Composition 000

Andy Farnell

8th June 2007

1 Synopsis

Overview A quick look at some very basic Pd concepts. We explore connecting together a metronome and random number generator, playing a sinewave oscillator, scaling a signal using multiplication and introduce some simple GUI objects.

New units used

- `slider`: A GUI slider, horizontal or vertical with settable range.
- `toggle`: A GUI object to switch things on or off , sends a 1 or 0.
- `[metro]`: Generate periodic bang messages.
- `[*]` Multiply two float messages.
- `[+]` Add two float messages.
- `[/]` Divide two float messages.
- `[random]` Generate a random integer.
- `[mtof]` Convert MIDI note numbers to frequency (Hz).
- `[osc~]` Audio signal sine wave oscillator.
- `[*~]` Audio domain multiply.
- `[trigger]` Sequence evaluation order of message events.
- `[loadbang]` Create one bang when patch is loaded, for initialisation.

2 random note

Figure 1 summary

- a timebase generates bang messages
- random number zero to 23
- add 48 as lower limit
- random range is 48 to 71

Timebase To start with we want to create a regular stream of pulses to trigger events. These are bang messages, which really just means "compute something". To get a source of bang messages we need a metronome object, `[metro]`. When a value of 1 appears on the left inlet of a metronome it switches on and begins to emit bangs. It will turn off again uopn recieving a 0 value. The right inlet of `[metro]` sets the speed, or rather the period which is in milliseconds. A toggle switch outputs a 1 or 0, so we can start and stop the `[metro 300]` unit. When started it outputs bang messages at a rate somewhat above 3 per second. This is the most simple timebase, just a regular stream of bang messages.

Random number To turn the bang messages into frequencies we need to make numbers, float values that can be understood by an oscillator or other units. Each bang message recieved by `[random 24]` causes it to choose a new random integer number of 24 possible random values, in the range 0 to 23. We add this to a fixed value in `[+ 48]` so that the minimum value is 48, and the maximum value will be $48 + 23 = 71$. If you attach a number box you will see the number changing to show a different float message is on the wire. On a musical keyboard there are about 88 notes, two octaves of which covers 24 keys, so this seems like a good pitch range for melodies, starting on `C3` and ending on `B4`.

Oscillator Connecting this to the frequency inlet of `[osc~]` produces an audio signal. The left inlet of `[osc~]` sets its frequency in Hz or cycles per second. Both sides of the `[dac~]` are fed from the same `[osc~]` outlet so we get to hear both speakers on a stereo sound system. The patch plays loudly and low, so watch your bass bins. It doesn't produce the correct range of frequency. A range of 48Hz to 71Hz is right down in the bottom, so we must scale it for MIDI note values to produce the right frequencies.

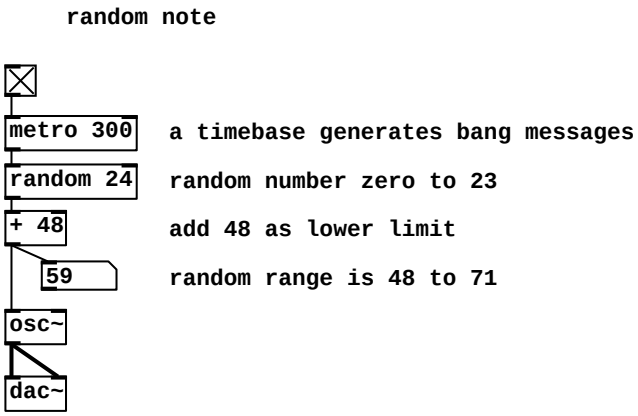

Figure 1: random-note

A-random-note-play.ogg A-random-note-play.pd

3 sine wave synth

Figure 2 summary

- number box
- message melody
- note number to Hz
- oscillator

MIDI note convertion Here we have an `[osc~]` object with its frequency inlet connected to a number that comes via a `[mtof]` unit. This converts "midi" to "frequency", where a midi note is an integer in the range 0 to 127, and frequency is a value in Hertz between zero and about twelve thousand.

