SuperCollider 2 – Filter

I denna laboration ska ni fortsätta att arbeta i ljudprogrammeringsmiljön SuperCollider (SC). Laborationen går ut på att vidareutveckla arbetet från den första SC-laborationen och testa på olika typer av filter och filterinställningar.

Inför laborationen

Förberedelse – 1. Gör klart den första SC-laborationen

Denna laboration bygger vidare på den första laborationen där SC introducerades och olika syntesmetoder testades. Därför är det bra om den första laborationen är klar till denna laboration (men den behöver inte vara redovisad).

Förberedelse – 2. Läs på om filter

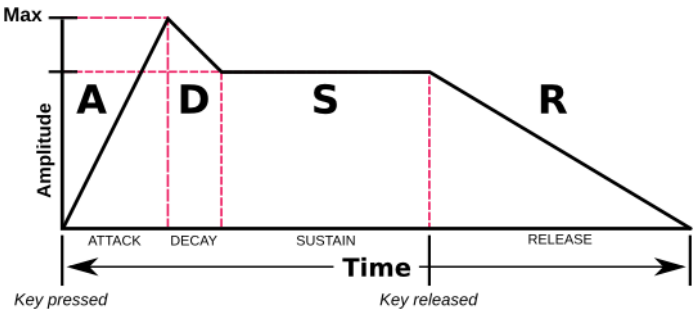
Kolla igenom följande länkar:
https://www.masteringbox.com/filter-types/
https://en.wikipedia.org/wiki/Audio_filter

Uppgiften

I den här laborationen ska ni vidareutveckla synthdefinitionen från den första SC-laboration, och (huvudsakligen) experimentera med olika filter.

Skapa en envelop

Det enkla arpeggiot som skapades i förra SC-laborationen låter lite tråkigt och luddigt när alla toner spelas med legato. Detta går dock enkelt att åtgärda med en envelop. En envelop är ljudets volymkontur, och det klassiska exemplet är ADSR-envelopgeneratorn som består av Attack, Decay, Sustain, Release. Attack är tiden det tar från att en tangent trycks ned på en synt tills ljudvolymen når max, Decay är tiden det tar från att attackfasen är klar till ljudet når sitt viloläge dvs Sustainnivån, och Release är tiden det tar efter att tangenten släpps upp tills ljudet är helt tyst.

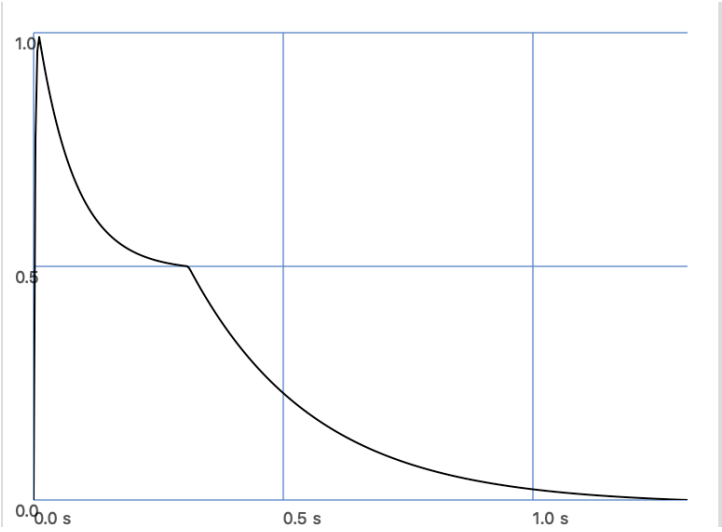


I SC finns det många olika envelopformer att välja mellan. Man skapar en envelop med `EnvGen`. Men, det skulle gå att skapa en ganska komplex vågform om envelopen loopades och justerades till att svänga mellan -1 och 1. En envelop är normalt en kontrollsignal, och därför är `EnvGen` oftast i control rate `.kr`. EnvGen skrivs så här: `EnvGen.kr(envelope, gate: 1.0, levelScale: 1.0, levelBias: 0.0, timeScale: 1.0, doneAction: 0)`, och det viktigaste här är inputargumentet för `envelope` och `gate`., där `envelope` är själva formen på envelopen (antingen en `Env`-instans eller en kontrollarray) och `gate` är kontrollsignalen om tangenten på klaviaturet är nedtryckt (1) eller inte (0).

