Unorthodox Sound Synthesis

Seminar
Electronic Studio, TU Berlin
February 11th to 13th, 2020

Techniques

Daniel Mayer
Institute of Electronic Music and Acoustics
University of Music and Performing Arts Graz

About the accompanying SC examples

- All examples of these files need miSCellaneous_lib (>= v0.22) to be installed, you can install it from the Quarks gui interface (needs git installed) or directly. See the installation instructions here:
  https://github.com/dkmayer/miSCellaneous_lib
  https://daniel-mayer.at/software_en.htm

- There are examples taken over from the help files, but often slightly altered and partially simplified. Further examples have been added.

- For the sake of clarity most given examples are rather short.

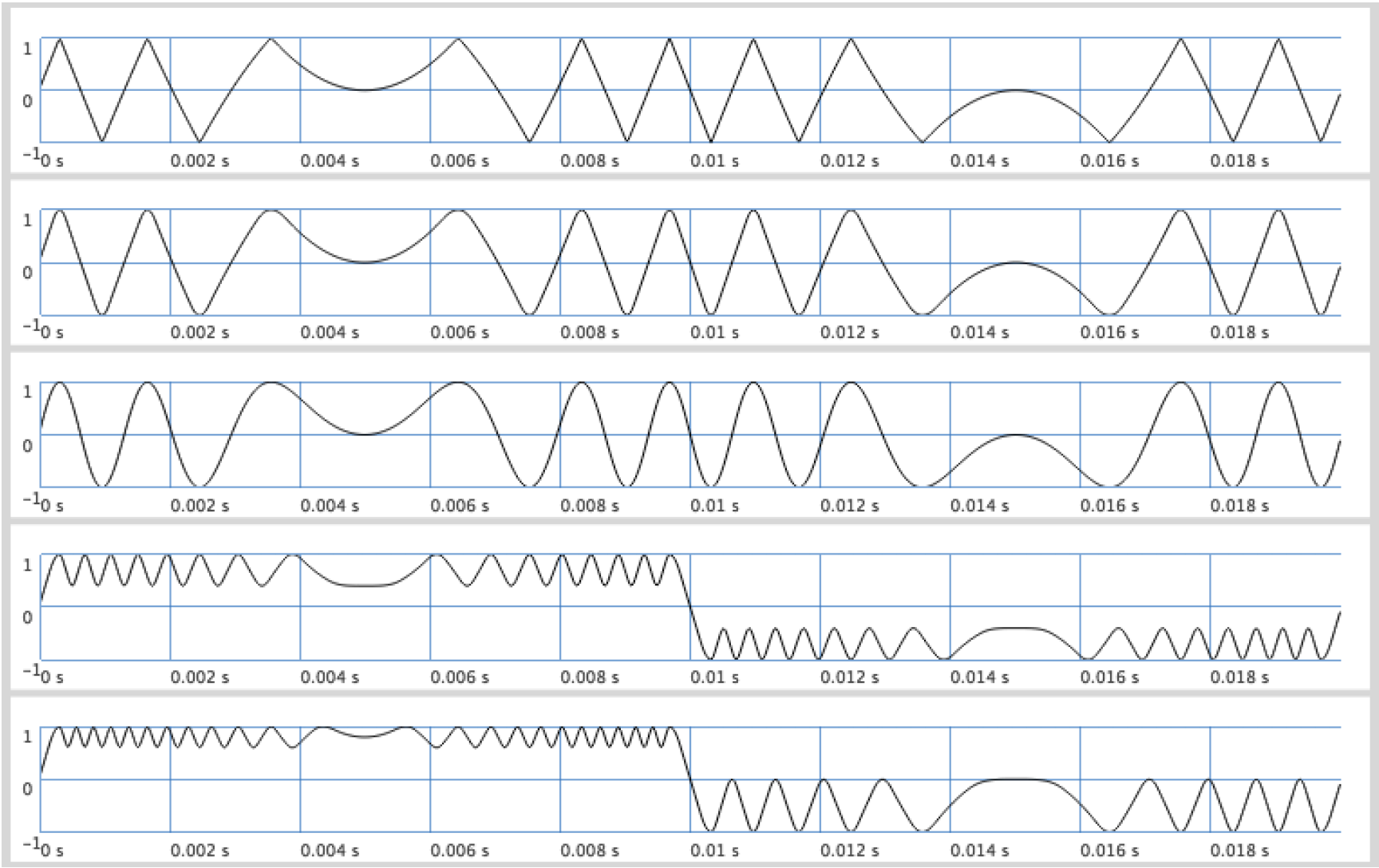
- Clearly, for practical usage one would rather write them as SynthDefs (or Ndefs), and/or add control with envelopes, apply sequencing, guis etc.

