Composition 002

Andy Farnell

14th June 2007

1 Synopsis

Introduction This is the third tutorial in a series on composing electronic music with Puredata. Composition000 and Composition001 introduced simple DSP and message sequencing. Here we consolidate that knowledge with a study no more difficult than the last part, showing the development of a 'real' piece of music. This might serve as a good enough template, or inspiration for documenting you own Pd projects, breaking down each abstraction and explaining your code comments with more detailed text alongside the diagrams and sound captures. Skip this part if you are focusing on MIDI composition and go to the next appropriate tutorial.

Aims To complete a piece of music with at least six parts of diverse timbre. Use GUI objects to make a mixer and reuse abstractions. Explore random choices, selection and sequence in regard to making a music score. Substitute parameters into lists to make adaptations to a synthesiser. Create some delay and filter effects.

Methods We will add parts in compositional order, as I did when making the original piece. Starting with some chords and a pad sound we establish the harmonic structure and pace. A bass part and drums fix the beat and root, then we add some melody and arpeggio instruments. Each part is built in its own abstraction and some parts are included many times. The sequencer, synth, and any effects will be built each time before listening to the result.

New units used

- `[vd~]`: Variable delay.
- `[spigot]` A tap or valve for messages.
- `[del]` Delay a bang message.
- `[sig~]` Turn a message into a signal.
- `[vcf~]` Better variable bandpass filter.
- `[delread~]` Read from a named delay buffer.
- `[delwrite~]` Write to a named delay buffer.
- `[clip~]` Audio clip, set max and min signal range.

2 transport-controls

2.1 transport controls

Figure 1 summary

- default tempo 240 bpm
- send out global data

start and stop Inlet `[inlet start-stop]` receives a Boolean value, 1 to start the metro in the masterclock and 0 to stop it. Each time this toggles the default period is broadcast and the clock period refreshed. A toggle input box on the transport GUI sends this value. Also on the transport GUI is a bang button which is received by `[r zero]` to set the time counter back to 0.

output values Four different messages are continuously sent by the masterclock, `[s trig]` emits bangs on every single beat of the metro, `[s scoretime]` sends a float representing the time for the conductor of the score, `[s drumtime]` is always a beat ahead of the scoretime `[s synthtime]` moves at half the regular rate. We don't use all these time values in this composition but this is a useful way to output time values, so that score changes can be processed one step ahead of the main timeline. A1-transport-controls.pd

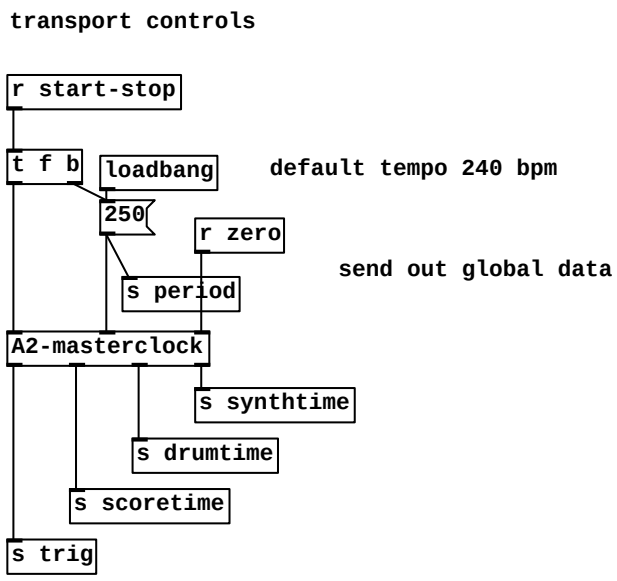

Figure 1: A1-transport-controls

2.2 timebase

Figure 2 summary

- start now
- define order
- counter
- synth is half speed
- synth first in order
- drums next in order
- scoretime next in order
- send bangs

metronome The metro is always started by default here, you can remove the loadbang if you don't want the piece to start automatically. The metro period is set from the middle inlet and if anything is received on the right inlet the counter is set to zero.

late and early beats You can see how time is made a beat earlier by taking its value from after the `[+ 1]` instead of from the float box. You could chain these or add arbitrary increments or decrements to the timeline. If you want a part to play 4 beats ahead add `[+ 4]` before its time inlet. Beare of negative times that won't exist when the timebase is zero.

dividing time Dividing time is as simple as using `[/]`. But this raises interesting points to do with number lines and quantisation. A timeline that is multiplied by an amount is stretched or shrunk like a rubber band. If you are using select statements that work on integer beat marks then you need to requantise with `[int]` and `[change]`. If a timeline is doubled then it loses half its resolution, but if it is squashed to half then its resolution would double, if it were say audio tape, however it halves. If we had an integer timeline $1, 2, 3...$ and divide by 2 we get $0.50, 1, 1.50, 2...$, some of the values have become non-integer. Although we can make any floating point number that is a function of a periodically updated timebase it is sampled at the rate of the timbase itself, which is when message events occur. These two things, the value and the time it is sent, are separate things. Using a rounding process two notes must now fall into each event slot (pigeonhole principle). Going the other way let's take a timeline $1, 2, 3, 4$ and multiply it by 3. We get $3, 6, 9, 12$ You can build up sophisticated rythmns using `[mod]`, `[int]`, `[change]` and `[div]`, but it's recommended that if you study this be aware it depends on the implementation of `[int]`, there are other ways of rounding numbers that will break your compositions if you translate them or the definition of `[int]` changes. A2-masterclock.pd

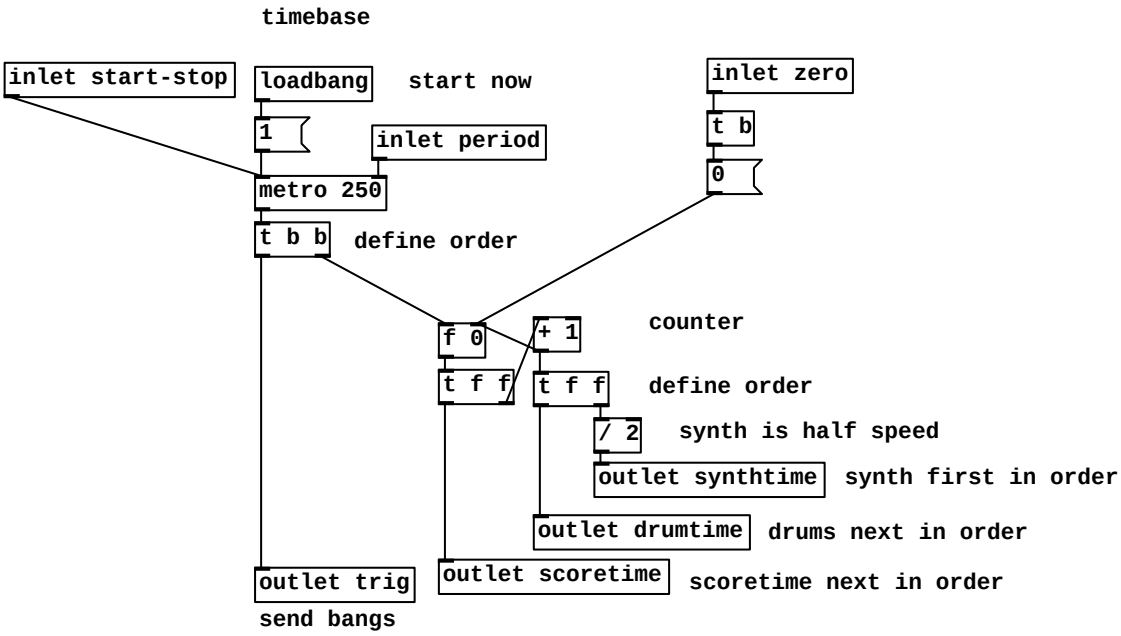

Figure 2: A2-masterclock

3 mixer-master

3.1 mixer master

Figure 3 summary

- all channels

only one master This is a simple part shown only for completeness. It is a wrapper for the mixer main output `[mixer-output]` that implements a mute. Normal volume control floats are received on `[r mast]` and passed through a multiplier. The second inlet `[r mastmute]` places a one or zero on the lefthand side of the multiplier and refreshes the value. When this value is zero the volume is zero, when it is 1 the volume is restored to its last level set on the right side of the multiply. For each fader a toggle on the main GUI acts as an on/off or channel mute. B1-mixer-master.pd

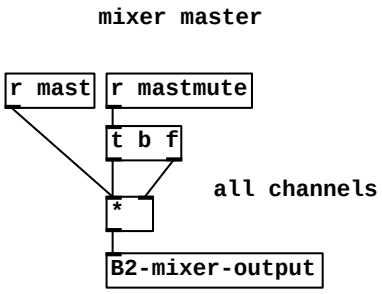

Figure 3: B1-mixer-master

3.2 mixer output

Figure 4 summary

- main left and right outs

catch main bus `[catch~ left]` and `[catch~ right]` pick up from two globally visible main signals "left" and "right" which are driven by `[throw~]` objects. Here both are multiplied by a simple line with 20ms slew for a fast but slightly less clicky fade action. Outputs for the `[dac~]` are specified explicitly, but you don't have to do this unless working with multi channel audio. B2-mixer-output.pd

3.3 mixer channel

Figure 5 summary

- volume and pan

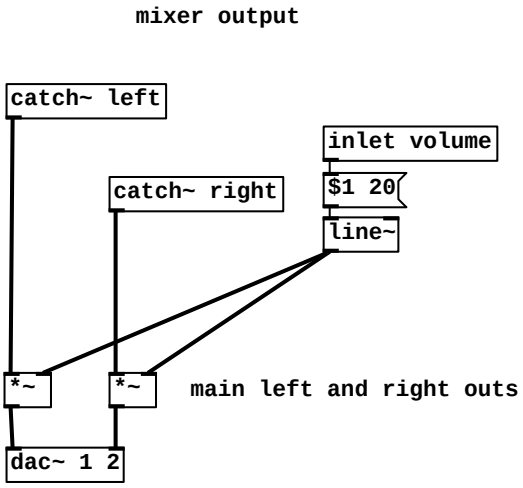

Figure 4: B2-mixer-output

- throw to left and right bus

many channels This abstraction is repeated many times in the composition, it's a mixer channel that sends to the main bus using `[throw~]`. It has it's own `[line~]` unit which is controlled via `[inlet volume]` and `[inlet pan]` which carries a value in degrees from -90 to +90. In this case we have made an unusual compromise between mono and stereo sources by having a stereo signal panned midfield at a width of 60 degrees, which leaves it 120 degrees of movement. B3-mixer-channel.pd

3.4 simple panner

Figure 6 summary

- pan in degrees -90 ¿¡ +90
- right = 1 - left

opposite volume for left and right The value on `[inlet pan]` is first rescaled from degrees to a normalised range. A proper pan uses cosine fades for equal power panning, but this panner is easier to understand and fine for most things. The left and right volume control values are opposites of each other so when the left pan signal is 1 the right is 0. B4-pan.pd

