Best-practice geluidsgeneratie en -verwerking
Onderzoeksvoorstel Bachelorproef

Victor Van Weyenberg[1]

Samenvatting
Audio hardware is duur. Software-based Digital Audio Workstations (DAW) bieden dankzij de groeiende werkkracht van persoonlijke computers een goed alternatief. Dit onderzoek levert via een vergelijkende studie tussen drie Software Sound Synthesizer (SWSS) libraries een best-practice voor geluidsgeneratie en -verwerking. Vereenvoudigde algoritmes creëren ruimte voor een betere geluidskwaliteit. Dit onderzoek zal een hulp zijn voor developers bij de keuze of creatie van hun SWSS's.

Sleutelwoorden
Onderzoeksdomein. Software — Development — Geluid — Generatie — Verwerking — Performantie — Efficiëntie — Kwaliteit

Co-promotor

Contact: [1] victor.vanweyenberg.w3751@student.hogent.be;

Inhoudsopgave

1. Introductie

Als tegenhanger van hardware synthesizers, maken software-based Digital Audio Workstations (DAW) de laatste jaren om tal van redenen steeds meer opgang. De werkkracht van persoonlijke computers groeide en groeit enorm. DAW's bieden een goedkoper alternatief op het aankopen van alle hardware die nodig is voor audiobewerking. Alle functies van de dedicated hardware kunnen ook ondergebracht worden in één DAW, dat maakt DAW's een meer polyvalent substituut.

Voor de development van DAW-software zijn er verschillende libraries beschikbaar bedoeld voor geluidsgeneratie en -verwerking. Deze libraries worden verder benoemd als Software Sound Synthesizers (SWSS). Dit onderzoek neemt de werking van drie SWSS's onder de loep. Er zal een vergelijking gemaakt worden tussen de broncode en performantie om zo een best-practice van software geluidsgeneratie en -verwerking te omschrijven.

Dit onderzoek zal een mapping creëren die voor developers de efficiëntie, performantie en kwaliteit van SWSS's en DAW's aantoont.

2. State-of-the-art

Pope (1993) heeft een gelijkaardige studie gedaan tussen de destijds drie meest gebruikte SWSS's. In zijn paper bespreekt hij *Music V*, een SWSS geschreven in 1967. Schrijven in *Music V* bestaat uit twee delen: de definitie van het instrument en het schrijven van de partituur. Voor het definiëren van instrumenten beschrijft *Music V* computer modellen van oscillators, filters, envelope generators, amplifiers etc. (Moog & Gamer, 2015) Door die modules correct te patchen, bekomt men het gewenste timbre. Naast het timbre kon men ook een partituur schrijven die instaat voor de melodie van de geprogrammeerde muziek.

Dit is een structuur die we ook in hedendaagse SWSS's terugvinden. *JSyn* (Burk, 2017), *Beads* (Bown, 2017) en *JMusic* (Brown, 2015) zijn drie open-source, Java-based SWSS's die aan die structuur voldoen. Omwille van de gelijkaardige structuur zijn deze libraries gekozen om vergeleken te worden in dit onderzoek.

Pope (1993) besprak *Music V* omwille van de simpliciteit. Dat was voor developers de grootste troef van de SWSS. Pope (1993) refereert naar *Music V* als de grondlegger van latere SWSS's. Om dit te illustreren bespreekt hij drie C-based SWSS's. Hij vervolgt met een vergelijkende studie op basis van architectuur en performantie.

De hierboven vermelde studie dateert van 1993. DAW's worden vandaag de dag nog uiterst zelden in pure C geschreven. Er is intussen een groot aanbod aan SWSS libraries. Dit onderzoek wilt de studie van Pope (1993) actualiseren gebruikmakend van de drie Java SWSS libraries.

3. Methodologie

De eertse stap is een vergelijking van de architectuur van de drie libraries met commentaar op vlak van efficiëntie.

De tweede stap bestaat uit een performantietest op geluidsgeneratie en -verwerking. Er worden verschillende testscenario's uitgeschreven met meerdere stemmen, bewerkingen en melodieën. Ieder scenario zal getest worden op dezelfde machine in ieder van de gekozen SWSS libraries. Hier wordt er ook rekening gehouden met de kwaliteit van de output.

Op basis van architectuur, performantie en kwaliteit wordt een best-practice omschreven voor software geluidsgeneratie en -verwerking.

4. Verwachte resultaten

Een oppervlakkige lezing van de broncode van de libraries, levert op dat JSyn bij het aanmaken van één oscillator al snel veel *UnitPort* objecten creëert. Deze *UnitPort* objecten worden gebruikt om de connecties met andere modules te maken en zo een stem te vormen. Omdat er al zoveel geheugen gealloceerd wordt per stem, wekt dit het vermoeden dat JSyn niet hoog zal scoren op de performantietest.

Een meer eenvoudige vorm van modules connecteren vinden we terug bij JMusic. Iedere opvolgende module in de zogenoemde *Chain* krijgt de voorgaande module mee als parameter in de constructor. Daarnaast valt ook op dat alle basis oscillator types ondergebracht zijn in één klasse. Dit met als nadeel dat de code van het genereren en verwerken van de audio buffer zeer groot wordt. JMusic lijkt in eerste opzicht niet de efficiëntste code te hebben maar scoort qua performantie wel beter dan JSyn.

Beads valt in het geluidsgeneratieproces al zeer snel terug op basis datastructuren en wiskunde. Voor ieder basis stem type is er - net zoals in JSyn - een aparte *Buffer*-klasse voorzien. Deze kunnen afgespeeld worden door de relatief kleine *WavePlayer*-klasse. Hypothese: Beads heeft de meest performante en efficiënte code.

De bespreking van code in dit hoofdstuk is zeer oppervlakkig en enkel beperkt tot geluidsgeneratie met als enige doel een hypothese op te stellen.

5. Verwachte conclusies

Hoewel JSyn niet de meest performante code heeft, bereikt hij met de extra geheugenallocatie een hogere geluidskwaliteit. De hiervoor besproken *UnitPort* objecten zijn bedoeld om modules te connecteren, wat ook de essentie van een SWSS is.

Wanneer men veel meerdere stemmen genereert of bewerkingen uitvoert, zullen er sneller kwaliteitsbeperkingen optreden bij JMusic en JSyn dan bij Beads.

Het KISS-principe levert de best-practice voor geluidsgeneratie en -verwerking. Het vereenvoudigen van algoritmes zoals geluid filteren of een blokgolf genereren bespaart geheugen zodat het programma meerdere stemmen kan genereren en bewerking kan uitvoeren zonder kwaliteitsverlies. De developer kan dan opteren om dat geheugen te gebruiken voor betere buffermanagement, betere verwerkingskwaliteit of hogere sample rates.

Referenties

Bown, O. (2017). *Beads JavaDocs*. Verkregen van http://www.beadsproject.net/doc/

Brown, A. (2015). *JMusic JavaDocs*. Verkregen van http://explodingart.com/jmusic/jmDocumentation/index.html

Burk, P. (2017). *JSyn JavaDocs*. Verkregen van http://www.softsynth.com/jsyn/docs/javadocs/

Moog, R. A. & Gamer, C. (2015). Electronic instrument. Verkregen van https://www.britannica.com/art/electronic-instrument

Pope, S. T. (1993). Machine Tongues XV: Three Packages for Software Sound Synthesis. *Computer Music Journal*, *17*(2), 23–54. Verkregen van http://www.jstor.org/stable/3680868