

Synthesizer-programmering

**Kort introduktion til digital lyd- og musikprogrammering
i Pure Data (PD)**

Anders Monrad, juli 2024

1. Programmeringssprog

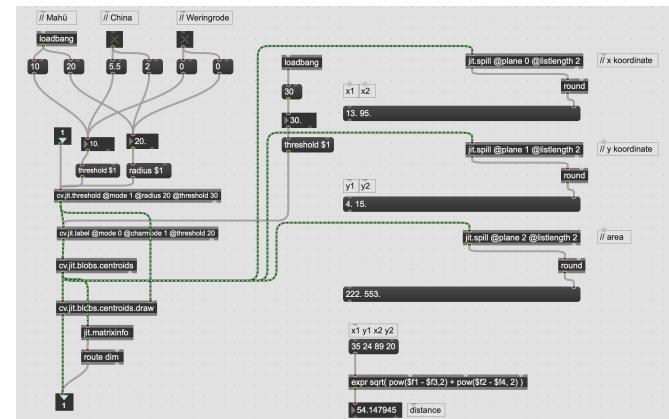
Hvordan programmerer man elektronisk lyd?

Normalt tænker i nok, at kode ser sådan her ud:

```
666     event LfoFrequency (float value)
667     {
668         value *= 0.01f;
669
670         float octave = -5.5f + 9.725f * value; // 10 Oktaven
671         int octaveInt = int (floor (octave)); // 0 = Viertelnoten; -6 = 16 Takte, 4 = 64tel
672
673         state.lfoRatio = 0.015625f * (1 << (octaveInt + 6)); // (1 << n) = (2 ^ n), 2^{(-6)} = 0.015625
674
675         if (octave - octaveInt > 0.58)
676             state.lfoRatio *= 1.5f;
677
678         if (state.midiTempo == 0) // MIDI Sync is off
679         {
680             let pitch = -92.7f + 116.7f * value; // -92.7 to 24
681             state.lfoFreq = pitchToFreq (pitch);
682         }
683     }
```

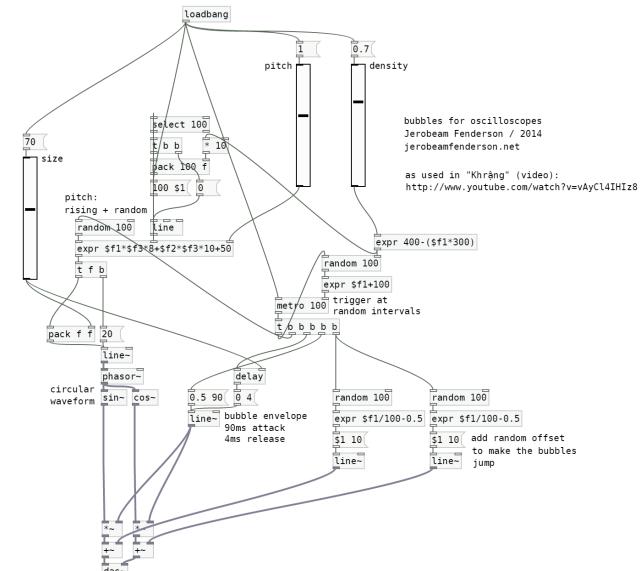
Og man kan godt kode lyd vha. tekstkode, bl.a. med programmeringssprogene Csound, Supercollider, ChucK, det helt nye sprog C Major (som er vist i dette kodeeksempel) m.fl.

Men i musikerkredse er såkaldte "patching-sprog" (visuelle programmerings-sprog) blevet standard. Det er dem der underves i på musikkonservatorier i hele verden. F.eks. det kommercielle sprog "Max/MSP":



Max/MSP-patch

Eller Max/MSP's søster-sprog Pure Data (PD), som har dén fordel, at det er open-source, og dermed gratis. Det er dét, vi vil bruge i dette forløb:



Pure Data-patch

At Patching-sprog har vundet
indpas blandt musikere
hænger sammen med, at det
er umiddelbart mere intuitivt
end tekstkode – og dermed
hurtigere at prototype i,
hvilket passer bedre til
musikeres kreative process
som i højere grad er trial/error
– ikke ligesom en ingeniør
eller software-udvikler der
planlægger sin kode i detaljer
inden den implementeres.

Og først og fremmest så
hænger det også sammen
med tradition. Det svarer til
"patching" på klassiske,
analoge synthesizere (som
mange musikere i øvrigt
stadig bruger):

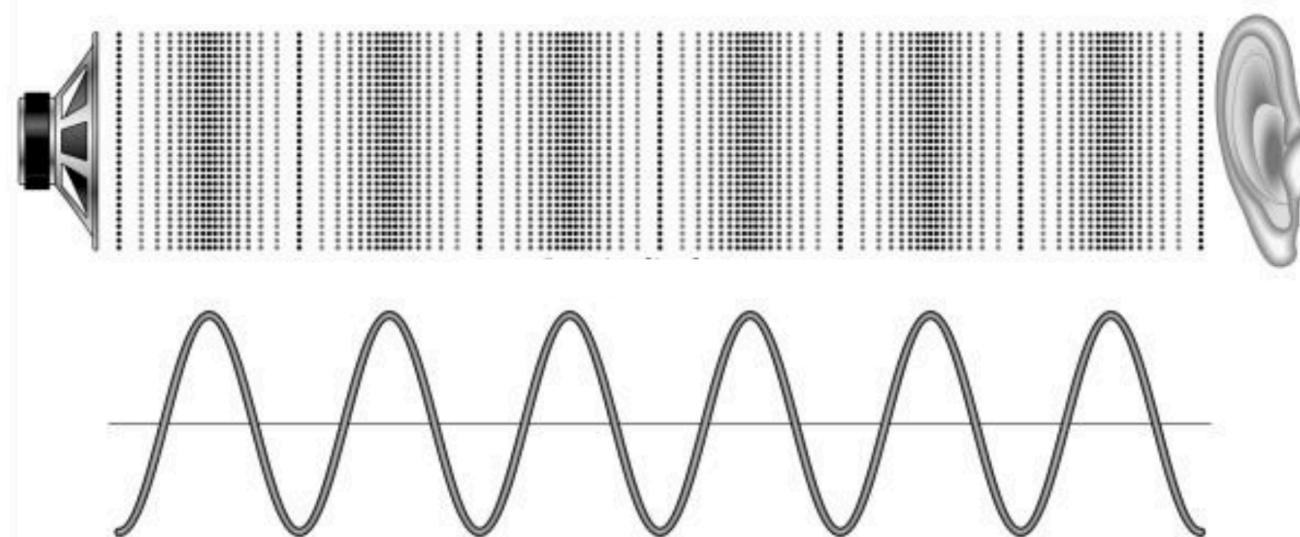


Klaus Schulze (Tangerine Dream) - 1970's

2. Lydbølger

Hvad er lydbølger?