Läs mer här:
https://doc.sccode.org/Classes/EnvGen.html

Med en instans av `Env` skapas själva konturen av envelopen. Denna skrivs som `Env.new(levels: [ 0, 1, 0 ], times: [ 1, 1 ], curve: 'lin', releaseNode, loopNode, offset: 0)`, men det finns ett antal versioner på `Env` som underlättar arbetet att skapa nivåer och tider, exempelvis ADSR.

En ADSR skrivs: `Env.adsr(attackTime: 0.01, decayTime: 0.3, sustainLevel: 0.5, releaseTime: 1.0, peakLevel: 1.0, curve: -4.0, bias: 0.0)`. Låt oss bryta isär detta exempel. Attacktiden är 0.01 sekunder, det är en kort attacktid. Att ange 0 som attacktid kan resultera i korta klickljud när ljudet startar. Det kommer alltså ta 0.01 sekunder för ljudet att gå från tyst till max ljud. Sedan tar det 0.3 sekunder för ljudet att gå till sustainnivån som är på 50% av max ljud. När tangenten på klaviaturet släpps upp och ljudet ska slutar tar det 1 sekund innan ljudet är helt tyst. Peaknivån, `peakLevel`, är 1, dvs nivån som är max för ljudet. Envelopens krökning styrs av `curve`, där 0 är en linjär kurva, medan värden över eller under 0 skapar logaritmisk respektive exponentiell krökning. Med `bias` kan envelopen justeras i höjdled genom att addera ett värde till envelopen, man kan säga att envelopens DC-offset justeras. Genom att skriva `.plot` efter en `Env` så öppnas ett fönster där envelopens kontor ritas ut. Här nedan plottas ADSR-exemplet ovan, men med en Sustain-tid som är 0 då ingen tangent blir nedtryckt eller simulerad vid plot.



Läs mer här:
https://doc.sccode.org/Classes/Env.html

Skriv i exemplet på ADSRen ovan i synthdefinitionen från förra laborationen. Tänk på att argumenten inte behöver specificeras om de skrivs i rätt ordning, och argument som inte ges blir default:

```
var envelope = EnvGen.kr(Env.adsr(0.01, 0.3, 0.5, 1.0, 1, -4.0));
```

Jag föredrar att lägga envelopen i en variabel i stället för att skriva den direkt på oscillatorn. För enveloper i synthdefinitioner kan man inte använda `.plot` , i stället får ni kopiera envelopkoden och exekvera den raden med `.plot` utanför er exekverbara kod inom parenteser, om ni vill plotta envelopen medan ni arbetar.

Envelopen lägger ni sedan på det summerade ljudet i synthdefinitionen genom att multiplicera ljudet med envelopen.

För att envelopen ska fungera behövs en `gate`-signal som signalerar tangentnedtryck och tangentuppsläpp på ett klaviatur. En `gate`-signal är ett argument som ges efter själva envelopkonturen, vilket även ska sändas från klientsidan och tas emot som ett input-argument i synthdefinitionen.

Ändra sedan i koden på klientsidan så att en tangennedtryckning (gate = 1) skickas med varje ny ton, sedan ska for-loopen vänta halva tiden (dvs 0.15 sekunder i stället för 0.3), därefter ska ett tangentuppsläppas (gate = 0) sändas och sedan ska for-loopen vänta igen. Så, istället för att ge en ny ton och vänta och sedan loopa, kommer loopen ge en ny ton och simulera tangentnedtryck, vänta lite, släppa upp tangenten, vänta lite till och sedan loopa.

Ibland behöver servern bootas om för att ändringar ska slå igenom i synthdefinitioner. Det kan också fungera att avbryta exekveringen och sedan exekvera igen…

Hur låter det? Hörs det att envelopen påverkar ljudets volymmässiga kontur?

_____

_____

_____

Om ni använder sinusoscillatorer är det inte säkert att attacken i ljudet är så tydlig, prova i så fall en vågform med mer övertoner. Experimentera sedan med olika inställningar på envelopgeneratorn, testa med Release, testa med 0 i Sustain, testa med 0.5 i Attack, osv. Blir det bättre? Vilka inställningar föredrar ni, och varför?