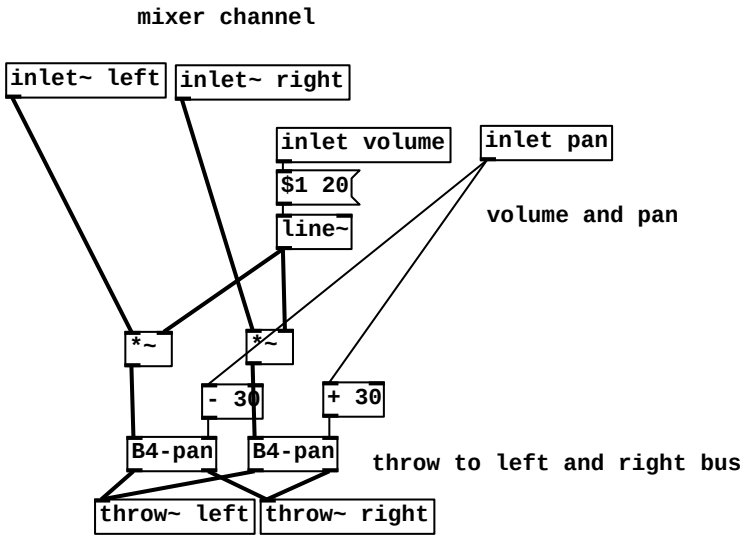

Figure 5: B3-mixer-channel

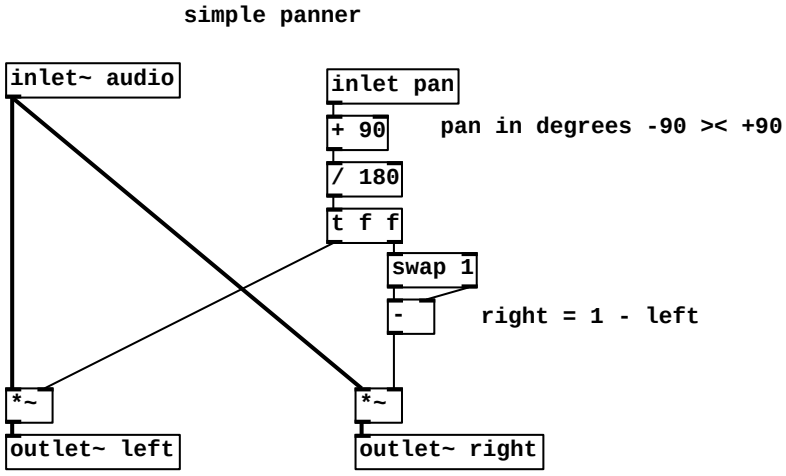

Figure 6: B4-pan

4 score

4.1 score

Figure 7 summary

- simple progression
- chords sent to everything

score messages The entire piece repeats every 128 beats of scoretime. There are changes on the first, 33rd and 65th beats which broadcast one of three chord lists on the global channel `[s chords]`. C1-score.pd

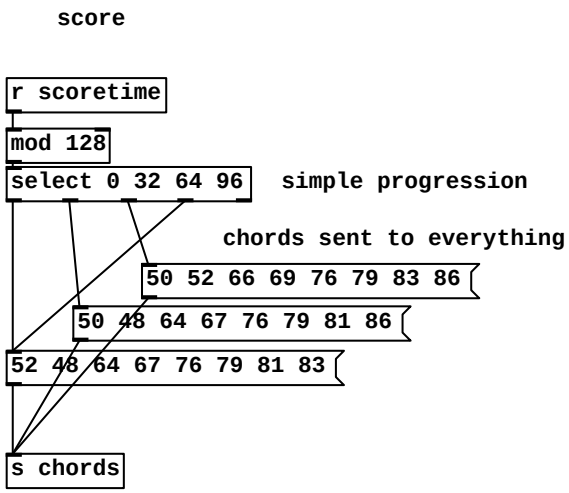

Figure 7: C1-score

5 stereo-pads

5.1 stereo pads

Figure 8 summary

- wrapper for synth and channel
- receive global chord list
- play it all as pad

First synthesiser wrapper This patch is typical of a wrapper. It is mainly a way to tuck two or three related things in a box together. We have a synth part which is going to play some audio and all the global messages needed to make that work, in this case only `[r chords]`, and we have a mixer channel with a few extras to implement a channel mute. G1-stereo-pads.pd

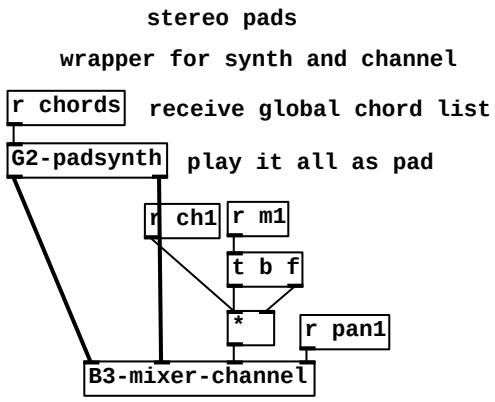

Figure 8: G1-stereo-pads

5.2 pad synth

Figure 9 summary

- unpack 8 notes
- 2 x oscillator bank
- panning and effects

synth gubbins All the notes from the chord are unpacked and sent to separate oscillators. A `[phasorblock]` each contains 4 phasors so we have one half of the chord going to the left side, and the other to the right side of a stereo chorus/delay. The amplitude decay time of the synth, which is processed by `[stereoamp]`, is set to 4 times the current period, so it will move automatically if there is any tempo change. Locking synth parameters to global variables like tempo is very useful, you may have many lfos all running in synchrony if you use a fast timebase by running the metro at 16 times the drum speed to broadcast a `time` value from which message rate lfos can be derived. G2-padsynth.pd

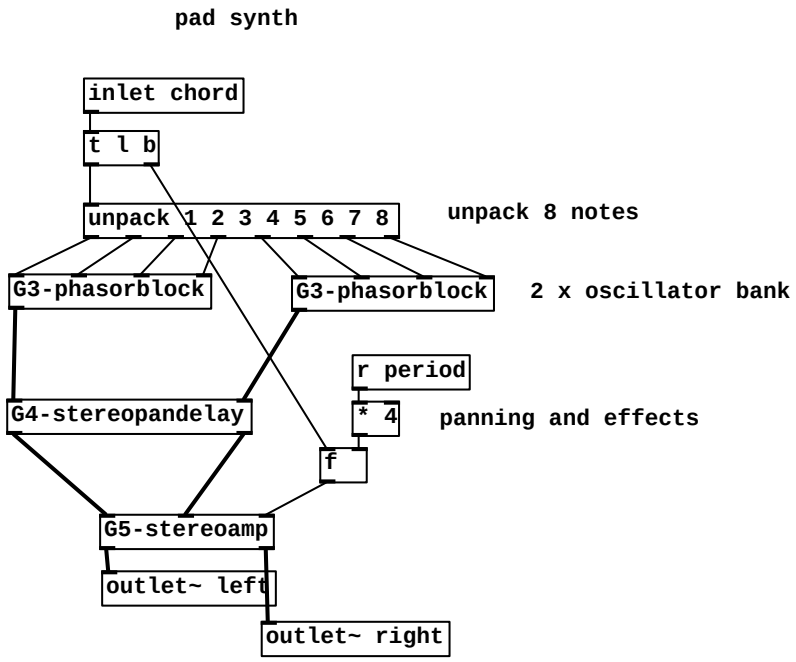

Figure 9: G2-padsynth

5.3 oscillator bank

Figure 10 summary

- 4 x phasor
- separate [mtof]
- remove dc offset
- scale quite low

phasor bank Bunching up a load of osc sources in one signal abstraction is quite typical but with phasors we need to remember to remove the DC offset

because a phasor is not symmetrical about zero. There are clever ways to reuse a `[mtof]` if we have lists of notes, but it is a very inexpensive unit so we opt to have separate inlets and separate `[mtof]` for each voice. G3-phasorblock.pd

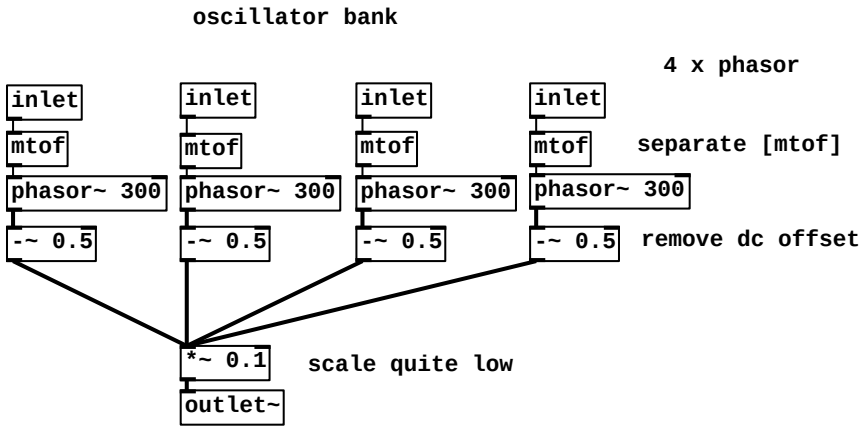

Figure 10: G3-phasorblock

5.4 stereo pan delay

Figure 11 summary

- slowly moving random line
- variable delay for pitch shifting

tones for composing with When setting the mood for a piece by choosing some chords it often helps to use "big" sounds. Rich sounds in certain timbres suggest things harmonically. Pure sine waves don't give the potential for suggesting harmonically related chords because they have no overtones of their own. A way to get a richer sound from fairly weak sawtooth waves is to employ a chorus or vari-delay effect.

deep swirly effect Instead of a regular lfo that sweeps back and forth over a fixed period we have a wandering control signal. This is used to sweep two bandpass filters with a slightly less bright version going to the delay. Delay unit `[vd~ $0-ffff]` is a vari-rate delay with its time in the range 10 ms to 50 ms and swept by the same wandering signal. This results is a rather old fashioned, warped kind of effect with occasional fast excursion that make the sound go out of tune. G4-stereopandelay.pd

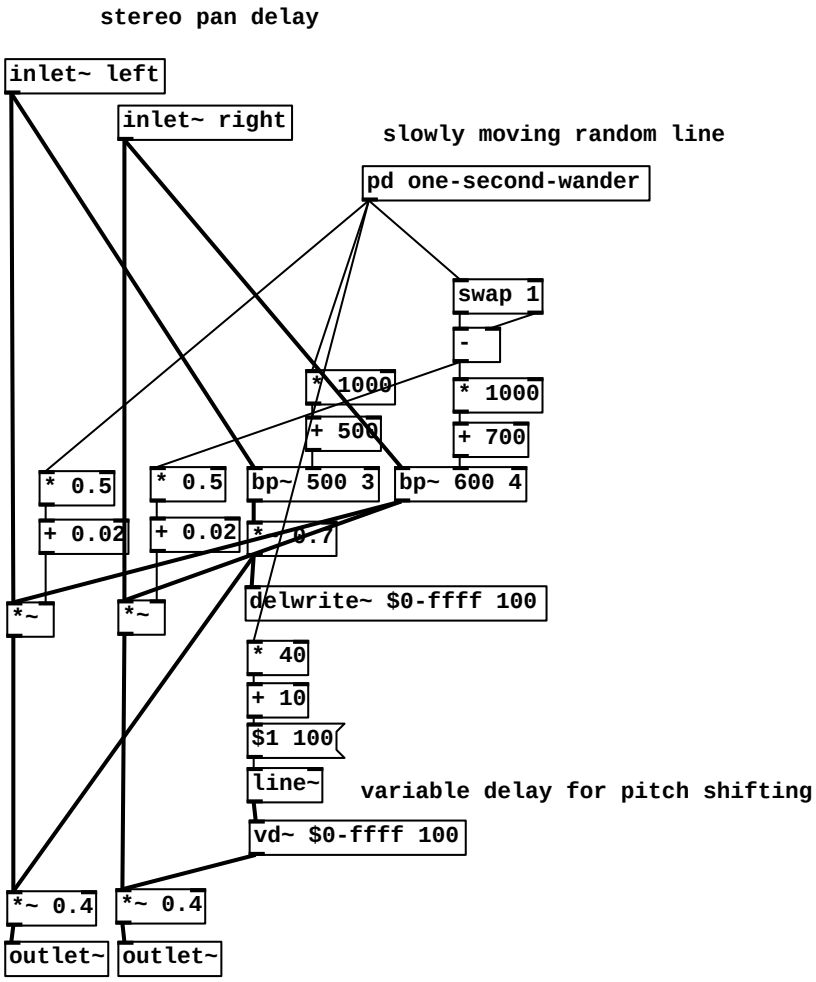

Figure 11: G4-stereopandelay

5.5 stereo amp

Figure 12 summary

- long envelope
- channels linked

long amplitude envelope To control both left and right outputs of the chorus together we need a stereo envelope. The control signal holds for about twice the decay time, which in this case is 16 times the period. Look back to the `[padsynth]` to see we used 4 times the period. This gives a soft fall to the pads so they don't play in a constant dirge, slightly more string-like than organ. G5-stereoamp.pd