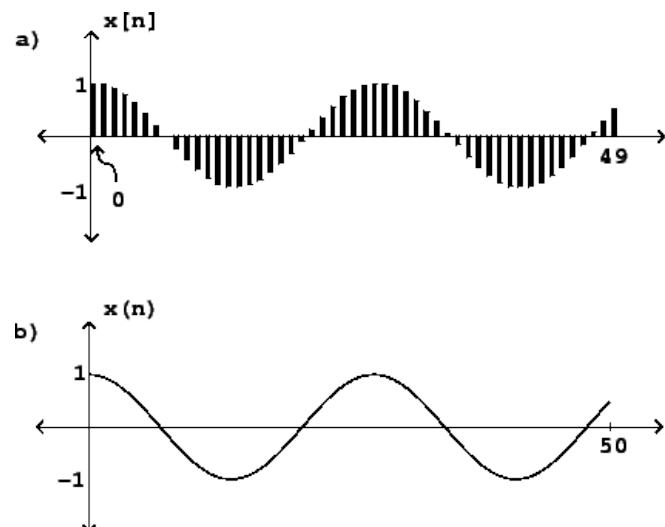
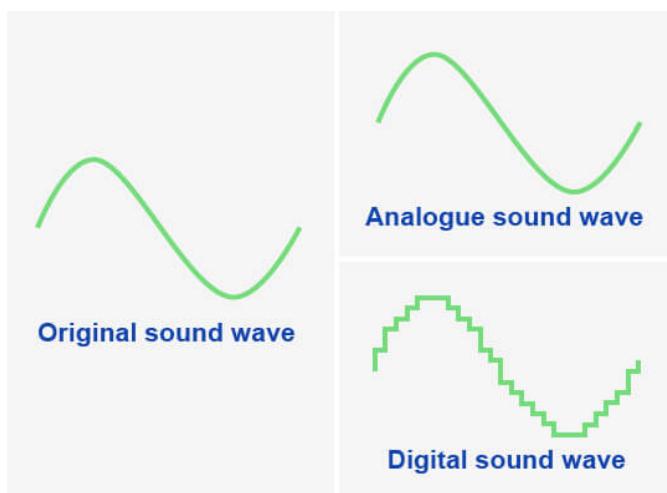
Lydbølger er trykbølger som forplanter sig igennem et materiale, ved at molekylerne sættes i bevægelse – f.eks. igennem luft - eller under vandet, igennem metal m.m. I det ydre rum er der dermed ingen lyd, fordi der ikke er noget materiale, den kan forplante sig i. Når i hører ekspllosioner i kampscener i rummet i en Star Wars-film svarer det altså ikke til virkeligheden. Når lydbølger opfanges af vores ører, sætter de trommehinden i svingninger, som bliver afkodet som et væld af forskellige lyde i vores hjerner. Herunder ser i en harmonisk svingning (sinustone), afbildet som en to-dimensionel bølge (waveform):



3. Digital Lyd og Lydsignaler

Hvad er Digital lyd?

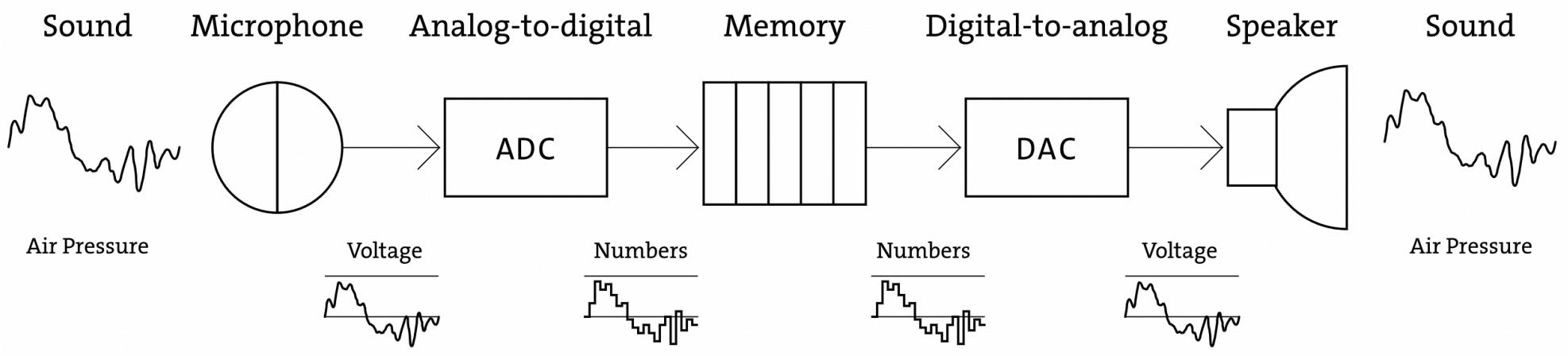
Arbejde med lydbølger på en computer kaldes *digital signal processing* (DSP). Digitale lydbølger på en computer omtales således som signaler. Disse signaler kan sammenlignes med digitale billeder, som er ”pixelerede” (dvs. består af et antal pixels som man kan se når man zoomer ind). På samme måde består et lydsignal af separate *samples** (talværdier mellem -1 og 1) – som man også kan se enkeltvis, hvis man zoomer nok ind på lydsignalet. Lydsignalets oplosning kalder man *samplerate*. I en computer arbejder man typisk med en samplerate på 44100 samples i sekundet, som svarer til oplosningen på en CD.



*) Bemærk at begrebet *sample* har dobbelt betydning: *sample* kan både referere til et enkelt trin i en digital waveform, som beskrevet her.

Sample kan også referere til en lydfil som kan afspilles i en sequencer, f.eks. trommesamples - eller som en lydeffekt. Dem gennemgår vi senere.

For at disse samples kan blive omdannet til et rigtigt (analogt) lydsignal, skal de gå igennem en såkaldt DAC (Digital to analog-converter) som omdanner det digitale signal til en elektrisk spænding. Herefter kan signalet sendes ud igennem en højtalер som "rigtig" lyd. I en mikrofon sker den omvendte process – dvs. at en rigtig lyd bliver omdannet til et elektrisk signal, og derefter via en ADC (Analog to digital-converter) til et digitalt signal i en computer.



