de filter-*kernel*:

```
function CONVOLVE(signaal, kernel)
    convolutie[] = float[#signaal + #kernel]{0 ... 0}
    for index i in signaal do
        for index k in kernel do
            convolutie[i + k] += (signaal[i] · kernel[k])
        end for
    end for
    return convolutie
end function
```

Op zichzelf is dit algoritme echter nog niet voldoende. Het uit de convolutie resulterende signaal is namelijk langer dan het originele signaal. Om dit te verhelpen is een simpele lus-buffer geïmplementeerd welke het overgebleven stukje *PCM* steeds in de volgende convolutie bijvoegt.

2.4 *Presets*

Binnen de applicatie is het mogelijk gemaakt voor gebruikers om hun analyseresultaten in een bestand op te slaan zodat er meerdere van deze bestanden in verschillende scenario's te gebruiken zijn en zodat deze gedeeld kunnen worden. Ieder bestand bevak de filter-*kernel* en de vertragingen die uit het *transient*-analyseonderdeel zijn gekomen.

2.5 Persoonlijke herconfiguratie

Om gebruikers in staat te stellen zelf ook het een en ander aan het filter aan te passen is een simpele *equalizer* in de applicatie ingebouwd. Met deze kunnen de lage, midden en hoge frequenties afzonderlijk versterkt en verzwakt worden. De hiermee aangegeven parameters worden ook in het eerdergenoemde *preset*-bestand opgeslagen.

Referenties

[Smith et al., 1997] Smith, S. W. et al. (1997). The scientist and engineer's guide to digital signal processing.