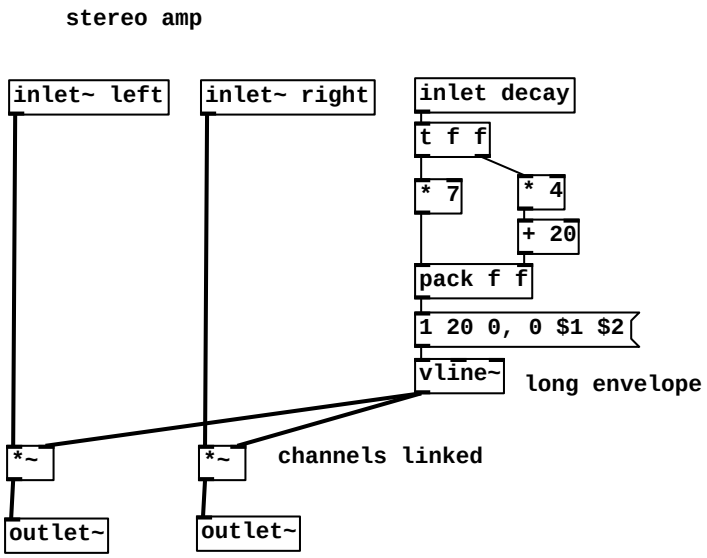

Figure 12: G5-stereoamp

5.6 play the pad line

Figure 13 summary

sound of the pad synth Only one fader is up on the GUI, panned to the middle. There are no other parts in the piece at present. Experiment with panning and fading the part. Test out the on/off (unmute buttons) and resetting the timebase. Try putting different notes in the score chord lists. G6-pad-line-play.pd G6-pad-line-play.ogg

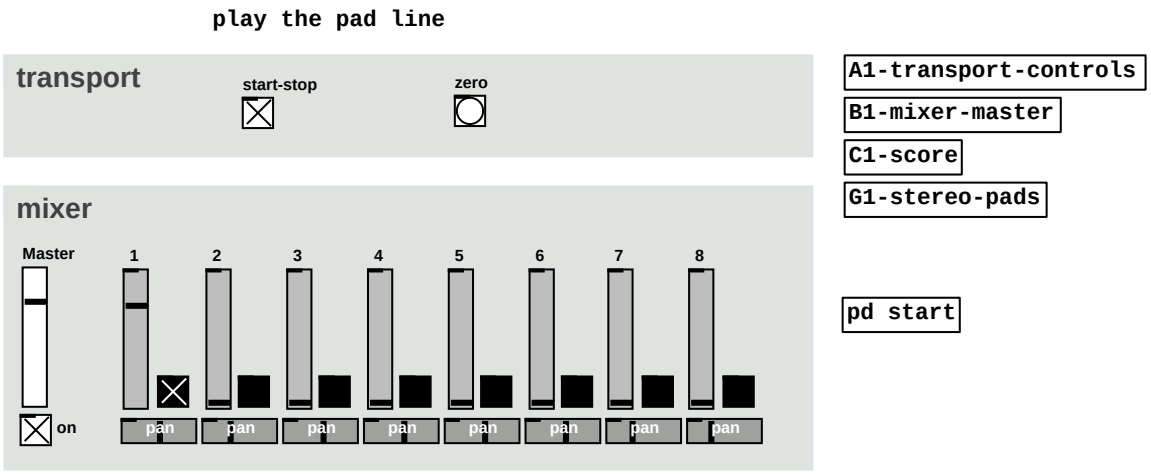

Figure 13: G6-pad-line

6 drums

6.1 drum part

Figure 14 summary

- separate kick and snare synths
- separate kick and snare sequencers
- panning and volume stuff

mixer channel As usual there is a mixer channel at the bottom of this wrapper. There are two instruments in here, a kick and snare drum, one panned to the left and one to the right. The channel is controlled by fader 2 and pan 2.

parts Five abstractions make up the patch, a sequencer for the kick drum pattern and a kick drum synth, and the same for the snare drum plus an effects patch. H1-drums.pd

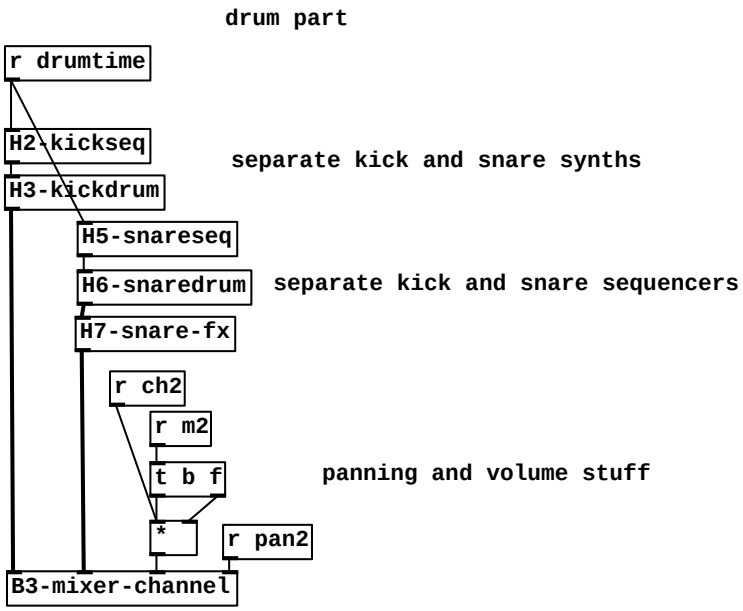

Figure 14: H1-drums

6.2 kick sequence

Figure 15 summary

- 16 beats to a bar
- 2 part sequence
- add in occasional

two select units When placing beats you don't have to use one `[select]` object and insert each beat in the right order, that would be most frustrating, instead its okay to just add more `[select]` operations for new beats as needed. Here the first and second bar are handled by different `[select]` units.

skip some beats Beats 2, 7 and 10 do not play every time. A `[spigot]` is like a valve, it lets its input through to the output whenever a 1 message appears at its right inlet, and continues to do so until it is sent a 0 message. The messages on the left inlet are then blocked and do not appear at the output. Here a `[random 2]` unit in combination with a spigot permits one in every two beats to pass through.

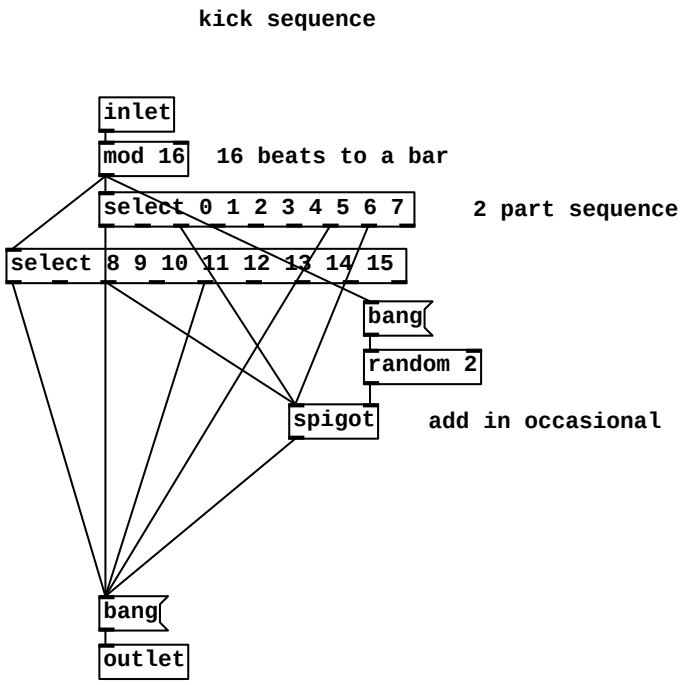

Figure 15: H2-kickseq

H2-kickseq.pd

6.3 kick drum wrapper

Figure 16 summary

- choose one of three
- some with shorter decay

three kick sounds This wrapper modifies the performance of the kick drum synth by storing three preset sounds, long and low, medium, shorter and higher. On each received beat one is selected at random, including a rest where no kick plays at all. Note how this behaviour is built in to the instrument now, not the sequence as we have defined it. Combined with a measure of uncertainty for individual beats in the previous patch the result is a much sparser pattern than you might expect.

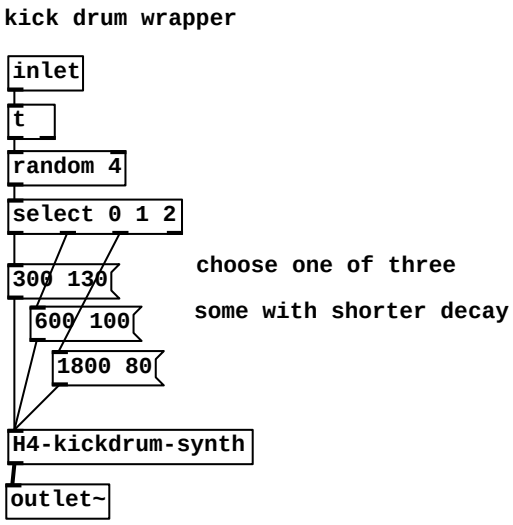

Figure 16: H3-kickdrum

H3-kickdrum.pd

6.4 kick drum synth

Figure 17 summary

- short line decay
- 4th power decay
- base 20 and sweep 100
- affects amp and pitch

parameters Two parameters are passed for each kick sound, a decay time and a pitch range. This list is unpacked first from the list output of `[t b b l]` with the first decay value going to the inlet of a `[pack]` object and the second going to the set the frequency sweep scale.

envelope A more complicated arrangement of messages for this envelope shows the simplicity, and limitation, of `[line~]`. In order to have a non-zero attack we need to hold off the decay message for at least as long as the attack

rise. This is done with a message delay `[del 2]` giving a 2ms delay to the bang message.

signals The signal is an sine wave with amplitude and frequency as functions of the fourth power of the falling line segment. Frequency is down to a minimum of 20Hz, and as high as the second parameter passed. Amplitude is always from full to zero, decaying in the time set by the first parameter. H4-kickdrum-synth.pd