- There are a few examples for gui control, which you can use as a rough template.

Wavefolding

- Proposed by Don Buchla and implemented in the Buchla 259
- Esqueda, Fabian & Pöntynen, Henri & Bilbao, Stefan & Parker, Julian. (2017). Virtual Analog Model of the Lockhart Wavefolder. Proceedings of the 14th SMC, Helsinki.
- Esqueda, Fabian & Pöntynen, Henri & Välimäki, Vesa & Parker, Julian. (2017). Virtual Analog Buchla 259 Wavefolder. Proceedings of the 20th DAFx, Edinburgh.

Smooth Wavefolding in SC, Ex.1, plot



Smooth Wavefolding in SC, Ex.1, code

```
{ [
    // folding with main lib's Fold ugen
    Fold.ar(SinOsc.ar(50) * 10, -1, 1),

    // folding with rather low smoothing
    // wave shaper is partiallly a sine wave
    SmoothFoldS.ar(SinOsc.ar(50) * 10, smoothAmount: 0.3),

    // folding with maximum smoothing
    // wave shaper is full sine wave
    SmoothFoldS.ar(SinOsc.ar(50) * 10, smoothAmount: 1),

    // wave is folded back only to border ranges
    SmoothFoldS.ar(SinOsc.ar(50) * 10, foldRange: 0.3),

    // folding with different sizes of border ranges
    SmoothFoldS2.ar(SinOsc.ar(50) * 10, foldRangeLo: 0.5, foldRangeHi: 0.2)
] }.plot(1/50)
```

Options of Smooth Wavefolding

- Sinusoidal or quadratic smoothening
  (classes with suffix S or Q)

- lo and hi bounds

- foldRange (relative to bounds), either global or differentiated
  (lo and hi, classes with suffix S2 or Q2)

- smoothAmount

- All params can be modulated at audio rate

Buffer Modulation (Buffer Scratching) – an Alternative Microsound Technique

Technically this is waveshaping with audio data as transfer function ...

... read by a buffer pointer

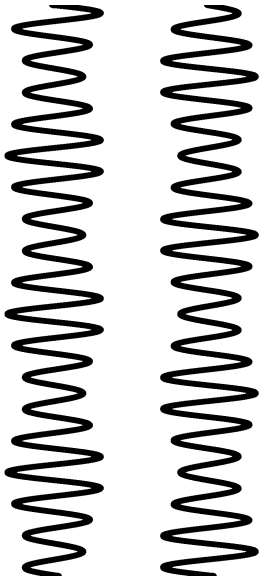
L/R-decorrelated buffer modulation: slightly altered/modulated pointer signals

transfer function

L/R buffer pointers

here amplitude-modulated

both have same middle position



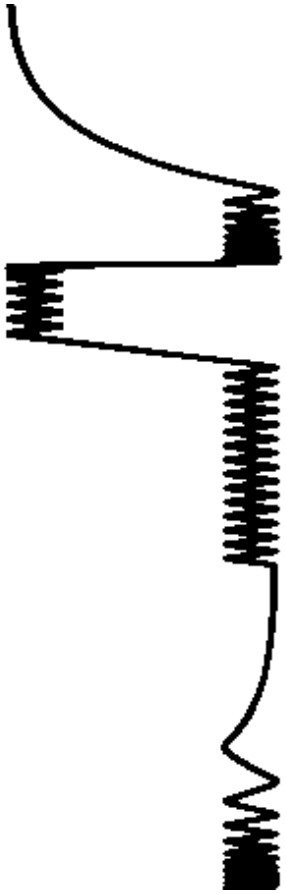
Adaption of a common granular technique: modulated movement through a buffer