_____

_____

_____

Lägg till brus

Ofta vill vi undvika brus. Brus är egentligen all form av störande ljud, även om vi oftast tänker på högfrekvent brus som när gamla TV-apparater inte hade någon signal. Men, brus kan också vara en väldigt bra källa för att skapa ljud och effekter, eller för att använda till mätningar av högtalare eller annan ljudteknisk utrustning. Det finns olika typer av brus som är enkla att använda i SC, som: `WhiteNoise`, `PinkNoise`, `BrownNoise`, `GrayNoise` och `ClipNoise`. De olika brustyperna skrivs på samma sätt, med två argument, `mul` som multiplicerar utgången med detta värde (default = 1) och `add` som adderar utgången med detta värde (default = 0).

Skriv en liten synthdef för att testa de olika brusen så att ni kan jämföra dem:

```
(
SynthDef(\noiseSynth, {
    Out.ar(0,{WhiteNoise.ar()}!2);
}).play;
)
```

Experimentera med de olika brustyperna. Hur låter de, hur ser frekvensinnehållet ut, vad är det som skiljer dem åt?

_____

_____

_____

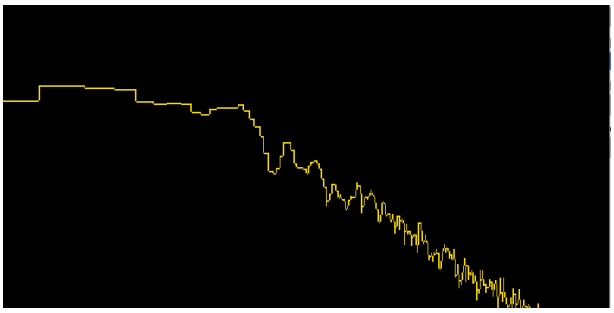
Läs mer om brus här:
https://en.wikipedia.org/wiki/White_noise
https://en.wikipedia.org/wiki/Pink_noise
https://en.wikipedia.org/wiki/Colors_of_noise

Lägg sedan till ett brus till er ursprungliga synthdefintion och mixa ihop med de övriga ljudkällorna.

Lågpass- och högpassfiltrering av ljudet

Nu finns ett mixat ljud som innehåller ganska mycket övertoner. Det vore bra om ni har två oscillatorer som är lite lite snedstämda, och helst med vågformer som har mer övertoner än sinusvåg, och någon form av brus, samt en suboktavgenerator som ligger en oktav under grundtonen för de andra oscillatorerna. En suboktavgenerator skapas genom att göra en ny oscillator och sedan ta MIDI-notnumret -12. Egentligen går det åtta toner på en oktav, men det är 12 tangenter med halvtonerna. Tänk på att justera nivån på ljudet när ni summerar så att ni inte överstyr ljudet.

Normalt filtreras ljudet innan man applicerar envelopen. Så, det är det summerade ljudet som ska stoppas in i filtret. Testa först ett lågpassfilter (LPF). Ett lågpassfilter dämpar frekvenser högre än brytfrekvensen i filtret. Ett lågpassfilter skapas med `LPF` och normalt ligger det i audio rate och skrivs: `LPF.ar(in: 0.0, freq: 440.0, mul: 1.0, add: 0.0)`. `LPF` är ett andra ordningens Butterworthfilter.



Läs mer här:
https://doc.sccode.org/Classes/LPF.html
https://en.wikipedia.org/wiki/Low-pass_filter
https://en.wikipedia.org/wiki/Butterworth_filter

Argumentet `in` är det summerade ljudet, `freq` är brytfrekvensen för filtret. OBS! Ange aldrig 0 i brytfrekvens. De övriga argumenten känner ni igen, och behövs normalt inte anges. Testa med en brytfrekvens på 440Hz.

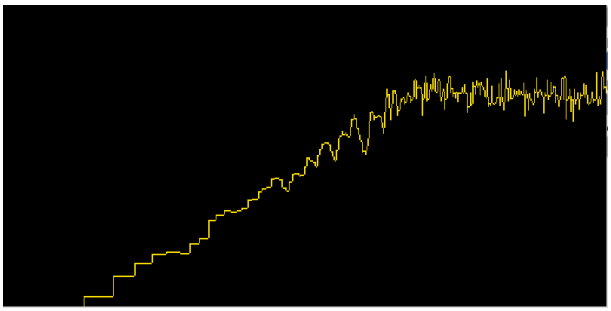
Hur låter det? Testa andra brytfrekvenser, hur påverkar det ljudet?

_____

_____

_____

Testa sedan att ändra LPF till ett högpassfilter (HPF). Ett högpassfilter dämpar frekvenser lägre än brytfrekvensen för filtret.