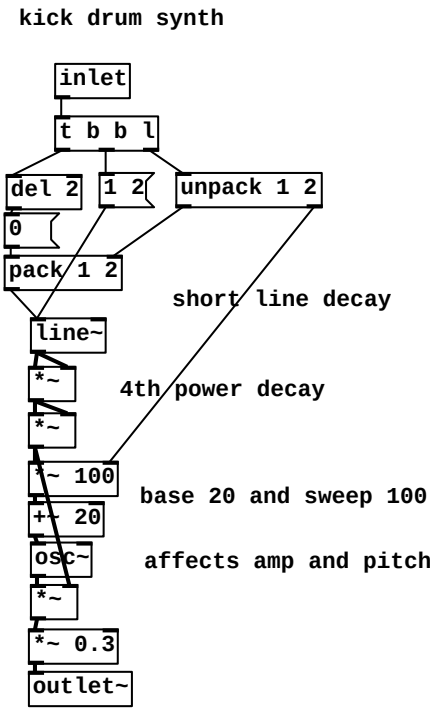

Figure 17: H4-kickdrum-synth

6.5 snare pattern

Figure 18 summary

- same pattern repeated

simple 7 beats Couldn't be much simpler this one, it plays beats 2, 3, 4, 5, 10, 11 and 12. H5-snareseq.pd

6.6 snare wrapper

Figure 19 summary

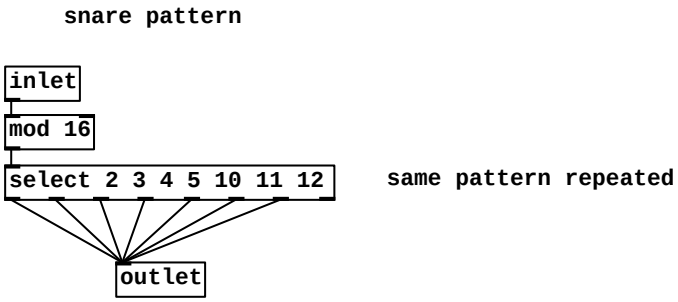

Figure 18: H5-snareseq

- choose one of four
- various pitch and decay

4 parameters Each of four parameters is unpacked and connected to its destination in the DSP. The first parameter sets the top of the sweep range and the second sets the minimum with `[* ]` and `[+ ]` units respectively. Parameter 3 goes directly to the resonance inlet of filter `[bp~ 400 2 ]`.

signals The signal source is `[noise~ ]`, which is modified by the filter. Notice that the control signals for the filter cutoff are not at audio rate. Message domain control is used because `[bp~ ]` cutoff is a message rate inlet. However, we convert the amplitude control to a smoothed signal at the last moment, `[lop~]` is used to remove any sudden jumps. H6-snaredrum.pd

6.7 snare synth

Figure 20 summary

- get parameters
- power 4 decay curve
- smooth signal

4 sounds In an identical fashion to the kick drums, some snare drum random variation. There is a 1/3 chance that no beat will play. On the remaining 2/3 of beats one of 4 possible parameter messages is sent. The values represent filter amount, filter base, filter resonance and total amplitude. As you can see there is a mixture of short, long, high and low values in the four messages.

H61-snare-synth.pd

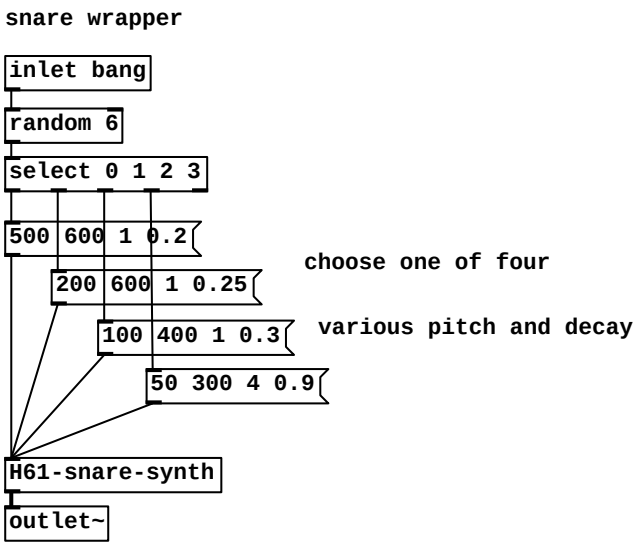

Figure 19: H6-snaredrum

6.8 snare effects

Figure 21 summary

delay and sweep filter These effects are a slowly moving filter on top of a delay unit (which we will examine in the next section). A `[loadbang]` sets up an initial delay time, fixed to 375ms here, and feedback (60 percent), as well as setting the initial resonance of `[vcf~]` which prefers to be set up this way.

low frequency oscillator An lfo can simply be a normal oscillator running very slow when you want audio rate control signals. The cutoff frequency of `[vcf~]` is set by an audio signal which gives a much smoother change of settings. If a control rate filter adjustment is used here at such a low frequency we would certainly hear occasional clicks with a `[bp~ ]` unit. The lfo sweeps up in one direction between 300Hz and 5300Hz about every 2 seconds. H7-snare-fx.pd

6.9 play the drums

Figure 22 summary

old style electro drums This kind of sound is typical of electronic music circa 1979/80, like Kraftwerk (Autobahn), Ultravox (Vienna) or early Human League (Being Boiled). Resonant noise sweeps in the lower ranges have quite a powerful sound.

beat density It's hard to pin down this beat. With just enough randomness the actual beat is unclear on first listening to the drums alone, it seems to

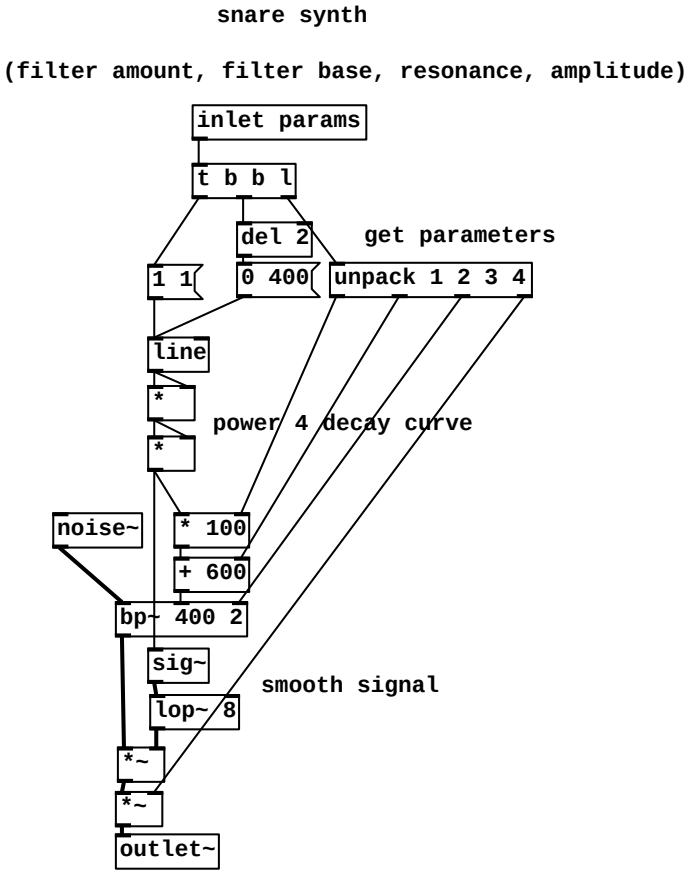

Figure 20: H61-snare-synth

mutate around a bunch of similar patterns. This effect is helped lots by the delay. It doesn't really make sense until we also have a bassline to anchor key points with, so let's build a bass line in the next section. H8-drums-play.pd H8-drums-play.ogg

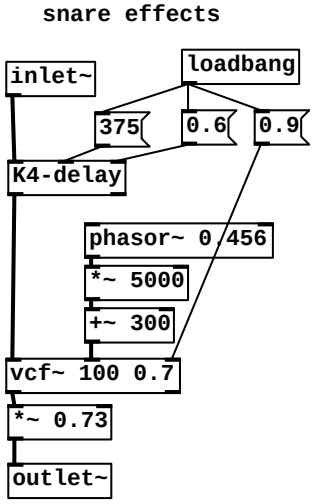

Figure 21: H7-snare-fx

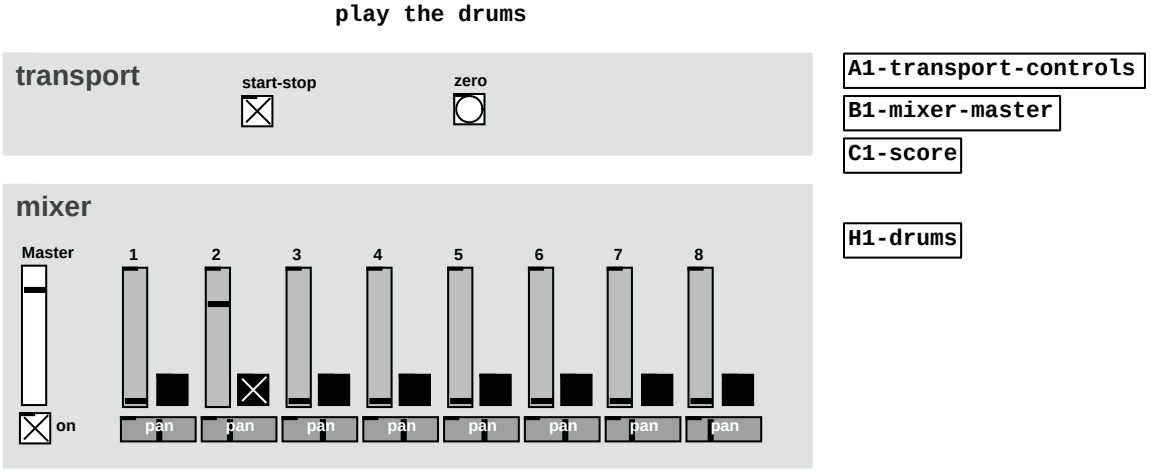

Figure 22: H8-drums

7 bassline

7.1 bassline part

Figure 23 summary

- time and feedback for delay fx

sequencer, synth and delay Again a top level wrapper for a bassline part. This arrangement should be familiar by now, you can encapsulate the sequencer, synth, effects and mixer channel all in the same place. Delay effects are set to twice the global period with a 60 percent feedback amount. This instrument will be controlled by fader 3. I1-bassline.pd

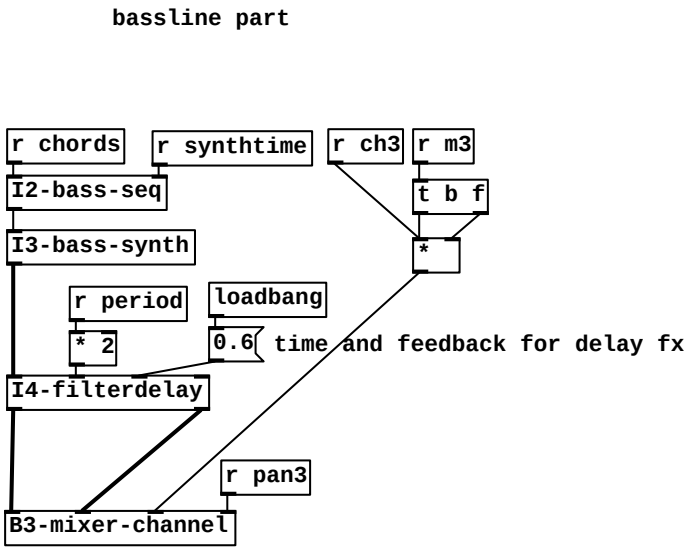

Figure 23: I1-bassline

7.2 bass sequence

Figure 24 summary

- 16 beats to a bar
- 2 part sequence
- add in occasional suboctave
- get the current chord
- pick the first two notes

musical interpretation In some senses bass is the most important part of a composition since it roots everything else, but it doesn't have to be added first. Exactly how you choose to flesh out a compositional idea is entirely personal so my reasons for choosing part patterns and order would not be your choice. Follow along and then rip the patch apart and do it your way with what you learn.