The reading signal can be defined as sum of an LFO, which defines a global buffer movement and a local (stereo) oscillation

Buffer Modulation
Matters 1, 2: folded wave as local oscillation

Matters 1:
folded sine
and triangle

Matters 2:
folded stochastic
waveform



Functional Iteration Synthesis (FIS)

- Proposed by Agostino Di Scipio

- Idea: define a non-linear function $f(x, r)$ with a parameter r. Let f^n be the n-th iteration of the function (n is fixed) and x_0 be a start value, then investigate $f^n(x_0, r)$.

- Ex.: $f(x, r) = \sin(r * x)$
  $f^3(x_0, r) = \sin(r * \sin(r * \sin(r * x_0)))$

- f^n can be used as a synthesis engine if we take r and/or x_0 as time-variant.

Relation and Difference to Non-linear Feedback and Boundary Value Analysis

- With non-linear feedback we also take a non-linear function f, but regard the sequence of samples

  $y_0 = x_0,\ y_1 = f(x_0),\ y_2 = f(f(x_0)),\ y_3 = f(f(f(x_0)))\ \ldots$

  So we are iterating ad infinitum, whereas with FIS the iteration number n is **fixed** !

- Investigations of iterations of non-linear functions (like the logistic map) tend to have a focus on infinite iteration, the analysis of fixed points etc. These different approaches to non-linear iteration can both be used for synthesis.

The class GFIS – generalised FIS

- The essential data for the GFIS pseudo ugen is a SC Function, which is in turn applied *n* times to build the UGen graph representing the iterative calculation

- The iterated Function can have arbitrary (LFO-)modulators at each iteration step (difference to strict FIS)

- The Function itself can vary at each iteration step as the iteration step is passed as argument (difference to strict FIS)

- GFIS can deal with multichannel signals and specify iteration on single components (difference to strict FIS)

Single Sample Feedback / Feedforward

- A generalised feedback / feedforward relation can be written

$$out[n] = F(out[n-1], out[n-2], \ldots, in[n], in[n-1], \ldots)$$

out[n] denotes the n-th output sample which is calculated by *F* from previous output samples, the feedback data

$$out[n-1], \ldots, out[n-j]$$

and previous (+ actual) input samples, the feedforward data

$$in[n], in[n-1], \ldots, in[n-k]$$

- *F* can be any function (non-linear)

- *F* can take and produce arrays (also different fb/ffw sizes)

Linear Time-Invariant (LTI) Filters

- New samples are generated by the linear difference equation

$$out[n] = a_0 * in[n] + a_1 * in[n-1] + ... + a_{n-j} * in[n-j] + b_1 * out[n-1] + b_2 * out[n-2] + ... + b_{n-k} * out[n-k]$$

where a_i are the feedforward and b_i the feedback coefficients. LTI filters are well-explored, Fb1 can be used to implement LTI filters that are not included in SC's main library, coefficients can be controlled dynamically.

- From the „unorthodox" viewpoint though it is also interesting to look at non-linear filters like Dobson-Ffitch (1996):

$$out[n] = a * out[n-1] + b * out[n-2] + d * out[n-L]^2 + in[n] - c$$

Single Sample Feedback with the class Fb1

- Single sample feedback can in principle be done in plain SC by setting the server‘s blocksize to 1 (and a couple of other tricks for special cases).

- The class Fb1 provides an interface for doing single sample fb, regardless of the server‘s block size. The feedback / feedforward relation can be passed to Fb1 by a SC Function in a syntax which is very close to standard DSP notation.

- Fb1 options: arbitrary and dynamic lookback depth, multichannel handling, setting initial values and others enable complicated fb setups with very condensed code.