Number box type 2 The number boxes in pd-gui can be used either as display devices or as input devices. Here is shown the type 2 box which can be coloured and has other useful properties. Moving the number by clicking and pushing your mouse up and down sends new values to `[mtof]`. If you want finer grain control than just integers try holding down the shift key before you click.

Message melody Some message boxes can be immediately used to compose a melody. This isn't the perfect way to write music, but you can simply add message boxes and click them.

Range Listen to the output frequency change as you move the number over the whole MIDI scale from 0 to 127. Connect another number box to the output of `[mtof]` and notice the relationship between key number and pitch is not a simple linear one. Each 12 notes doubles the frequency.

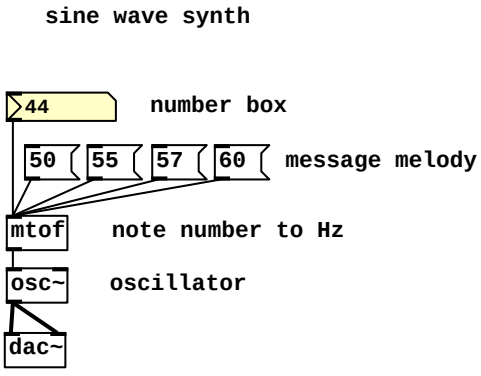

Figure 2: sine-wave-synth

B-sine-wave-synth-play.ogg B-sine-wave-synth-play.pd

4 random midi note sine wave

Figure 3 summary

- send bang to start once at load
- every 300 ms
- pick a random note from 2 octaves
- start on note 48
- convert midi number to Hz
- cosine/sine generator
- make it quieter

Randomness So now we can connect the random number generator and get an octave of notes in a western chromatic scale, a wandering melody. Each number is as likely to occur next as all the others, including itself so some notes may appear to hold for more than one period. `[random]` is actually a pseudo random number generator that follows a very long but theoretically repeatable pattern, although distribution of the numbers is essentialy random. Because the object starts in the same initial state when a patch is loaded you hear the same random sequence unless you "seed" `[random]`. This can be extremely useful in composition to have control over, whether a sequence is predictable but complex, or completely unpredictable.

Loadbang There is a new `[loadbang]` object right at the top of the signal path. This emits one bang when the patch is loaded, so it automatically starts the metronome. Having control over this is useful, you may not want to have to switch on every process in your programs, some must start automatically.

Volume control This patch is made slightly quieter by multiplying the audio signal by 0.3, which is a useful scaling amount when mixing musical signals. In computer composition it's best to work with plenty of headroom until you need to mix something down, about one third of FSD is a fine place to work from. C-random-sine-play.ogg C-random-sine-play.pd

5 random sine wave with controls

Figure 4 summary

- slider controls rate
- every 200 ms
- order bang messages
- random between 0 and 100
- divide by 100 to get between 0 and 1

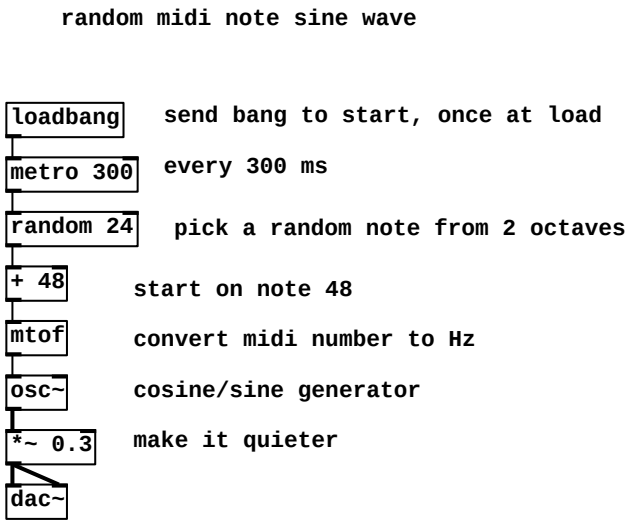

Figure 3: random-sine

- amplitude control
- be quieter

Slider Controls A slider can be placed anywhere you want to vary something by connecting directly to it. In this case we have slider outlet going to metronome period inlet. You should set slider properties such as colour, range and size by right clicking and using the `[properties]` dialog. Range is most often important. Here the slider goes between 10 and a few hundred, so that we never tell the metronome to have a period of zero (which makes a very strange noise sometimes). Another useful property to set is `[init]`, which makes the slider remember the last value set when you save and reload a patch.