notes and pattern Each chord in the score consists of 8 notes, the first two are important because they set the root and primary interval of the whole harmony. A bassline that alternates between these two notes is interesting with big chords because it gives two different interpretations. The three chords chosen are in tension, we can move between any two of them and the effect is moving towards concordance, but like steps in an Escher drawing it never gets there, there is no cadence so the progression remains forever unresolved. The bass line, and crucially the first and 13th beat marks a flip between either of the first two notes. This change of feel is enough to keep the piece ticking along and producing interesting new variations. Having occasional suboctave bass notes gives it a dark and dramatic feel, but the timbre of the high bass notes is bright and light enough to leave the total effect ambiguous.

select and float box The familiar technique of unpacking some values to float boxes and using the bang messages from selects is used. Note values 1 and 2 of the chord are unpacked, the second being copied to another float box for the suboctave. Beats 2 and 10 of the time sequence are given a probability of 50 percent. I2-bass-seq.pd

7.3 wrapper for cosynth bass sound

Figure 25 summary

parameter substitution In this wrapper you can see we have used an abstraction for a synth called `[z-3cosynth]`, and a message that holds all the parameters for a certain sound except the note number, which is substituted in. For synths that receive all their parameters on each note, note time initialised, we can use a message as a patch template and substitute new parameters on the fly by substitution. An alternative is to have each control with its own independent inlet. There are pros and cons to both methods in terms of flexibility and the amount of data sent on each note (and hence stored somewhere when sequencing). Don't worry about the contents of `[z-3cosynth]` at this point, it is a synth with 3 cosines, and all we need to know is that the parameters in the message give us a nice fat analogue bass sound. I3-bass-synth.pd

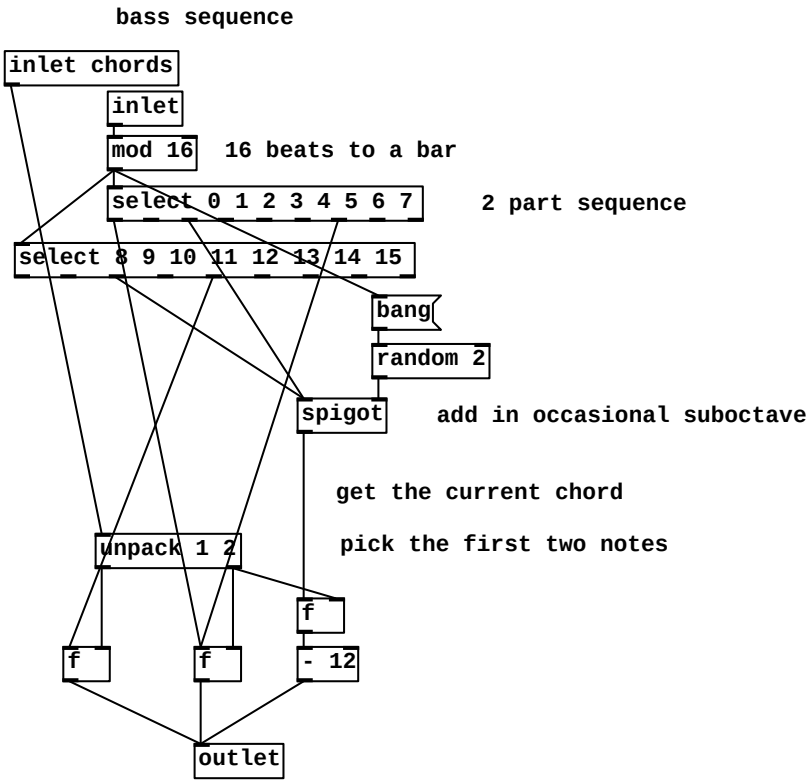

Figure 24: I2-bass-seq

7.4 filterdelay

Figure 26 summary

- slow moving saw lfo
- sweeps frequency up
- L-R delay

another effect Can you spot the important difference between this and the last effect? The filter comes after the delay now. Reordering of effects units makes a big difference to the overall result. This time I've added a `[lop~]` to the `[phasor~]` so that when it flies back to zero there isn't a very sudden change in cutoff. For this effect an extra inlet is included to vary filter resonance. I4-filterdelay.pd

7.5 play the bass line

Figure 27 summary

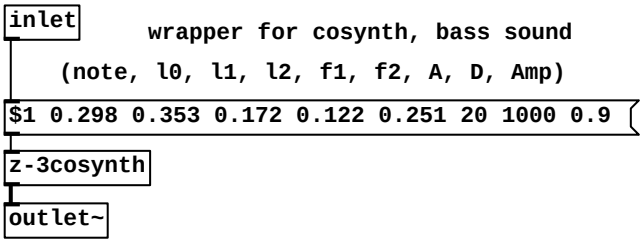

Figure 25: I3-bass-synth

bass line sound Maybe there is a hint of Korg CS or Yamaha MS about this sound, or the Korg Monopoly with 4 oscillators. Richness comes from clipping 3 closely detuned cosine generators to highlight their intermodulation. With filter-delay echoing each note the result is a more blended and continuous bass backing. I5-bassline-play.pd I5-bassline-play.ogg

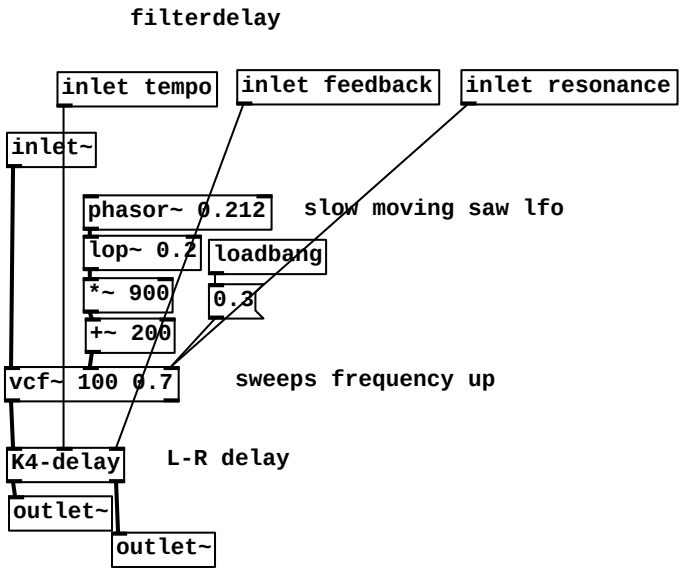

Figure 26: I4-filterdelay

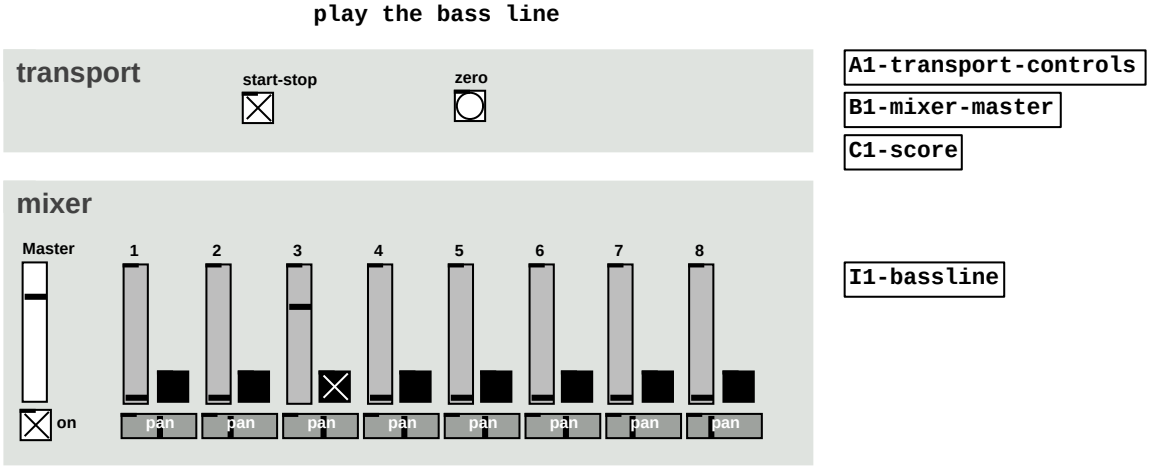

Figure 27: I5-bassline

8 plucks

8.1 pluck strings

Figure 28 summary

- same synth as before with different program
- effects

another 3cos synth Hardly any difference to the previous wrapper. Different delay and feedback times for the effects is all that is apparent at this level. J1-plucks.pd

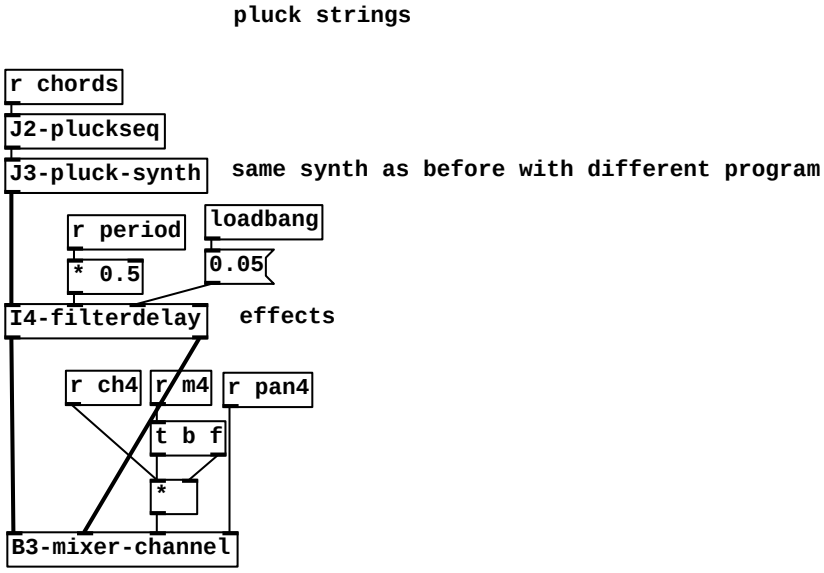

Figure 28: J1-plucks

8.2 random one of two

Figure 29 summary

half a chance We keep using the technique of choosing an event based on a half probability. Let's make that an abstraction and choose a smarter way to do it. Instead of passing a values with a spigot after they are chosen (with certainty), remove events from a stream before they trigger anything. It's very simple, for each bang on the inlet choose `[random 2]`, value of 0 or 1, and `[select 1]` to get only half of them. J2-half-a-chance.pd

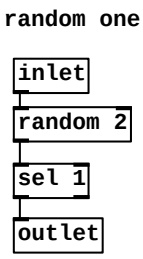

Figure 29: J2-half-a-chance

8.3 pluck sequence

Figure 30 summary

- sparse cycling melody

pizzy string motif Notes 5, 6, 7 and 8 are taken from the chord and repeatedly play a little motif 8 beats long. Beats 1, 2, 4 and 7 always play, but beat 3 is occasionally missing. J2-pluckseq.pd