Hur låter det? Testa andra brytfrekvenser, hur påverkar det ljudet? Jämför frekvensanalysen mellan LPF och HPF.

_____

_____

_____

Läs mer här:
https://doc.sccode.org/Classes/HPF.html
https://en.wikipedia.org/wiki/High-pass_filter

Välj den filtertyp (LPF eller HPF) som ni tyckte lät bäst, och skapa sedan en LFO. En LFO (lågfrekvensoscillator) skapade ni vid den första SC-laborationen för att modulera tonhöjden och/eller amplituden. I det här fallet ska filtrets brytfrekvens moduleras. Använd `range` för att justera oscillatorns omfång, ange den lägsta respektive högsta brytfrekvensen ni vill använda i filtret.

Hur låter det? Testa olika brytfrekvenser i `range`. Testa olika vågformer för kontrollsignalen. Testa olika hastigheter på LFOn (frekvensen på kontrolloscillatorn). Hur låter det?

_____

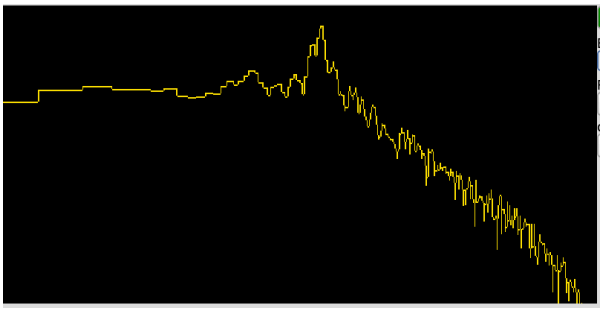
_____

_____

Resonans i filter

Det går också att använda resonanta filter i SC. Det innebär att en del av utsignalen från ett filter återmatas till ingången. Detta kan skapa en självsvängning i filtret, där filtret genererar en sinusvåg i brytfrekvensens frekvens. I SC kan man göra detta med `RLPF` respektive `RHPF`. Dessa skrivs som `RLPF.ar(in: 0.0, freq: 440.0, rq: 1.0, mul: 1.0, add: 0.0)` där argumentet `rq` står resonansen eller med andra ord relationen mellan bandbredd och brytfrekvens, och anges mellan 0 och 1 där 0 är max resonans och 1 är ingen resonans. OBS! Ange aldrig 0 i `rq`.

Läs mer här:
https://doc.sccode.org/Classes/RLPF.html
https://doc.sccode.org/Classes/RHPF.html



Testa att använda ett resonant filter och förändra värdet på `rq`. Hur låter det?

_____

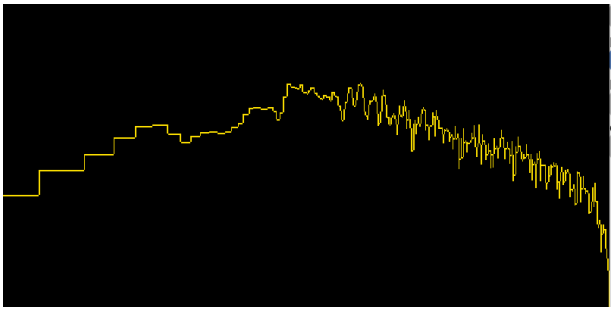
_____

_____

Läs mer här:
https://en.wikipedia.org/wiki/Resonance

Bandpassfilter

Ett annat filter som kan vara användbart är bandpassfilter, `BPF`. Ett bandpassfilter släpper igenom ett band av frekvenser över och under brytfrekvensen. Det skrivs så här: `BPF.ar(in: 0.0, freq: 440.0, rq: 1.0, mul: 1.0, add: 0.0).BPF` är också ett andra ordningens Butterworthfilter.



Läs mer här:
https://doc.sccode.org/Classes/BPF.html
https://en.wikipedia.org/wiki/Band-pass_filter

Testa att använda ett bandpassfilter och förändra brytfrekvenserna i `range`. Experimentera också med andra värden på `rq`. Hur låter det, och hur ser frekvensanalysen ut?