Scaling amplitude Now we vary the amplitude of the oscillator so some notes to play more more quietly and some more loudly. Thinking about appropriate ranges, to scale a signal we want to multiply it by a number between 0 and 1. At zero we will have complete silence, because anything multiplied by 0 is 0, and at 1 the signal will be at its normal strength. But `[random]` only gives us integers. To get a normalised signal, 0 to 1, we divide the output of `[random]` by the same value as its range minus one. Using the value of 100 we get one hundred possible steps now mapped between 0 and 1, $0.01, 0.02, 0.03...0.99$ which can be used to scale the audio. Notice that it doesn't matter which side of the (symmetrical) `[*~]` object we connect control signal or audio signal to.

Signal ordering with triggers Sometimes it matters which way you connect objects because of the order you want things to happen. A trigger is like

a splitter that can be used to keep program diagrams neat and logical. It's evaluation order goes right to left so in this case the random number affecting frequency is computed before the random number for amplitude. You will hear no difference if you swap the ordering on this simple patch, but in more complex Pd programs the ordering becomes important. D-random-sine-controls-play.ogg

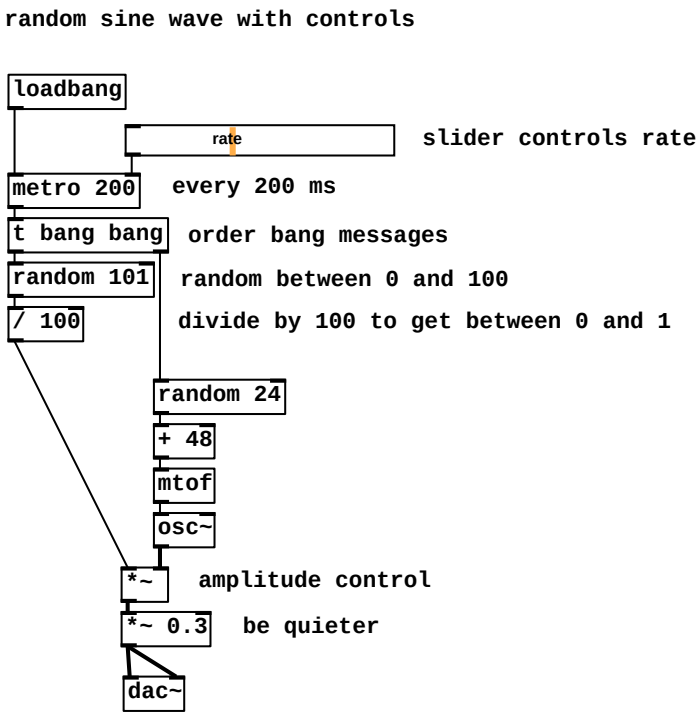

Figure 4: random-sine-controls

D-random-sine-controls-play.pd

6 Subpatched oscillator

Figure 5 summary

- inlet boxes connect to outside
- reusable code inside
- audio rate output

Subpatching If we want to tidy away a whole bunch of code into its own box we can use subpatches. Begin by creating a new object whose name begins with the letters pd and then a space. You can make the remainder of the name anything you like, including spaces and further symbols to help you remember

parameter inputs if you want, but these have no formal effect. Once created the subpatch will automatically open and you can copy and paste code into it, or write new code inside. A subpatch is a child of the main window and does not need saving separately.

Sub-patching the oscillator For every inlet box placed inside a subpatch a corresponding cable inlet appears on the subpatch box. Here we have split the previous patch at the point where MIDI notes and amplitude messages can be separated from the oscillator, and placed the oscillator inside its own box along with a `[mtof]` convertor. It's nice to tidy patches this way, but the real reason for subpatches is to allow easy reuse of code. You should generally make a subpatch of anything you may wish to use more than once. The outlet of the subpatch is an audio rate connection, so observe the tilde in the `[outlet~]` box.