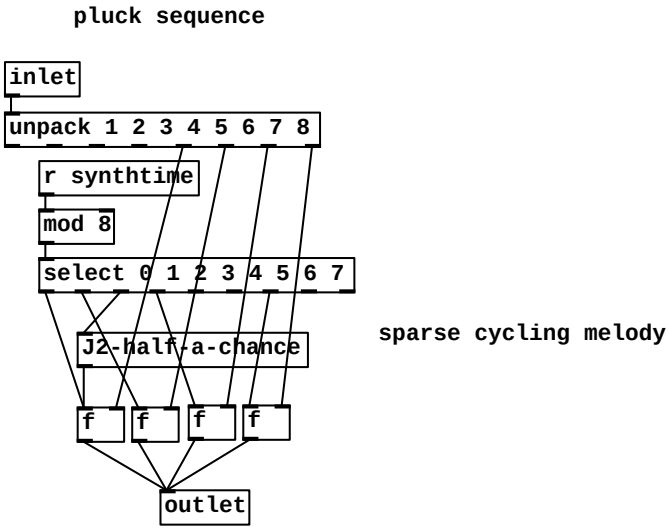

Figure 30: J2-pluckseq

8.4 pluckstring wrapper

Figure 31 summary

- different parameters
- same synth as before

identical synth with different parameters Now see the powerful reuse possible with abstractions. Synth `[z-3cosynth]` is the same one used for the bass line but a different set of parameters in the list makes it a plucked string sound instead. J3-pluck-synth.pd

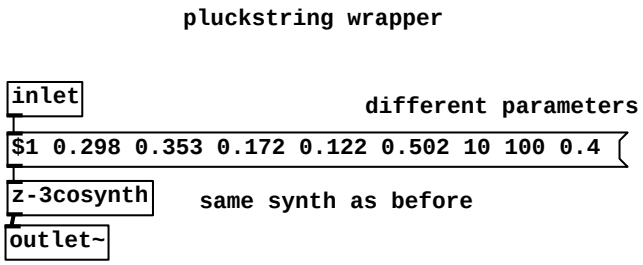

Figure 31: J3-pluck-synth

8.5 play the plucks

Figure 32 summary

slapback delay An important part of bringing this sound alive is the rather short delay time and feedback, half a beat, a common effect on twangy guitar and short percussives and often called a "slapback" delay. Delays shorter than a beat mimic early reverb reflections instead of distinct echos and can add a closeness to the sound. J4-pluck-play.pd J4-pluck-play.ogg

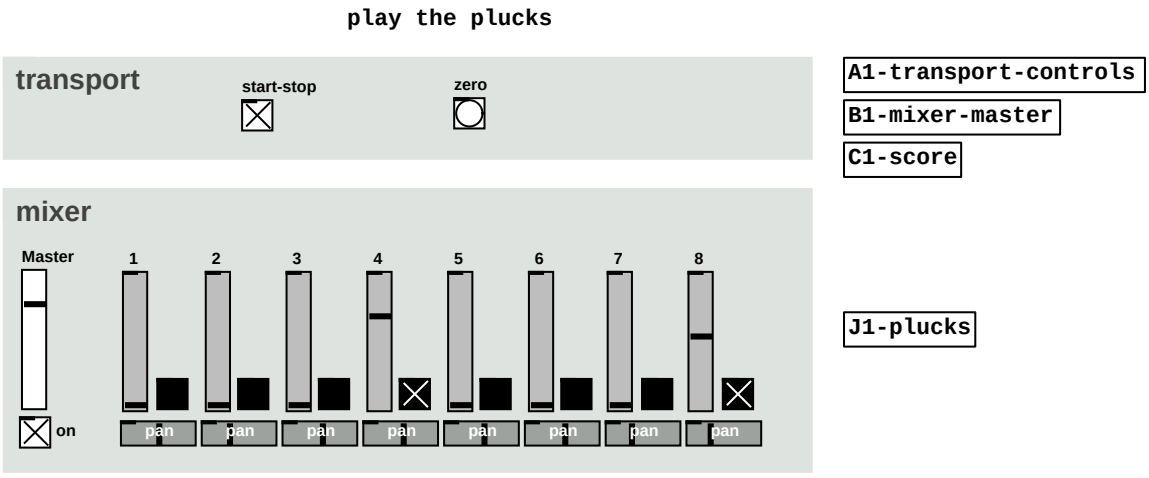

Figure 32: J4-pluck

9 arp

9.1 arp part

Figure 33 summary

- sequencer and synth
- delay fx at 3 x period

same wrapper new synth This diagram should be looking familiar now. Again, effects delay time is a function of period, 3 times the period, and there is the usual arrangement of sequencer and synth with the mixer channel set up for fader 5. K1-arp.pd

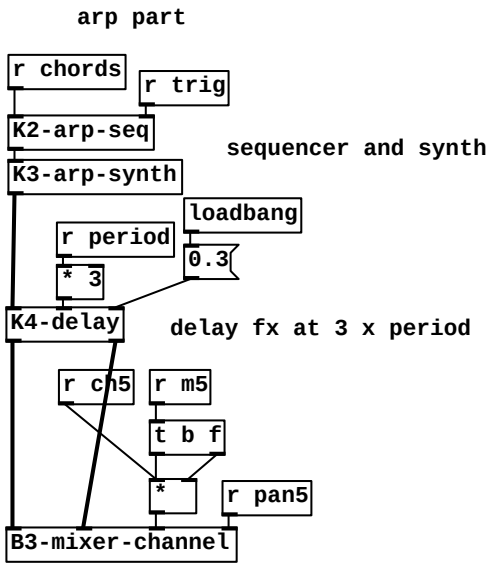

Figure 33: K1-arp

9.2 arp sequence

Figure 34 summary

- float box selector
- copy some notes octave up

random all notes selector All the 8 chord notes are unpacked and available with equal probability on each step, including extra copies of the upper 3 notes transposed by an octave, thus with 11 in total there is also the possibility of a rest from a random selector of 12. K2-arp-seq.pd

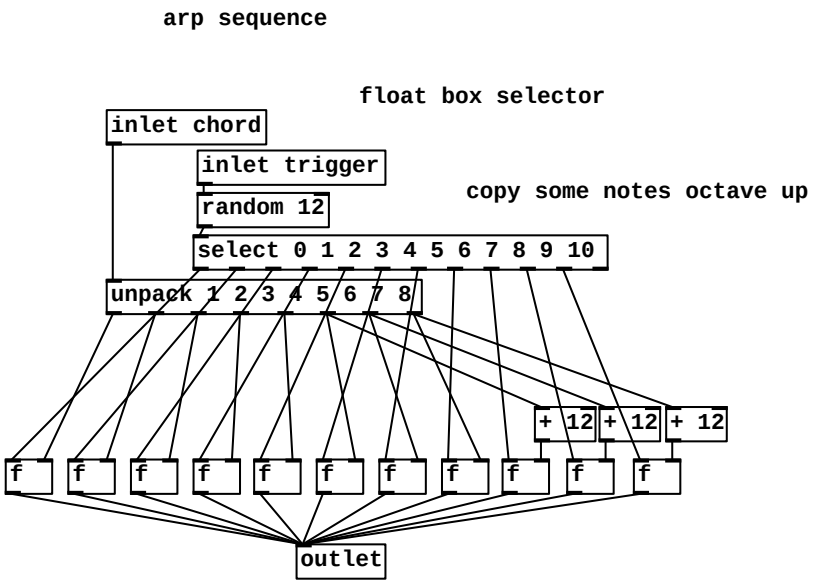

Figure 34: K2-arp-seq

9.3 slide arp

Figure 35 summary

- track at random rate
- envelope only sweeps

sliding sounds This is portamento, where a note always travels between its last value and the new one, as opposed to glissando where the note rises or falls consistently from some value to the target note. Slide time and decay are related so that quick slides have a short envelope and long slides have a matching long envelope. This sound has an interesting effect when combined with the sustained notes of the pad. K3-arp-synth.pd

9.4 cl delay

Figure 36 summary

- filter in feedback loop
- center then left only

center left delay Here is the delay at the heart of most our effects so far. In the feedback circuit is a filter to limit recirculating signals to the midrange. `[clip]` is added defensively to make sure that a feedback greater than 0.9 is never sent by accident. Delays work using pairs of objects in Puredata, one is to

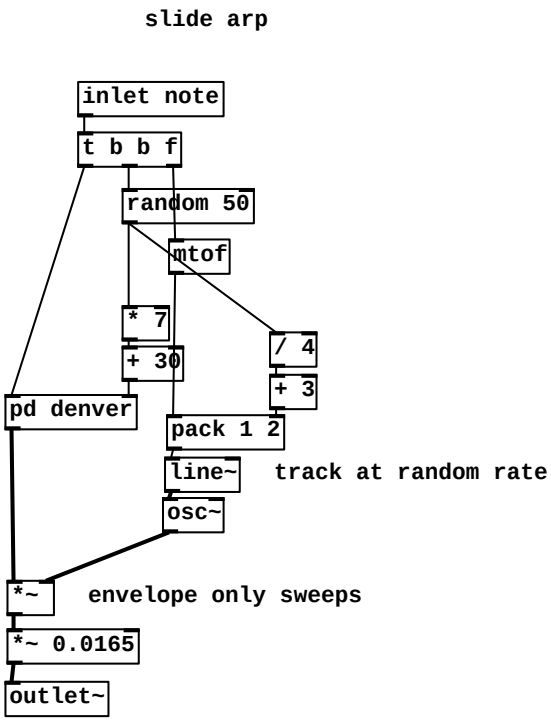

Figure 35: K3-arp-synth

send to the buffer and the other to read from it. The second parameter of the [delwrite] object allocates memory, in milliseconds (times the current sample rate). The names of reads and writes must match and should be uniqe to that pair of objects in most uses, although many reads from the same write is okay for multi-tap reverbs. Time may be varied on [delread] via the first inlet with a float message, but this causes unpredictable clicks as you might expect. Signals arriving at the inlet go first to left and right outlets, and then the delayed signal appears on the left channel only. You can create many variations on the basic stereo delay effect each with their own unique uses and applications. A ping pong is a delay that exchanges left and right in the feedback, a lcr distributes delays left, center, right, and so on. K4-delay.pd

9.5 play the arpeggio

Figure 37 summary

bubbling background A term you might use to describe a random note from the set on almost every beat is "bubbling". This part creates a continuously changing part in a high register that should be mixed in quite quietly to add sparkle to the arrangement. K5-arp-play.pd K5-arp-play.ogg

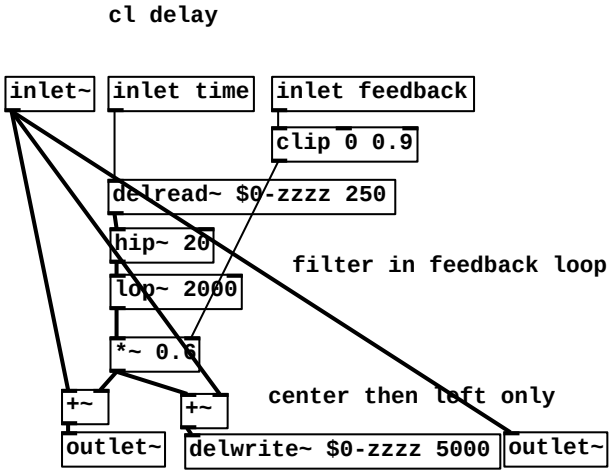

Figure 36: K4-delay

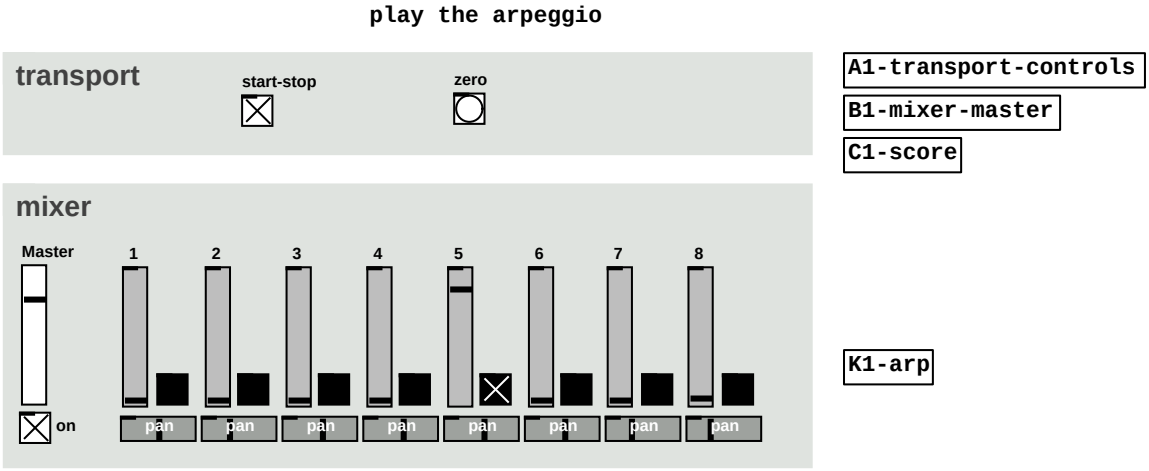

Figure 37: K5-arp

10 sawswoop

10.1 saw-swoop part

Figure 38 summary

long saw sweeps This type of sound features in some Pink Floyd songs and became popular again in 90s chillout/ambient music of artists like Namlook and Morris. It is a resonant saw pitch sweep going right beyond the normal range and used at the start or end of several bars to add new energy or a lead in to a new section.