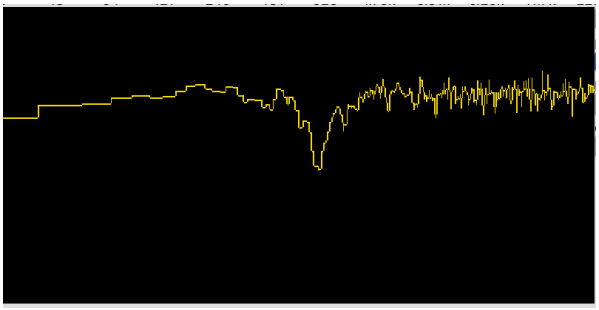
_____

_____

_____

Bandstoppfilter

Det finns också bandstopfilter (bandreject eller notchfilter) att använda i SC. Ett bandstoppfilter släpper igenom frekvenser ovanför och under brytfrekvensen för filtret, men dämpar frekvenserna närmast brytfrekvensen. Det skrivs `BBandStop.ar(in, freq: 1200.0, bw: 1.0, mul: 1.0, add: 0.0)` där `bw` är bandbredden på filtret där ett lågt värde (exempelvis 1) ger ett smalt filter (dvs ett brant filter) och ett större värde (exempelvis 6) ger ett brett filter.

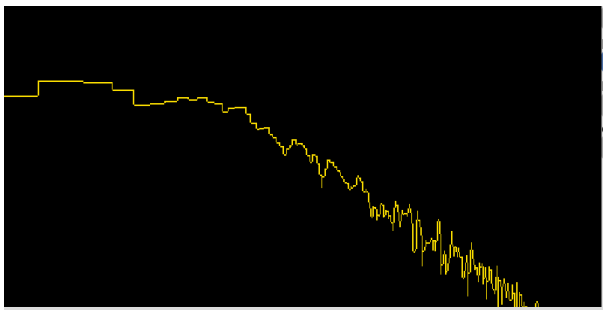


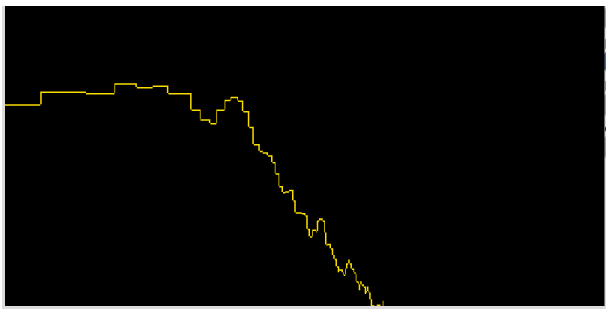
Läs mer här:
https://doc.sccode.org/Classes/BBandStop.html
https://en.wikipedia.org/wiki/Band-stop_filter

Testa att använda ett bandstoppfilter och förändra brytfrekvenserna i `range`. Experimentera också med andra värden på `bw`. Hur låter det, och hur ser frekvensanalysen ut?

Andra vs fjärde ordningens filter

Brantheten på ett filter, dvs hur mycket av oönskade frekvenser som finns kvar efter filtreringen beror på ordningen på filtret. En högre ordning dämpar bort mer oönskade frekvenser. Ett filter som därför är intressant att testa ytterligare är `BLowPass`. `BLowPass` är ett andra ordningens 12dB/oktav lågpassfilter. Det innebär att för varje oktav (dvs fördubbling av frekvensen) dämpas ljudet med 12dB. Om brytfrekvensen för filtret är ställd på 440Hz kommer ljudet vid 880Hz vara 12dB svagare, 24dB svagare vid 1760Hz, osv. Det finns också varianten `BLowPass4` vilket är ett fjärde ordningens filter, 24dB/oktav.





Till vänster används ett andra ordningens filter och till höger ett fjärde ordningens filter.

Gör en kopia på filterkoden, och sätt (tillfälligt) brytfrekvensen till en fast frekvens (exempelvis 400Hz) och använd `BLowPass`-filtret. Jag gjorde så här:

```
//var filteredOutput = RLPF.ar(mixedSounds, cutOffLFO, 0.5);
var filteredOutput = BLowPass.ar(mixedSounds, 400, 1);
```

Lyssna på ljudet. Ändra till `BLowPass4` och lyssna på ljudet igen. Hur skiljer sig ljuden åt? Hur påverkar ordningen på filtret ljudet? Prova gärna olika brytfrekvenser för att lyssna på skillnaden. OBS! Ange aldrig 0 i brytfrekvens.

Ändra resonansen, `rq`, från 1 (ingen resonans) till lägre och lyssna igen. OBS! Ange aldrig 0 i `rq`. Påverkas resonansen av filterordningen?