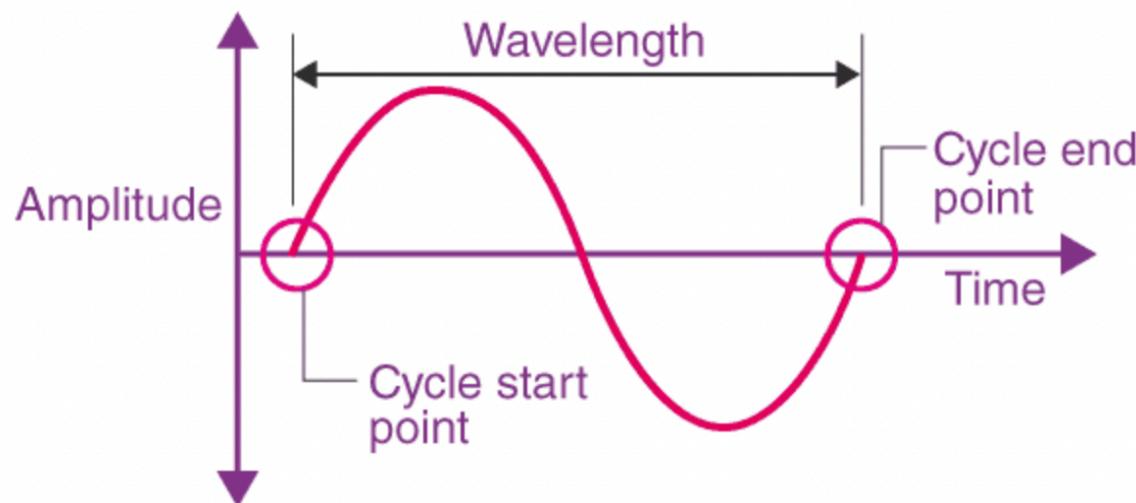
4. Bølgeformer (Waveforms)

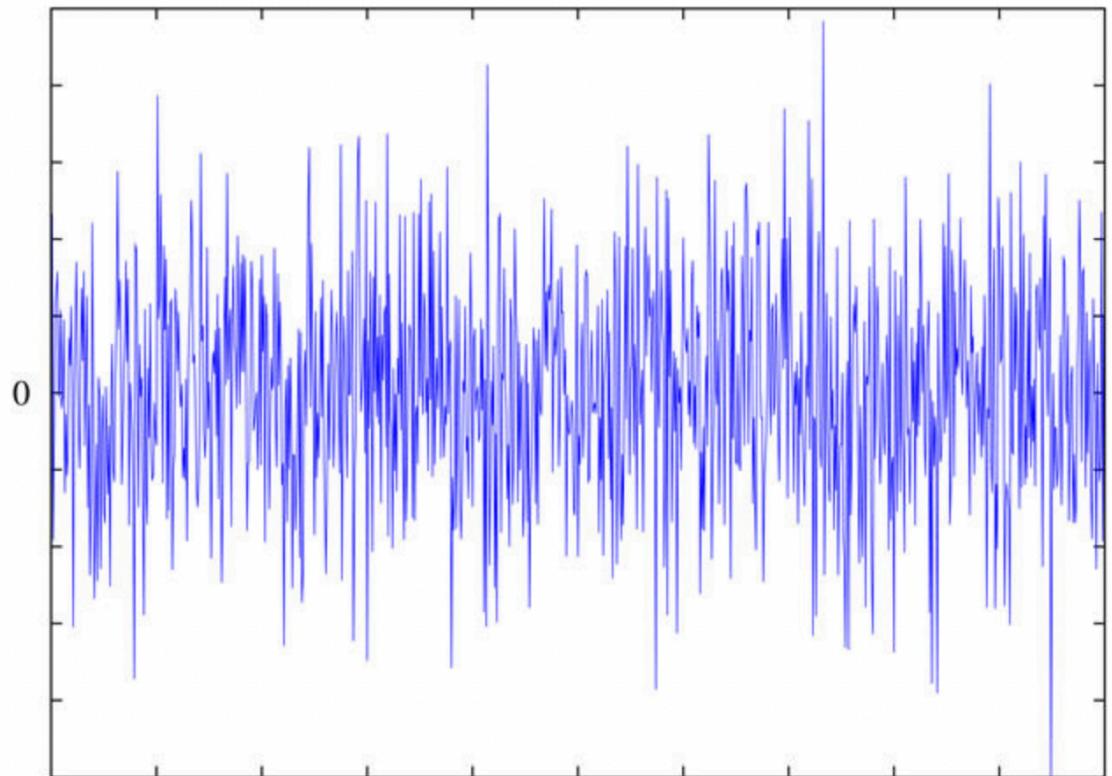
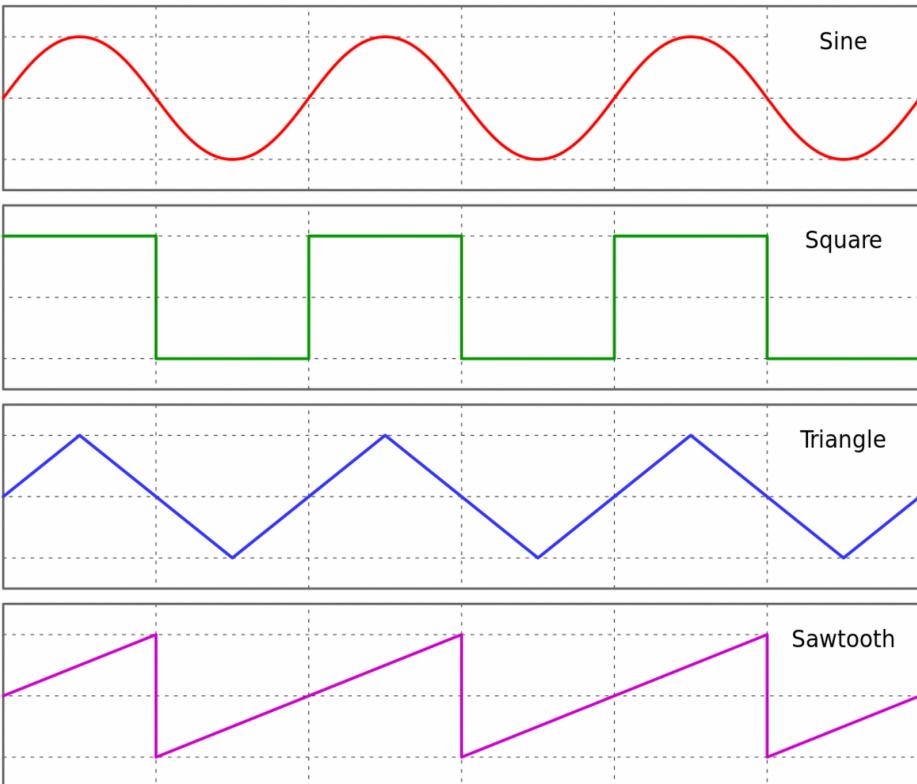
Bølgeformer

Der findes mange forskellige slags lydbølger. I musiksammenhæng skelner man oftest mellem harmoniske bølger (med en tydelig "tone"), og støj (uden tone). Hvid støj er et lydsignal der genereres som en kontinuerlig række samples med tilfældige værdier. Det lyder som vind eller susen.