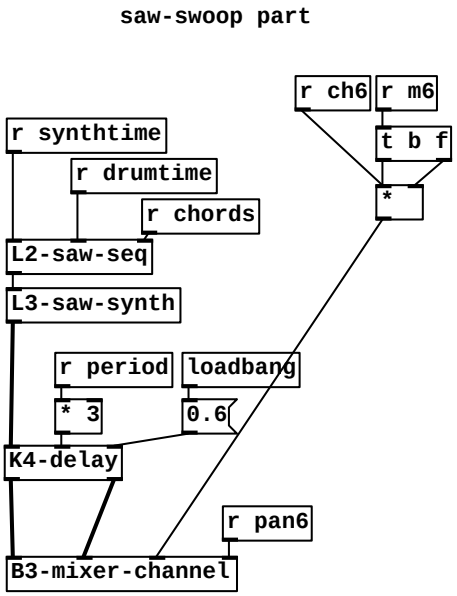

Figure 38: L1-sawswoop

L1-sawswoop.pd

10.2 saw sequence

Figure 39 summary

- non root notes
- random slice
- two possible patches

non root notes All the notes above 1 and 2 are unpacked from the chord. This composition originally had 10 and 12 note chords and there are two redundant values unpacked. Sometimes as a piece evolves, not every part of stays consistent and interesting "features" may be happened upon by accident. The result here should to get an extra two rests, but in fact we occasionally get a zero value note since the redundant floats are set to 0. I decided to keep this "bug" as part of the composition because when the note dives to zero it makes a very nice effect.

selection Two cycles of time are used here, a 16 beat bar and a 64 beat one. This is to add an extra few beats on 57, 61 and the last beat of each 4 regular bars. There are two synth patches selected in the sequencer. Notice parameter substitution occurs here in the sequencer this time, not in the synth, so we can choose different sounds on different beats. L2-saw-seq.pd

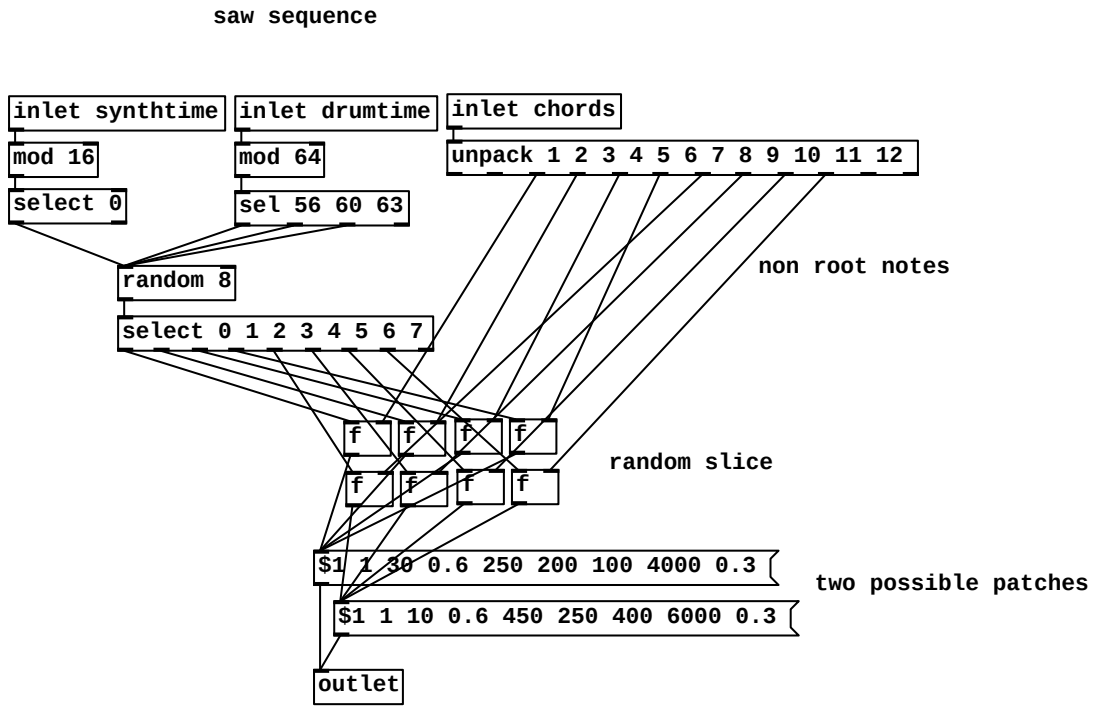

Figure 39: L2-saw-seq

10.3 sawslider

Figure 40 summary

- detutned phasors
- filtered
- clipped

detuned saws and filter There's quite a lot to see in this synth and it's layout is not easy to read. Basically, two phasors are at the core of the sound, detuned from each other so that one lags behind the other when they move but they eventually reach the same value. The summed phasors are filtered with bandpass `[bp~ 100 1]` and clipped `[clip~ -0.9 0.9]`. The filter resonance is set quite high by an incoming parameter.

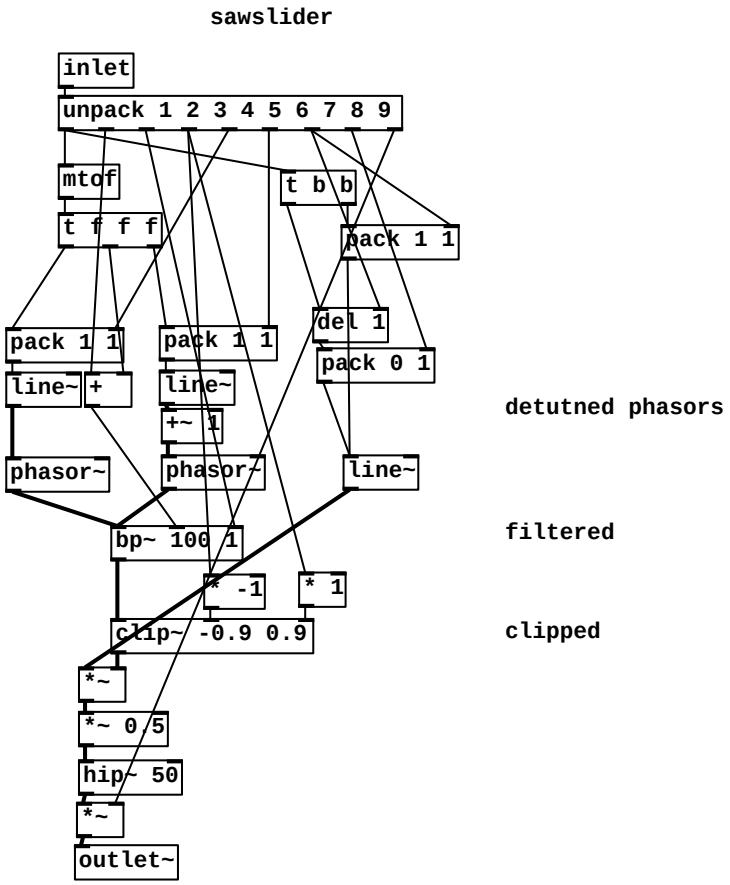

Figure 40: L3-saw-synth

L3-saw-synth.pd

10.4 play the saw swoops

Figure 41 summary

occasional swoops Few occurances of this sparse part occur in the composition. It also varies quite a lot as the last state of of the oscillators makes a difference to how the next note will play out. Sometimes you hear it sweep right up to the limits of the keyboard and then back to zero where it makes a clicking noise as the phasor frequency gets very low. L4-saw-play.pd L4-saw-play.ogg

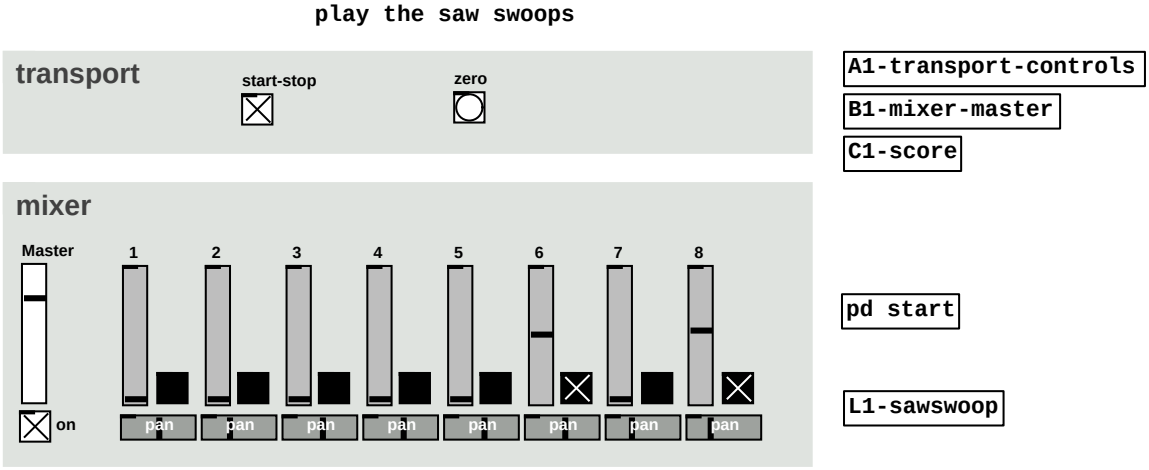

Figure 41: L4-saw

11 strings

11.1 stringy resonances

Figure 42 summary

- hollow brassy/stringy
- randomised synth parameters
- note selector
- fixed synth parameters
- usual synth abstraction

strings patch An extremely simple wrapper, all the interesting stuff is in the `[hollowstrings]` subpatch. M1-strings.pd

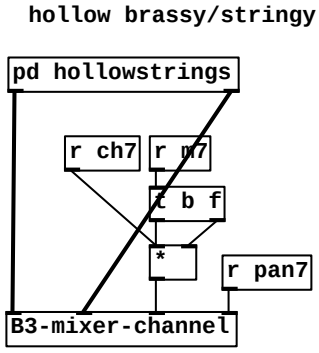

Figure 42: M1-strings

11.2 stringy resonances

Figure 43 summary

- random parameters for synth
- fixed synth parameters

experimental string sound Just to show that you don't always have to break up your code into neat boxes for synths, sequencer and effects here is a strange instrument that is all in the same abstraction. It uses the familiar `[z-3cosynth]` and operates on notes 7 and 8 of the chord an octave down.