_____

_____

_____

Självsvängning i filter

Ett 24dB/oktavfilter (fjärde ordningens filter) går att få att självsvänga, och går då att använda som en sinusoscillator. Detta går att testa på även i SC.

Men (!) filter i SC fungerar inte så bra när de matas med värden nära 0. Det finns stor risk för väldigt starka ljud och knäpp när brytfrekvensen är 0 eller om resonansvärdet sätts till 0. Om ni vill experimentera med detta ta för vana att alltid ta av er hörlurarna när ni ska exekvera koden! Om inget hemskt starkt ljud uppstår så bör det vara säkert att ta på sig hörlurarna igen.

Börja med att sänka ljudvolymen på mixen av ljuden. Det enklaste hade varit att ta mixen gånger 0, men då är det ingen ljudsignal och då fungerar inte filtren i SC. Multiplicera därför mixen med 0.01, så blir ljudnivån så låg att ni inte direkt hör den.

Sedan sätter ni brytfrekvensen på filtret (`BLowPass4`) till `inputFrequency` (eller vad ni kallade inputargumentet för tonerna från loopen på klientsidan) och ni använder `.midicps` för att göra om MIDI-notnumret till frekvens i Hz. Sedan sätter ni resonansen, `rq`, till 0.01.

Ta av er hörlurarna (för säkerhets skull) och exekvera koden. Ni ska nu ha en sinusvåg som spelar arpeggiot.

Denna självsvängning, sinusoscillator, går också att framställa med andra ordningens filter i SC även om det oftast inte är så i analoga tillämpningar.

Fler filter kan ni hitta om ni söker i Help browser efter filter.

Ta in externt ljud och filtrera

Det är tämligen enkelt att ta in ljud från exempelvis datorns mikrofon eller ljudingång. Genom att köra `ServerOptions.inDevices.postln;` listas alla inputdevices som finns tillgängliga i datorn. Dessa kan man sedan namnge exakt som de listas, eller som devicenummer `0`, `1`, osv. Om ni arbetar med en laptop med både de inbyggda högtalarna och den inbyggda mikrofonen, finns det stor risk för att ni får rundgång, så jag rekommenderar att ni testar detta med hörlurar.

Med hjälp av `s.meter;` går det att se och läsa av uppspelad ljudnivå såväl som inkommande ljudnivå.

I synthdefinitionen använder ni `SoundIn` för att ta emot ljud från en extern ingång. Tänk på att `SoundIn` bör ha audio rate, och anges så här: `SoundIn.ar(bus: 0, mul: 1.0, add: 0.0)`, där `bus:` är namnet eller numret på den enhet ni vill ta in ljudet från. Skapa en variabel för det inkommande ljudet, och sedan kan ni använda denna variabel i stället för de summerade oscillatorerna (och bruset) som ingång i filtret.

Läs mer här:
https://doc.sccode.org/Classes/SoundIn.html

Nu ska det inkommande ljudet filtreras och påverkas ljudvolymsmässigt av envelopgeneratorn. Ni kan välja att bara skicka det filtrerade ljudet till utgången, men att ha kvar envelopen visar på hur en extern ljudsignal kan påverkas både frekvens- och amplitudmässigt i SC.

Spela upp och filtrera audiofiler

I stället för att filtrera synthljud kan det vara kul att testa att filtrera en ljudfil (en wav-fil) om ni har en sådan i datorn. Det görs med en `Buffer` till vilken man läser in ljudfilen, enligt `Buffer.read(server, path, startFrame: 0, numFrames: -1, action, bufnum)`. Där `server` normalt är `s` och `path` är sökvägen till ljudfilen. `Buffer` funkar bara med wav-filer.

Buffer bör vara fylld med en ljudfil för att uppspelning ska funka, så för detta exempel valde jag att lägga ljudfilen i en variabel och köra den först i `fork`-processen men utanför `loop`-funktionen. Så här:

```
fork({
    var soundFileBuffer = Buffer.read(s,"/Users/niklas/Documents/TNM103 - Ljudteknik/laborationer/music.wav");
```

Därefter valde jag att låta processen vänta (`.wait`) en sekund innan jag sände bufferten till synthdefinitionen med:

```
~myFirstSynth.set(\bufnum, soundFileBuffer.bufnum);
```

Anledningen till att lägga in en kort paus är för att ljudfilen ska hinna läsas in till bufferten innan den skickas till synthdefinitionen.