Harmoniske bølger har til gengæld en række egenskaber: Bølgelængden (wavelength) afgør hvilken tonehøjde der er tale om. Jo langsommere bølgelængde, desto dybere tone, og desto hurtigere bølgelængde, desto højere tone. Bølgelængden angives som frekvens (cycles pr. second). Kammertonen (det midterste A på klaveret, som er dén instrumenterne stemmer efter i et symfoniorkester) har frekvensen 440 hertz.

Amplituden angiver højden på bølgetoppen, hvilket svarer til lydstyrke (volumen) af lydbølgen.





I elektronisk musik bruger man en række forskellige basale bølgeformer: Sinus-, Triangle-, Square- og Phasor-waves. Til højre ses en bølge med "hvid støj" (white noise).

Bemærk at disse basale bølgeformer er *syntetiske*, dvs. at de frembringes vha. en matematisk algoritme. I den virkelige verden er lyde langt mere avancerede og komplekse – og altid en blanding af utallige forskellige basale lydbølger som smelter sammen til de komplekse, sammensatte lyde vi kender fra naturen.

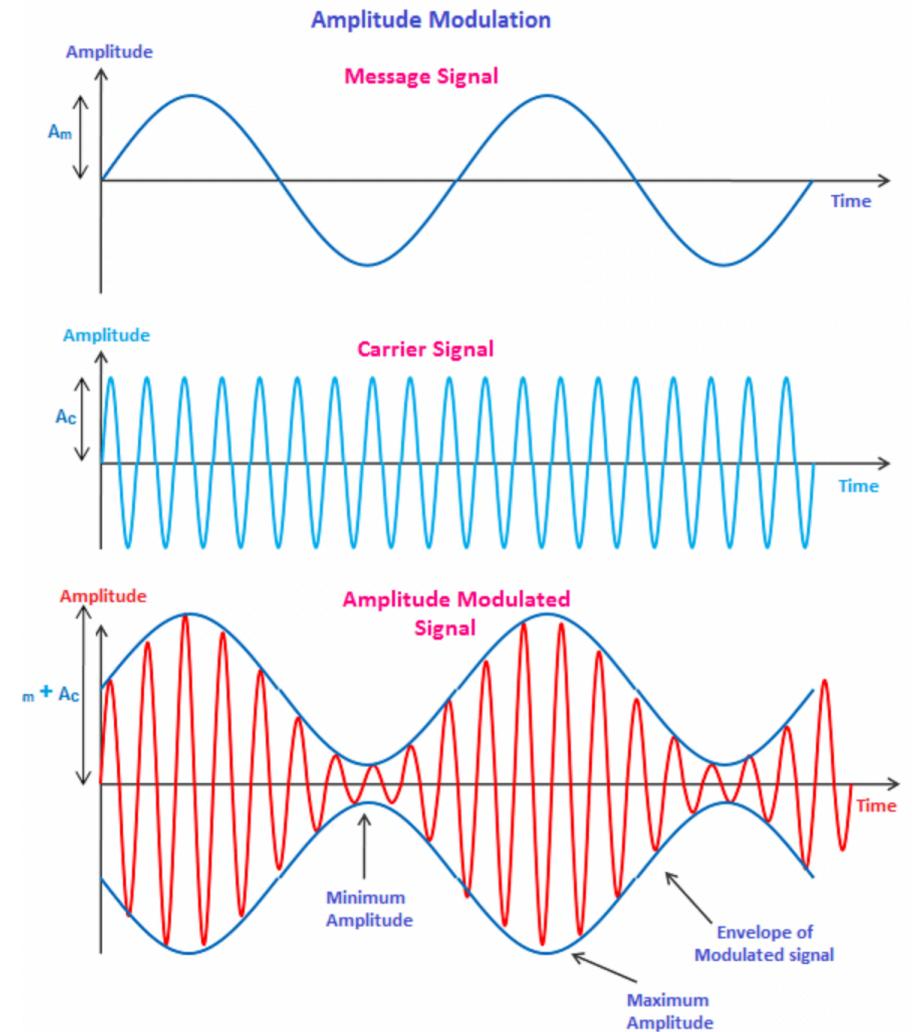
Man kan imidlertid efterligne naturlige lyde vha. af disse basale bølgeformer, når man lægger dem sammen, ganger dem med hinanden m.m. - og dermed efterligne lydfænomener i den virkelige verden via videnskabelige analyser og modeller.