Single Sample vs. „Default“ Feedback

- With a default blocksize of 64 we get a minimum fb delay of 64/44100 (or 64/48000) seconds.

- The short fb delay time of single sample fb can cause a drastically different behaviour. E.g. blowups can happen much more sudden, slow evolving changes, which might be present with block size 64, can disappear with single sample feedback.

- Fb1 is based on an idea of Nathaniel Virgo, who has written a Feedback quark, here some of his tipps: https://www.listarc.bham.ac.uk/lists/sc-users-2009/msg56802.html

Synthesis with (Systems of) Ordinary Differential Equations (ODEs)

- We regard systems of the form

$$Y'(t) = F(t,\ Y(t))$$

in the domain of real numbers where *Y* and *F* can be vector-valued functions and an initial value condition

$$Y(t_0) = Y_0$$

is given. *t* can be regarded as time in a physical interpretation.

- For many systems of that kind explicit solutions do not exist, instead numerical solutions can be found, often they show complicated behaviour even if the ODE system looks simple. **Value for synthesis:** we have a **powerful description system** to create interesting waveforms by a small set of controls.

A rather trivial example

- The second order differential equation

$$y''(t) = -y(t)$$

can, by the substitution

$$w(t) = y'(t)$$

be reduced to

$$y'(t) = w(t) \quad \text{and} \quad w'(t) = -y(t)$$

With the initial values

$$y(0) = 0 \quad \text{and} \quad w(0) = 1$$

we get the solutions

$$y(t) = sin(t) \quad \text{and} \quad w(t) = cos(t)$$

If we regard the initial equation and think about the not totally trivial inifite row definition of sin and cos, we can see this equation as a rather elegant description of the harmonic oscillation !

ODE integration clearly doesn't have a value for synthesis in this case, but there are many other ODE solutions, that can not be composed by standard math operators / ugens.

Numerical Solution of ODE systems for producing Audio

- Standard numerical procedures of ODE integration (like Euler, widely used variants of Runge-Kutta and others) are not suited for ODE audification as they tend to become instable with the large number of oscillations that are necessary to produce audio.
- Instead the so-called **symplectic** procedures are strongly recommended. David Pirrò (IEM Graz) has investigated this in his dissertation and uses the „rattle“ procedure in his ODE audification system *Henri*. The same procedure is used by the miSCellaneous_lib classes Fb1_ODEdef and Fb1_ODE, based on Fb1 (numeric integration is a special case of fb).
- D. Pirrò, Composing Interactions. Dissertation, Institute of Electronic Music and Acoustics, University of Music and Performing Arts Graz, 2017, pp.135-146.
  https://pirro.mur.at/media/pirro_david_composing_interactions_print.pdf
  https://git.iem.at/davidpirro/henri

Starting points for Synthesis with ODEs

- There exists a huge number of ODE models from physics, electrical engeneering, population dynamics, economics, chemistry etc., especially those with oscillatory or quasi-oscillatory solutions and/or chaotic features can serve as an audio workhorse.

- A good overview with an experimental approach:
  Trefethen, Lloyd N.; Birkisson Ásgeir; Driscoll, Tobin A. (2017): Exploring ODEs. SIAM - Society for Industrial and Applied Mathematics. Free download from: http://people.maths.ox.ac.uk/trefethen/Exploring.pdf 1

- You can experiment with your own ODE definitions, either by altering existing models (e.g. the harmonic oscillator or mass-spring-damper) or starting from scratch.

Implementation with Fb1_ODEdef / Fb1_ODE

- With Fb1_ODEdef an ODE system is defined and ready for usage in a SynthDef. Then the Fb1_ODE pseudo ugen merges the functional composition of numeric procedure and ODE into the SynthDef graph.

- For some well-known systems like Mass-Spring-Damper, Van Der Pol, Duffing, Hopf, Lorenz and some variants wrapper classes are included, so there's no need to define them with a Fb1_ODEdef.

- Besides the recommended symplectic integration methods (with adjustable "accuracy") families of alternative methods (Runge-Kutta, PEC, Adams-Bashforth, Adams-Moulton, improved/modified Euler) are implemented. As with Fb1_ODEdef also integration methods can be defined interactively with the class Fb1_ODEintdef.