Läs mer här:
https://doc.sccode.org/Classes/Buffer.html

Därefter behöver synthdefinitionen få ett till inputargument som jag valde att kalla `bufnum`. I själva synthdefinitionen behövs ytterligare kod, en sample playback oscillator för att spela upp ljudet från buffern. Det görs med `PlayBuf`.

För detta exempel räcker det med att ange antalet kanaler (`2` eftersom det är stereo) och `bufnum` till `PlayBuf`. Så här:

```
var audioFile = PlayBuf.ar(2, bufnum);
```

Läs mer här:
https://doc.sccode.org/Classes/PlayBuf.html

Där efter kan `audioFile` filtreras och lämpligen tas envelopen bort från ljudet som går till utgången (`output` i mitt exempel), men att ha kvar envelopen visar också ett bra exempel på hur amplituden i ett samplat ljud kan bearbetas i SC.

Vidare utforskning

Det finns fler typer av filter att experimentera med i SC, och för att komma åt dessa kan man behöva ladda ned SC3-plugins. Dessa finns att ladda ned här, men är inget krav för den här kursen:
http://supercollider.github.io/sc3-plugins/

Reverb och delay

Avslutningsvis ska ett enkelt delay (eko) och reverb testas. Med `AllpassC` skapas ett Schroeder allpassfilter att använda i en delaykedja. Detta filter skrivs `AllpassC.ar(in: 0.0, maxdelaytime: 0.2, delaytime: 0.2, decaytime: 1.0, mul: 1.0, add: 0.0)`. Inputargumenten är `maxdelaytime` som anges i sekunder och används för att skapa en

buffer för delayet, `delaytime` är delaytiden i sekunder och `decaytime` är tiden det tar för ekot att dämpas 60dB i sekunder.

Läs mer här:
https://doc.sccode.org/Classes/AllpassC.html

Ett delay, eller ett eko, har långsamma reflektioner medan ett reverb har snabbare och mer slumpartade reflektioner. Genom att ange delaytider på 0.2 och en decaytid på 0.66, skapas ett delay som passar ganska bra i tempo mot arpeggiot.

Hur låter det? Hörs effekten, och passar ekot/delayet?

_____

_____

_____

För att effekten ska bli bra behövs ljudet i synthdefinitionen (efter envelopen) kopieras/läggas i en ny variabel. Denna variabel är det som används som input i allpassfiltret. Därefter behövs ljudet från allpassfiltret mixas med ljudet (och troligtvis dämpas något i amplitud) innan det skickas till utgången.

Blir ekoeffekten tydligare nu? Prova att ändra decaytiden, hur påverkar det effekten?

_____

_____

_____

Om delaytiden i stället görs kort och slumpas kommer ett reverb att skapas. Med `Rand` kan ett slumpat värde (`float`) väljas enligt `Rand.new(lo: 0.0, hi: 1.0)`. Sätt decaytiden (i det här fallet reverbtiden) till 6 sekunder så att inte reverbljudet klingar ut och tystnar för tidigt.

Läs mer här:
https://doc.sccode.org/Classes/Rand.html

Hur låter reverbet? Skapas en känsla av rumsakustik?

_____

_____

_____

Schroeders reverberator hade tre allpassfilter i serie. Detta görs tämligen enkelt i SC genom att använda `.do`. Det är en typ av kontrollstruktur som loopas så många gånger som man anger före punkten, enligt `collection.do(function)` där `collection` är antalet loopvarv (3 i Schroeders fall) och funktionen skrivs inom måsvingar.

Jämför när 3 allpassfilter används mot 1 allpassfilter? Hur låter det, och vad är skillnaden? Blir reverbet mer verkligt ett riktigt akustiskt rum?

_____

_____

_____

Testa avslutningsvis att först använda delayet och sedan lägga reverb på delayet, i båda fallen kör ni 3 allpassfilter enligt Schroeders reverberator. Hur låter det? Skapas en bra effekt?

_____

_____

_____

Fler reverb

Det finns fler reverb i SC för de som vill testa på ytterligare, exempelvis FreeVerb, FreeVerb2 och GVerb. Sök i Help browser (eller Googla) för fler idéer.

Vidare läsning

Om spänningsstyrda filter:
https://en.wikipedia.org/wiki/Voltage-controlled_filter

Om självsvängning:
https://en.wikipedia.org/wiki/Self-oscillation