5. Modulation

Modulation

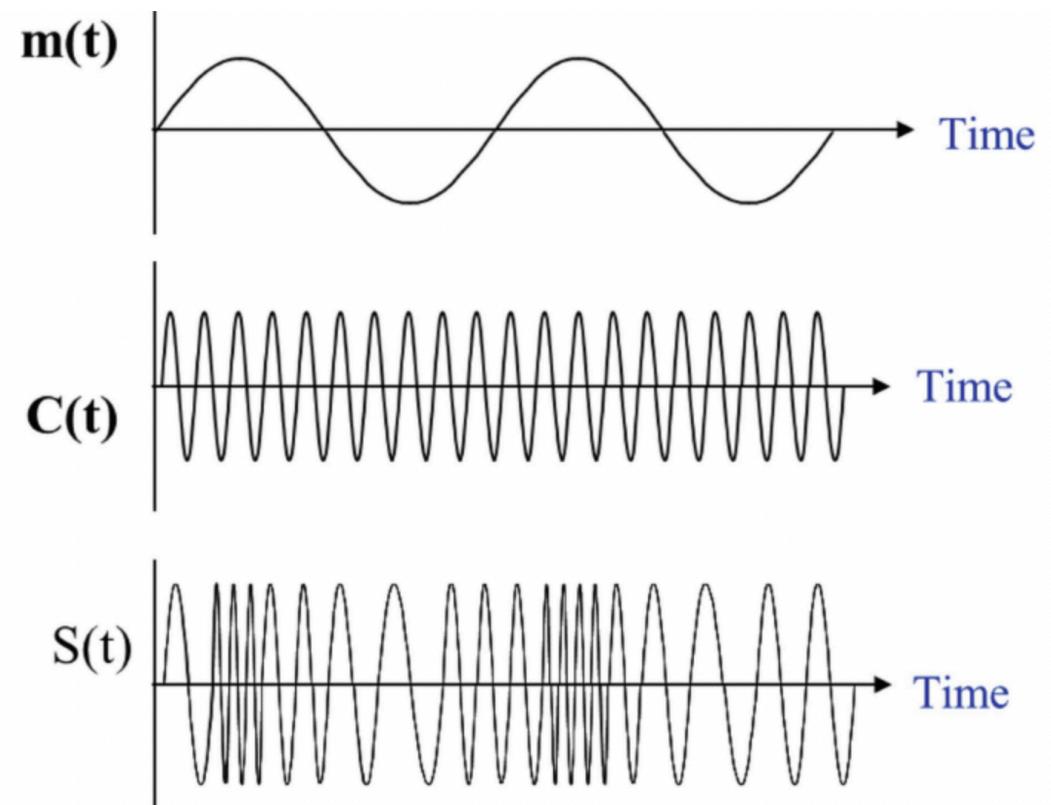
Modulation er, når man ganger (modulerer) 2 signaler med hinanden, således at man derved opnår et nyt signal.

I amplitude modulation (AM) ganger man amplituden af det ene signal med det andet. Dét kan sammenlignes med *tremolo*, (f.eks. i et hammond-orgel eller en guitar-pedal), der lyder som om man skruer hurtigt op og ned på volumen-knappen:



Frekvens-modulation (FM)

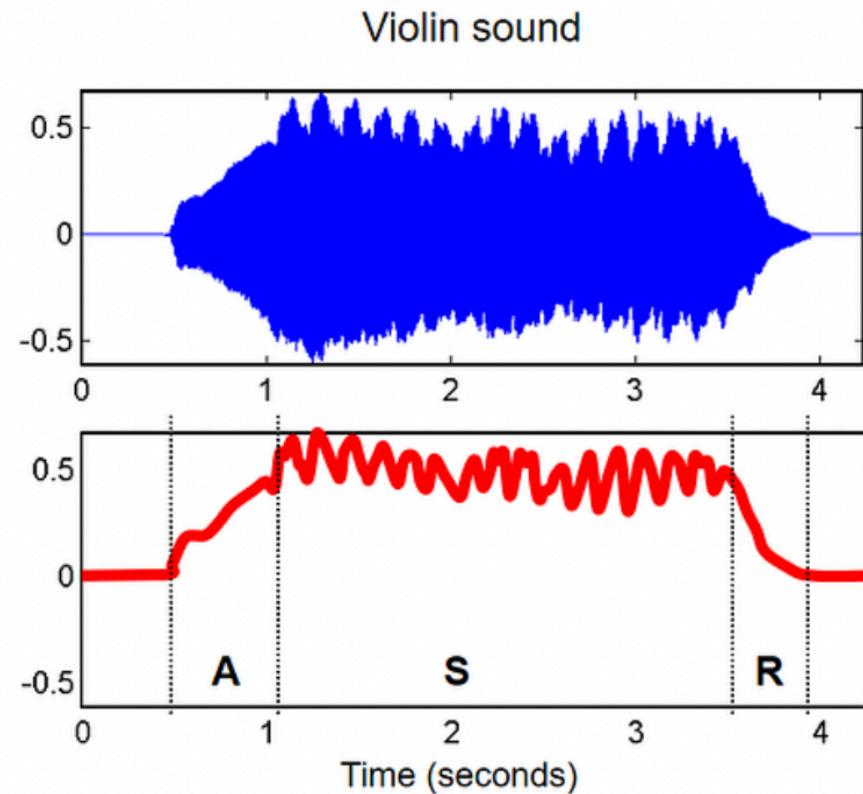
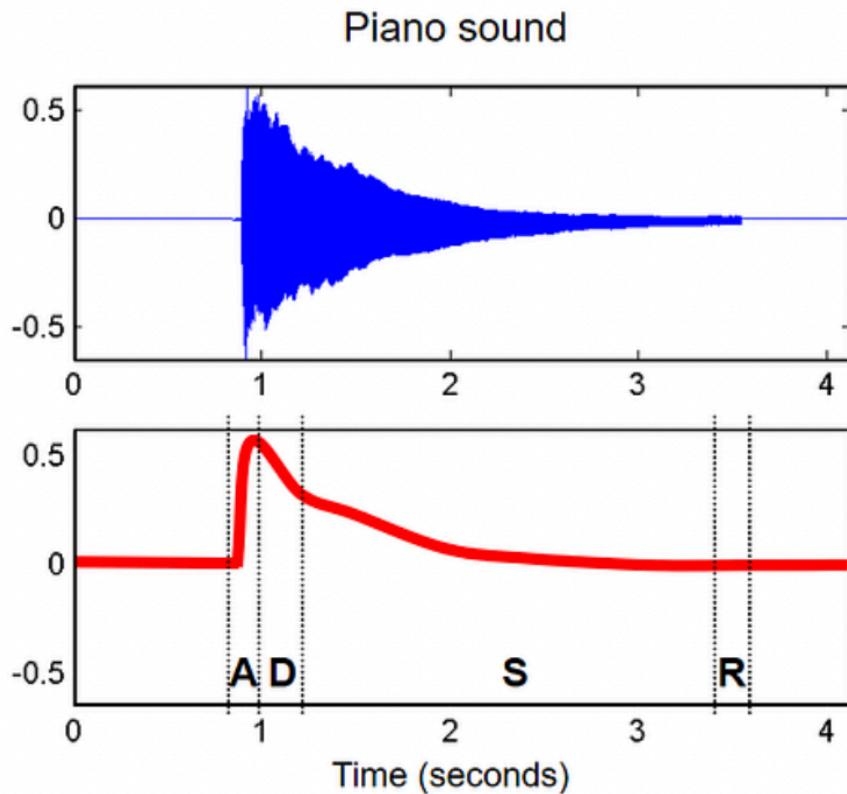
i frekvens-modulation (FM) ganger man frekvensen i det ene signal med det andet. Det kan sammenlignes med "vibrato" (f.eks. hos opera- eller soulsangere) hvor man ændrer tonen en lille smule op og ned:



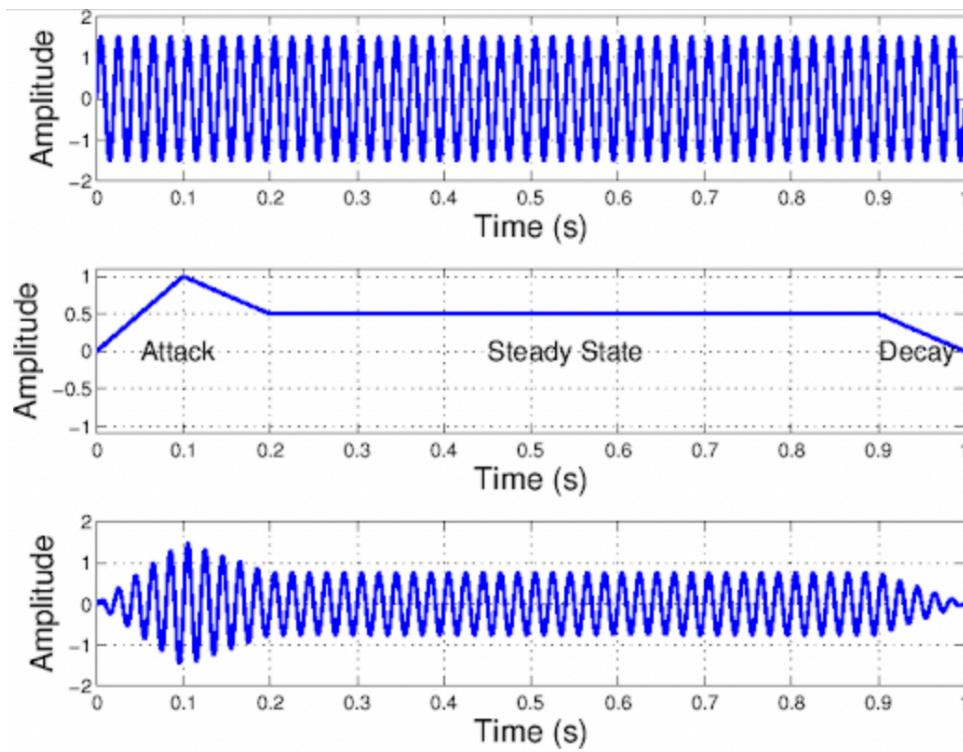
6. Envelopes

Envelopes

Man kan ændre et signal ved hjælp af en *envelope*. I lyd og musik er *envelope* et udtryk for hvordan en lyd ændrer sig over tid. Dét er afbildet med den røde graf herunder i forbindelse med en klaver- og en violin-tone:



I elektronisk musik bruger man envelopes til at forme et kontinuerligt lydsignal, så det får karakter af et "rigtigt" instrument. Man kan bruge forskellige slags envelopes til at give lyden forskellig karakter. Hvis man vil have noget der minder om en klaver-tone bruger man en envelope med et hurtigt, kraftigt *attack* i starten, som derpå *fader* hurtigt ud. Hvorimod en lyd der minder om en violin-tone *fader* mere gradvist ind og bliver på mere eller mindre samme lydstyrke indtil den slutter – ligesom i eksemplerne på forrige slide.
Se her hvordan en sinus-tone formes vha. en ADSR-envelope (attack-decay-sustain-release):



En ADSR-envelope er et helt centralet koncept i synthesizere. Man taler sågar om at ADSR-envelopen er synthesizerens *sjæl*.

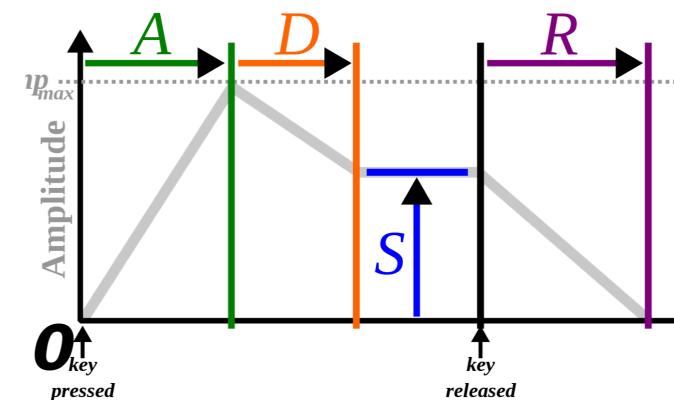
ADSR er en envelope, hvor man kan indstille lydens *attack*, *decay* og *sustain*, som triggies når man trykker på en tangent på sit keyboard:

Attack angiver hvor hurtigt amplituden skrues op fra 0 til max.

Decay angiver hvor hurtigt amplituden derefter skrues ned til et moderat niveau (steady state)

Sustain angiver amplituden på dette moderate niveau.

Og når man slipper tangenten på sit keyboard, angiver *Release* hastigheden hvormed lyden udtoner til 0 (fader ud).



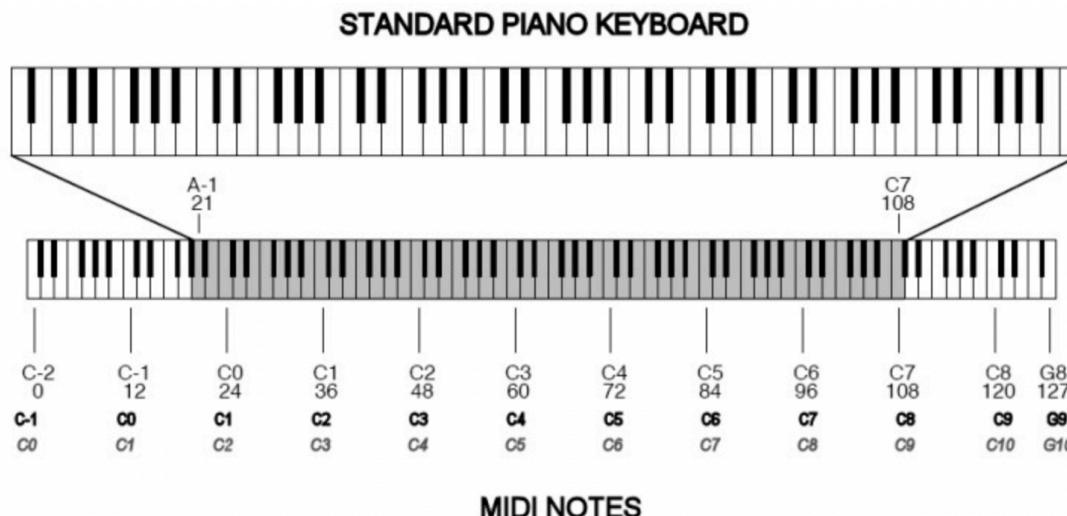
7. Midi

Midi

Midi (Musical Instrument Digital Interface) er det sprog, der gør det muligt at sende data frem og tilbage mellem et fysisk instrument og computeren. Når vi har kodet en synthesizer i Pure Data, kan vi således spille på den med et keyboard via Midi:

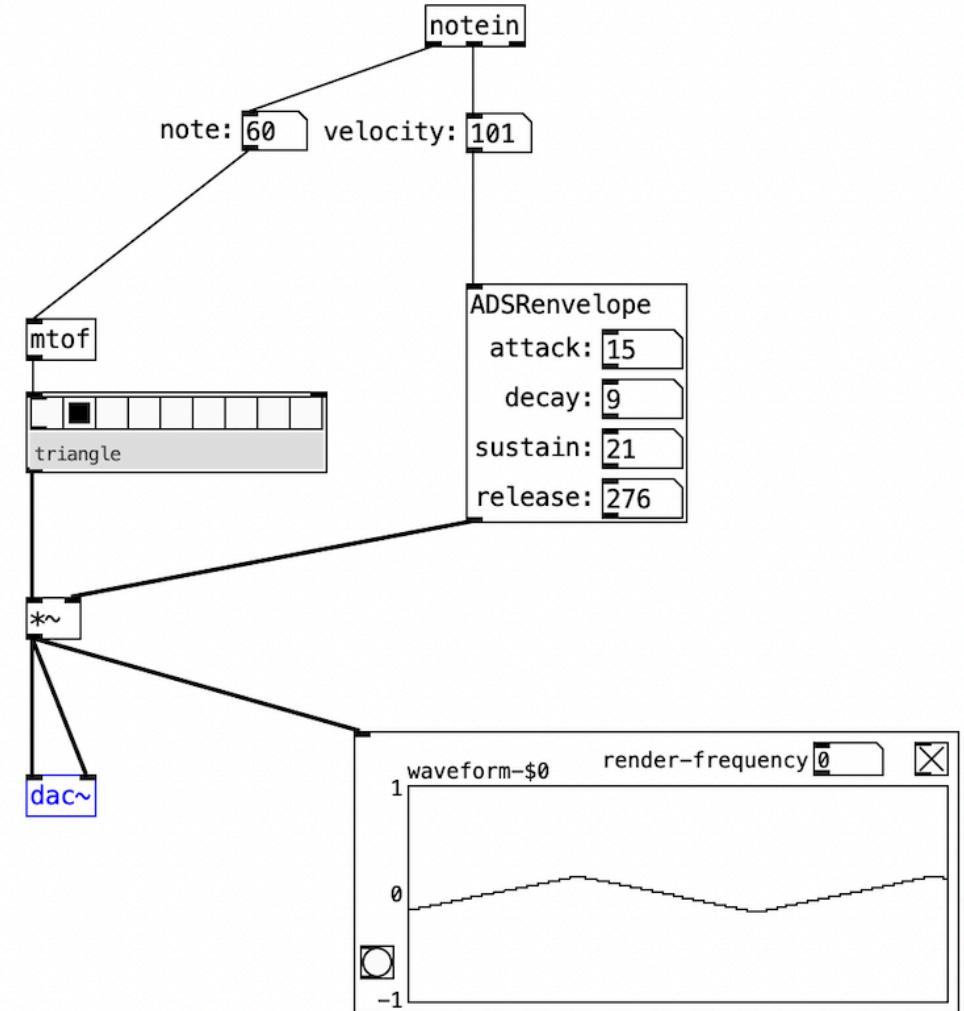
Når vi trykker på en tangent på keyboardet, modtager computeren en række midi-data, som man kan programmere til at styre forskellige parametre i sin synthesizer. De vigtigste midi-data er:

- Note On (fortæller at tangenten er trykket ned)
- Note Number (fortæller hvilken tangent der er blevet trykket ned)
- Velocity (anslag - fortæller hvor hårdt tangenten er blevet trykket ned)
- Note Off (sendes når vi slipper tangenten, og som fortæller at tonen skal slutte)



Typisk vil man *mappe* note-number til frequency (tonehøjden) via et mtof-objekt i sin synthesizer, og velocity (anslag) til amplitude (volumen) - se eksemplet til højre.

Men det behøver ikke nødvendigvis at være sådan: man kan også bytte om på parametrene, hvis man gerne vil have en synthesizer, der er mærkelig at spille på - så anslaget styrer tonehøjden eller noget helt tredje. Det er op til musikeren når han/hun designer sin egen synthesizer.

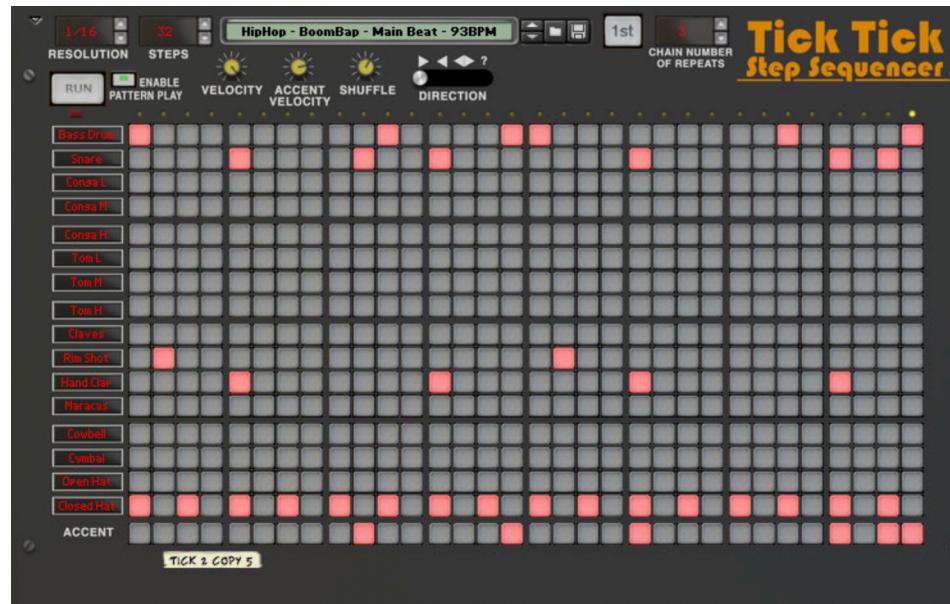


8. Step-sequencer

Step-sequencer

Udover at man kan tilslutte et midi-keyboard til sin synthesizer og spille på den som et konventionelt instrument, så er det særlige ved elektronisk musik, at man også kan få computeren/teknologien til selv at spille med, og dermed udgøre et helt lille orkester, som man kan styre vha. automatisering af forskellige musikalske processer.