random parameters Sometimes when exploring a new synth you might like to hook up a set of objects as on the top right of this patch, which generate random parameters within a fixed range. This setup produces a constantly changing variation on a string-like sound, somewhat hollow and scrapey. See how the dynamic parameters are freely interleaved with fixed ones in the substitution. M2-hollowstrings.pd

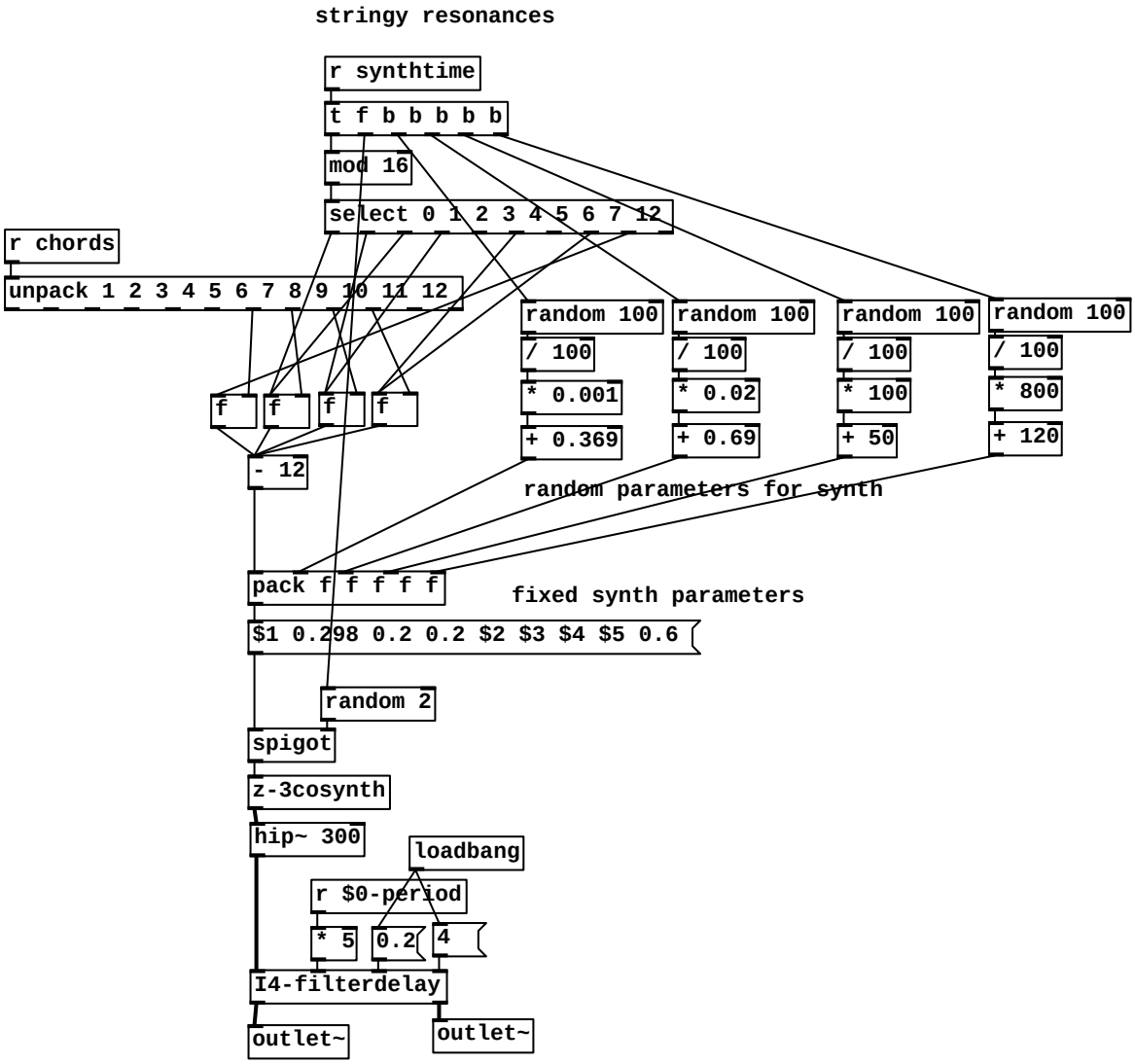

Figure 43: M2-hollowstrings

11.3 play the strings

Figure 44 summary

strange harmonics This part sounds quite weird alone. It was designed with the rest of the mix playing and fits in by offering some tones that are inharmonic and much more organic than the rest. It has an ambiguous timbre, sometimes stringy and sometimes more brass-like, but it does the job of pulling together some of the more disparate higher bits by providing ground between them. M3-strings-play.pd M3-strings-play.ogg

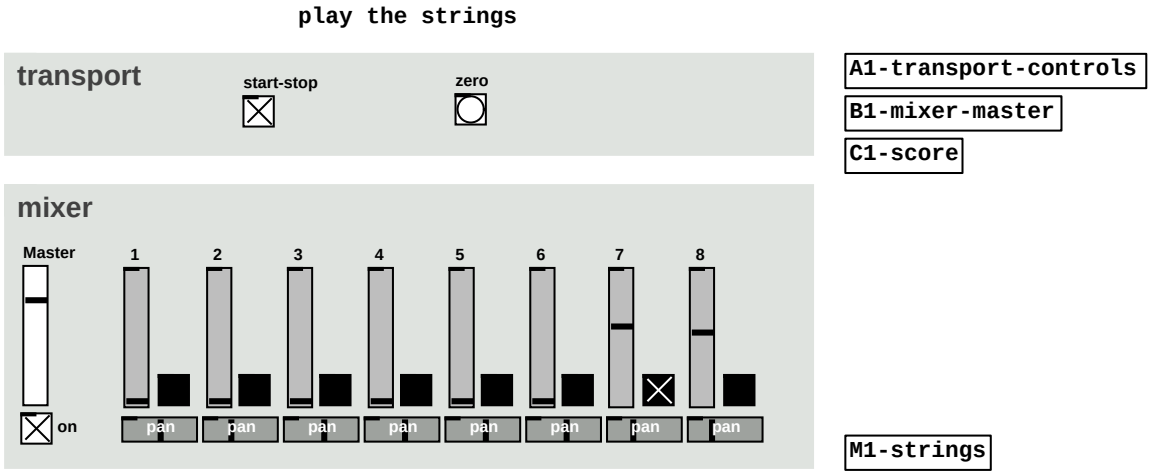

Figure 44: M3-strings

12 Composition002-play

12.1 play all instruments

Figure 45 summary

whole mix Here is the final arrangement of the composition "Moon over Pokesdown station". It took several hours but was completed in one session. Once you have a library of useful abstractions you are familiar with the time from idea to finish is really reduced.

analysis What is this music? Ignoring style and genre it's a good formula for "bed" music, so called not because it makes you want to sleep :), but because it is used in games and radio as an ever changing or evolving background piece, often to talk over. Although the music sounds coherent and thematic it is never exactly the same from one bar to the next so you can render out or synthesise indeterminate lengths of it that never loop. That is useful when you have features that are unknown in length beforehand, such as in a game where unpredictable player actions set the music changes. Algorithmic compositions can use rules to create hours of music, permuting variations on a theme or generating patterns from new rules. This example uses only arithmetic, selects and random sources and doesn't even change its main chord pattern, but it shows the how a piece of music can be built from only simple Pd code.

beard strokers What is procedural media? Puredata is an application to play the file included here. Unlike a wav or mp3 file of music it is not yet determined what it will sound like when you play it. Seeding random numbers will ensure that no two people ever hear exactly the same mix. Procedural video is already quite common, in games and screensavers, but procedural audio less so. This composition has no score dynamics at all, in fact it doesn't even have a beginning, middle and end, it just plays the sequences forever. Interaction is the mixer GUI we built. In Composition 3 we will look at some nicer ways of passing data using route and lists and how to build up a big picture of a score with different movements, tempos, mix parameters and grooves, and then how to influence the score in real time with controllers. N1-Composition002-play.pd N1-Composition002-play.ogg

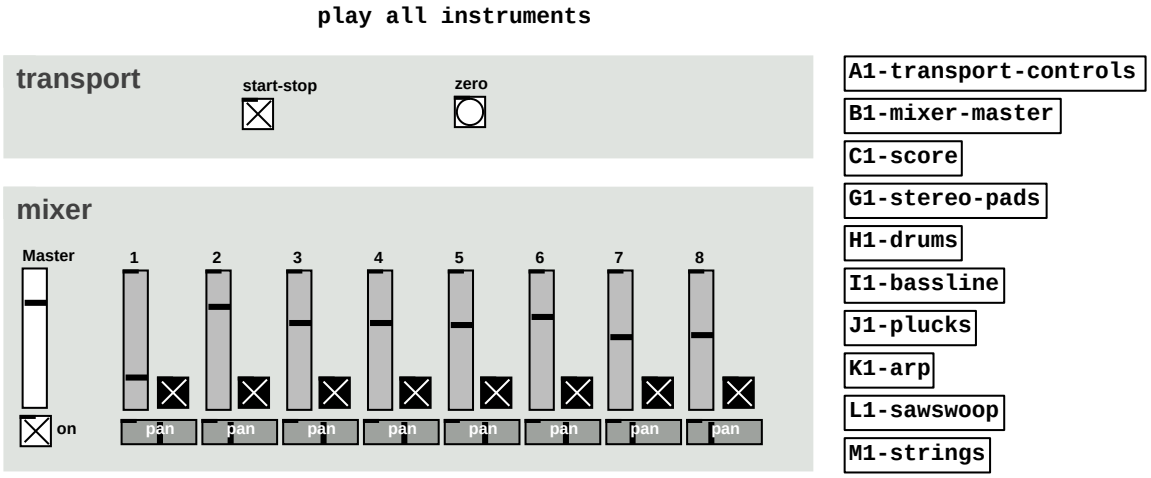

Figure 45: N1-Composition002

13 3cosynth

13.1 3 oscillator and clip synth

Figure 46 summary z-3cosynth.pd

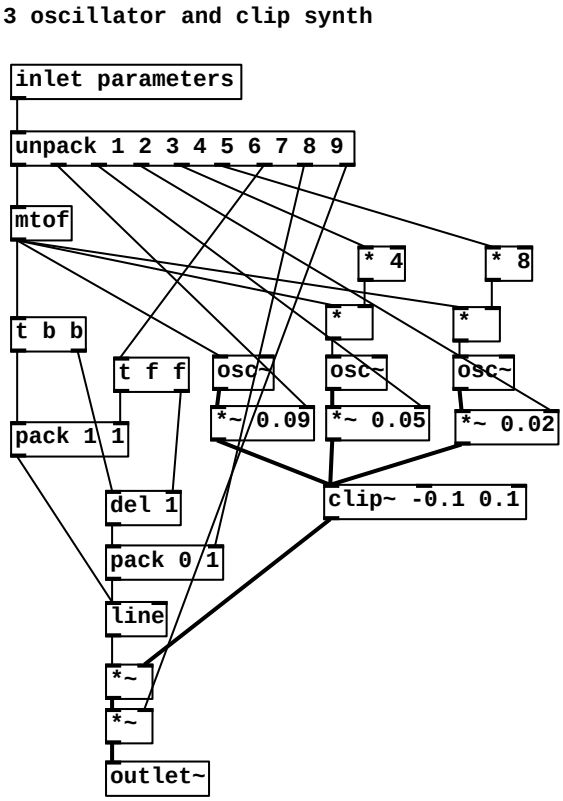

Figure 46: z-3cosynth

14 links

Composition-002.pdf Composition-002.tar.gz