Granulation in SC

- Granulation is a huge synthesis topic. As the sounding results can be extremely different, it's actually more than just another "synthesis technique", rather a whole family of synthesis techniques with a common strategy.
  For an overview see Curtis Roads (*Microsound*) and Alberto de Campo's chapter of the same name in *The SuperCollider Book,* SC examples can be downloaded from https://github.com/madskjeldgaard/scbookcode
  Rather basic examples are contained in the "Tour of Ugens" help file:
  https://doc.sccode.org/Guides/Tour_of_UGens.html - Granular Synthesis

- Since the 90s granular synthesis has established as a "mainstream" synthesis method, however there are so many possible variants (e.g. see C. Roads), that also from the "unorthodox" viewpoint it is still a very promising area.

- In SC triggering of grains can be done with language (e.g. Patterns, Tasks, Routines) and/or server (e.g. with dedicated UGens like TGrains). These two strategies and their combinations are described in miSCellaneous_lib's tutorials "Buffer Granulation" and "Live Granulation" with a number of examples. Ex. 1a, 1b and 2a are rather basic, the others require some familiarity with SC.

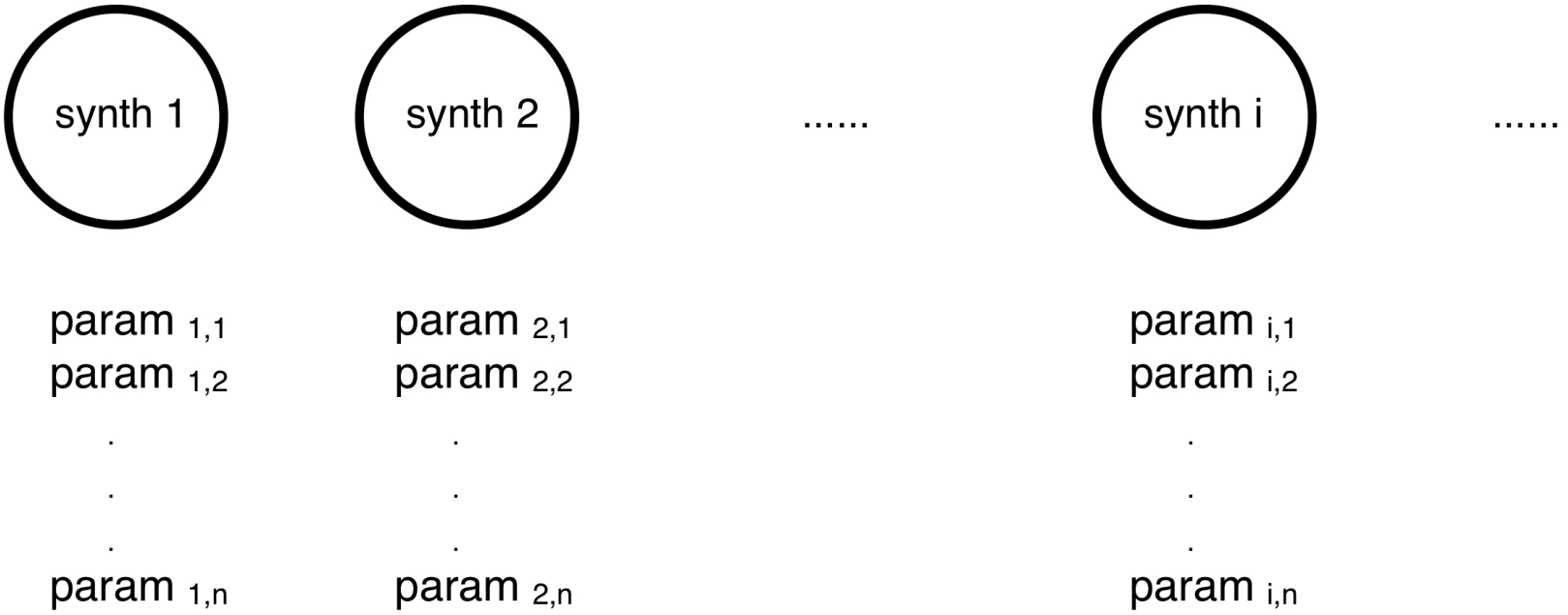
Sequencing of fx processings (PbindFx)