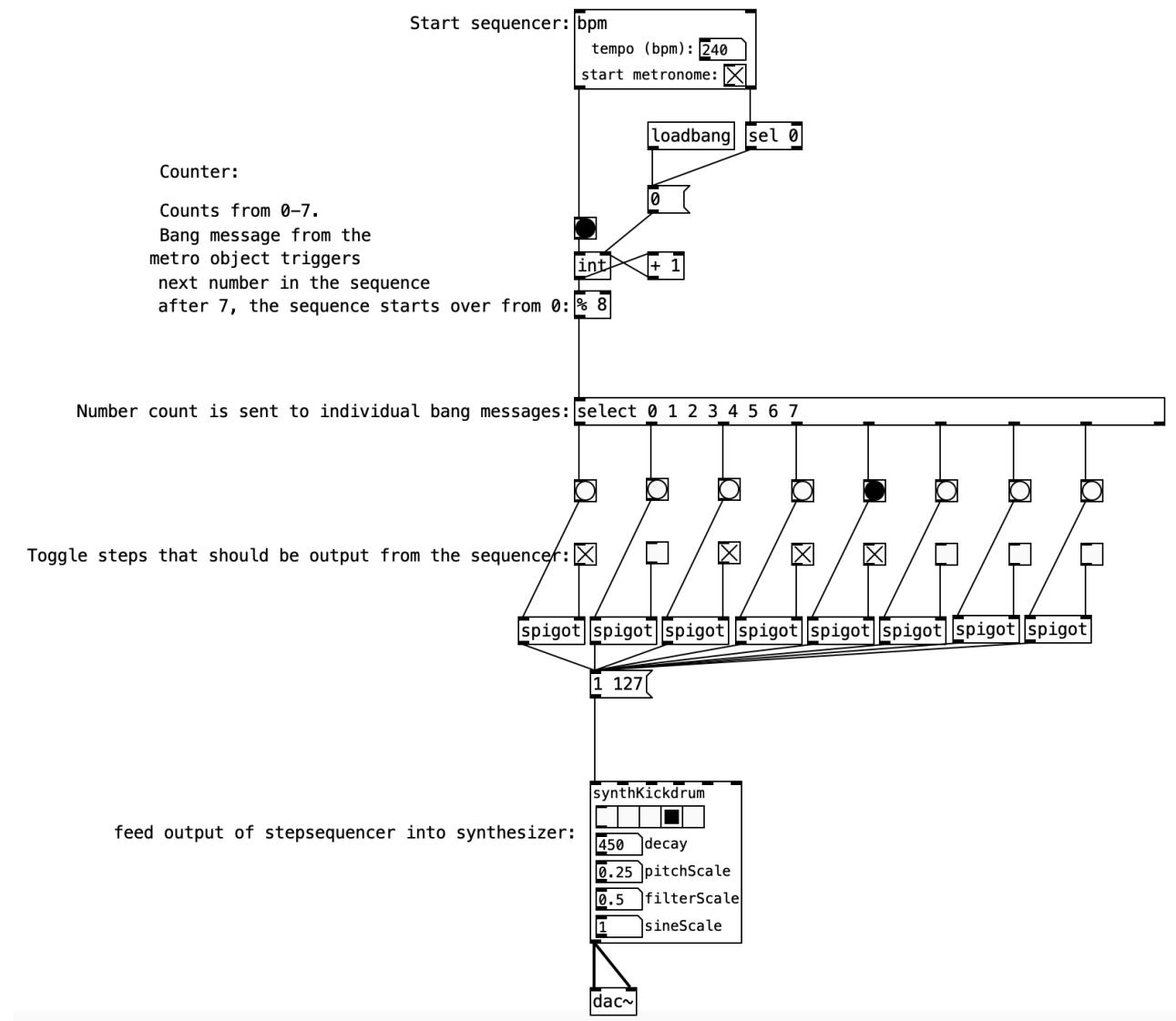
Et helt centrale stykke værktøj, eller instrument (om man vil) er den såkaldte stepsequencer, som man kan bruge til at programmere rytmer.



En step sequencer består dels af en tæller (counter) der genererer en sekvens af numre - svarende til tælleslag (taktslag) som i kender hvis i har spillet på et traditionelt musikinstrument.

Disse tælleslag afbides med et grafisk brugerinterface, så man kan se hvordan der loopes fra 1-8. Herefter kan man så vælge (ved at klikke i en toggle-box) hvilke tælleslag, der skal sende en impuls videre fra step-sequenceren. Disse impulser kan man herefter tilslutte en synthesizer eller en sampler.

Se en implementation af en step sequencer i PD her:



9. Bibliografi

Hvis i gerne vil dykke videre ned i synthesizer-programmering og Pure Data på egen hånd, er der mange glimrende ressourcer til rådighed. Der findes adskillige youtube-kanaler med tutorials i PD-programmering. Ligesom en række glimrende bøger og online tutorials, hvoraf jeg lister nogen af de bedste (inkl. links) her.

Links'ene er listet efter specialisering, med første link som den mest basic og let tilgængelige introduktion til PD. Herefter bliver de mere og mere specialiserede, med sidste link som den mest tekniske og detaljerede indføring i både teori og praksis.

Bibliografi:

Johannes Kreisler: Loadbang

<http://www.pd-tutorial.com/english/index.html>

PD Floss Tutorials (Online tutorials)

<http://archive.flossmanuals.net/pure-data/index.html#>

Francesco Bianchi, Cipriani Alessandro, Giri Maurizio:

Pure Data, Electronic Music and Sound Design - Theory and Practice - Volume 1

<https://www.amazon.com/Pure-Data-Electronic-Design-Practice/dp/889921221X>
kan lånes på [bibliotek.dk](#)

Andy Farnell: Designing Sound

<https://mitpress.mit.edu/9780262014410/>
kan lånes på [bibliotek.dk](#)

Miller Puckette: The Theory and Technique of Electronic Music

<http://msp.ucsd.edu/techniques/latest/book-html/>