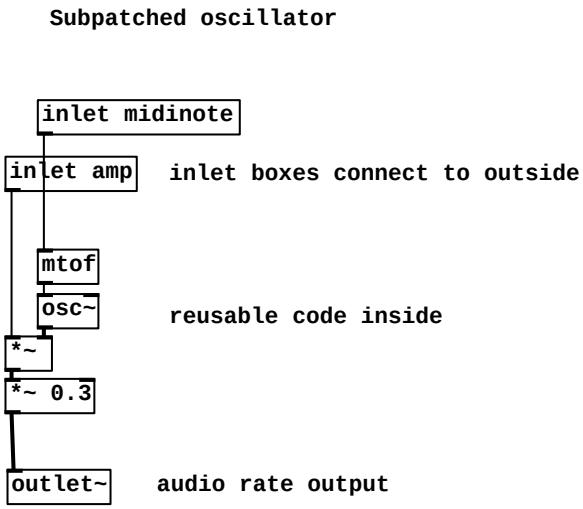

Figure 5: subpatched-oscillator

7 subpatched sine wave gens

Figure 6 summary

- Subpatched oscillator
- inlet boxes connect to outside
- every 200 ms
- reusable code inside
- second oscillator plays harmony

- audio rate output
- subpatched oscillator
- a second copy

Using subpatches In this final example we have copied the oscillator subpatch and changed the parameters of the composition a bit to get a more pleasing interval and scale. Multiplying the midi note by a small integer sets the scale interval, and adding to it provides a fixed offset. This way we have a scale of six notes, five semitones apart starting at C3 with the second part playing five semitones higher than the first. The variables that Puredata sees inside a subpatch are the same as those in the main patch, in programming words a subpatch does not provide scope. This means that some patches, especially those that use tables, cannot be subpatched to make them easily copied. For this we use abstractions instead, which will be introduced in the next tutorial.

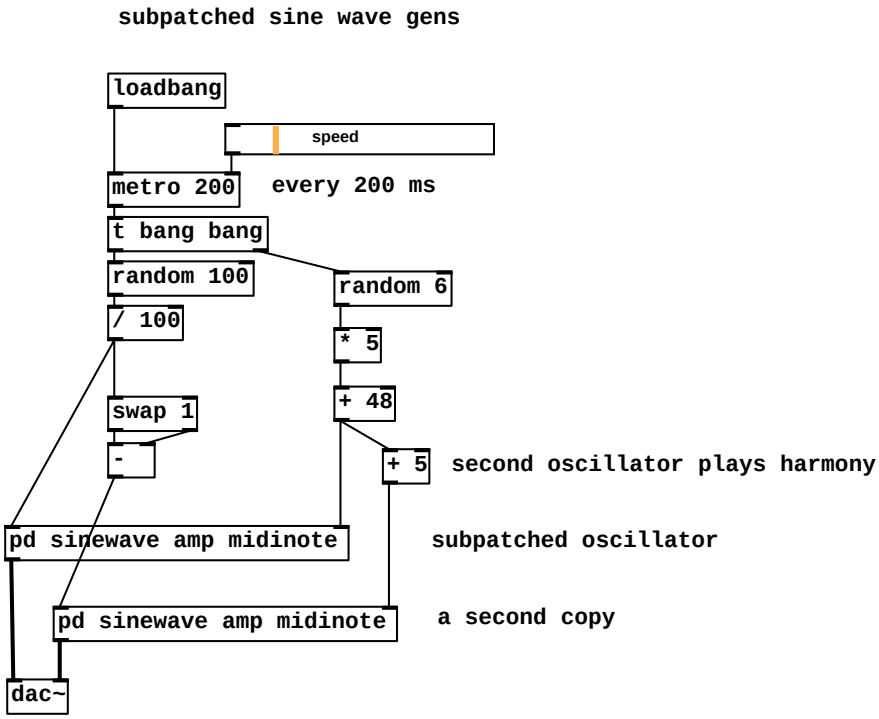

Figure 6: two-oscillators

F-two-oscillators-play.ogg F-two-oscillators-play.pd

8 links

Composition-000.pdf homepage