- Here the term FX (effect) is used in a very broad sense. In fact PbindFx is a tool to sequence events with arbitrary graphs of instruments/effects. It takes over many conventions from Pbind, a main working horse for sequencing in SC. Pbind is an extremely versatile class, not limited to the production of audio, it can sequence any kind of action ! There even exists a SC tutorial mainly based on event patterns: https://ccrma.stanford.edu/~ruviaro/texts/A_Gentle_Introduction_To_SuperCollider.pdf

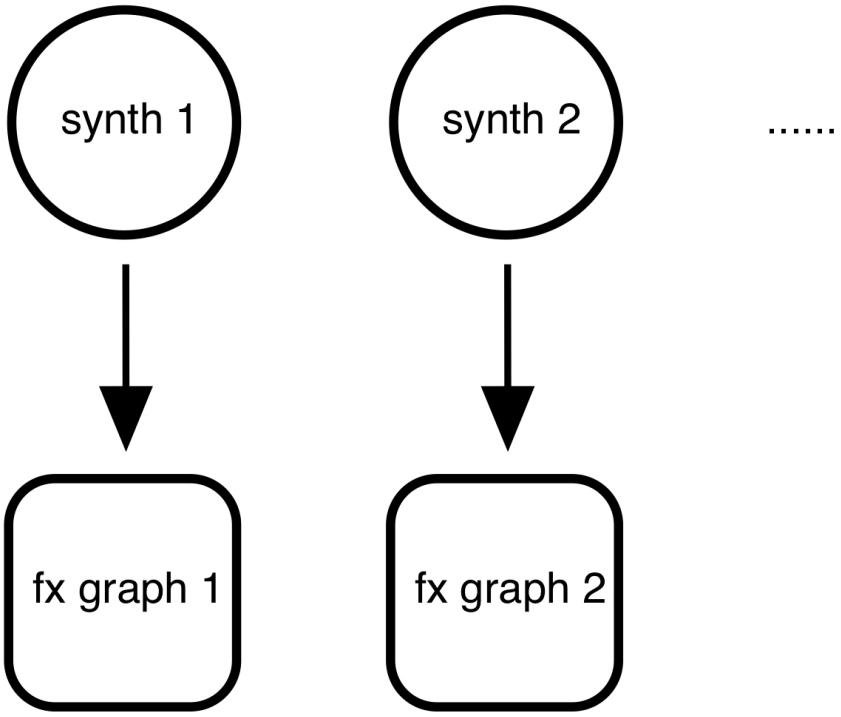
- As events can also be short, Pbind can also be used to do granulation, in fact this is a very flexible method (propably more flexible than with UGens). Here grain density depends on OSC bandwidth (max. a few hundred per second), but this doesn't have to be a practical limitation.

- Consequently PbindFx can also be used to do granulation, now with different fx processing per grain and sequenced parameters.

Sequencing with *Pbind*

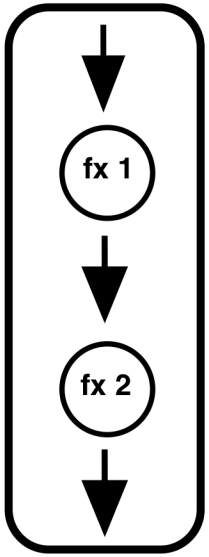


Sequencing with *PbindFx*

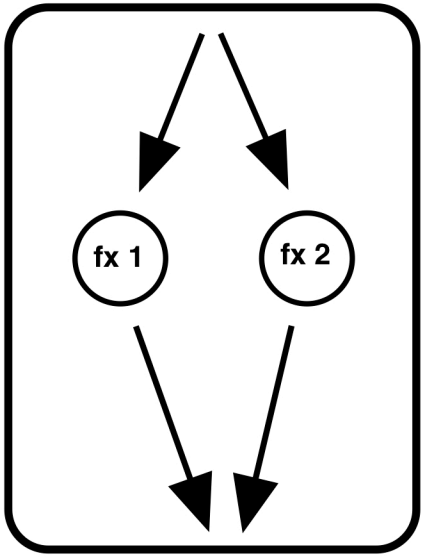


Sequencing with *PbindFx*, types of fx graphs

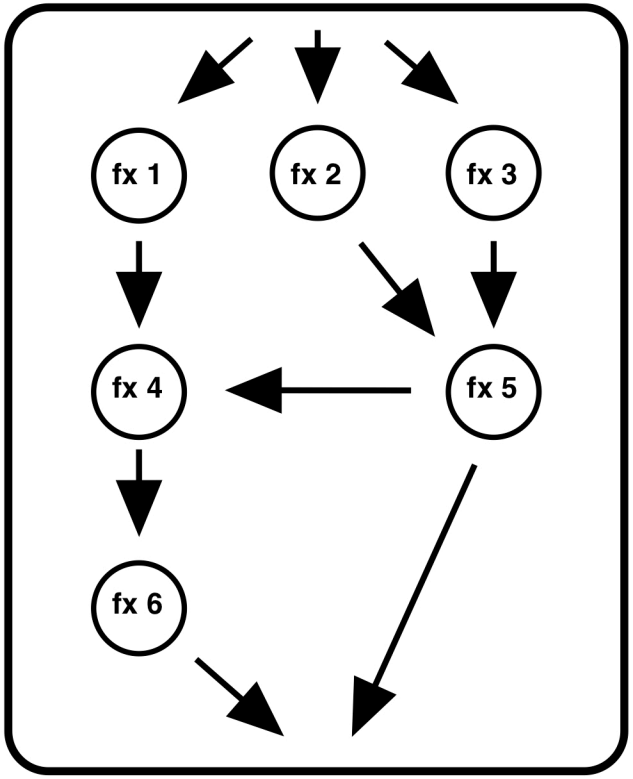
sequential



parallel



arbitrary (but non-cyclic)



Things to consider when working with PbindFx

- PbindFx is a rather complicated class, which extends the functionality of Pbind, the user should be familiar with the basics of Pbind.

- It is important to understand that fx synths are instantiated per event and freed after the event (same for buses). This has the consequence that the user has to think about possible delay lines of effects, clearly the cleanup for each event has then to be delayed too. All cleanup is done automatically but it's the user's responsibility to pass appropriate maximum cleanup delay times. See the chapter "Principle of operation" in PbindFx's help file.

- Similarily, delayed entries have to be considered resp. compensated too, typical example: sequence of events with fxs: echo – non-echo – echo.

- PbindFx does a lot of resource management behind the scenes but requires the user to stick to a number of conventions. The most relevant ones are concerning fx SynthDef ins and outs, additional bus args of SynthDefs and special cases of multichannel handling, see the chapter "Principle of operation" in PbindFx's help.

kitchen studies – a multifold project (2017-19)

- **Electroacoustic composition**
  based on a short sample, using a specific kind of granular synthesis

- **Publication of software**
  Classes written in SuperCollider (SC), source code of composition,
  both part of SC quark extension *miSCellaneous*

- **Artistic research project**
  published online in the database *Research Catalogue* (RC)
  http://researchcatalogue.net
  Search for *kitchen studies*

- **Research paper**
  Daniel Mayer. 2019. PbindFx: an interface for sequencing effect graphs in the SuperCollider audio programming language. In *Proceedings of the 14th International Audio Mostly Conference: A Journey in Sound* (*AM'19*). Association for Computing Machinery, New York, NY, USA, 287–291. DOI:
  https://doi.org/10.1145/3356590.3356639

Granulation with *PbindFx* in *kitchen studies*

Audio source (kitchen sound):

Short version (Part 4 omitted):

Part 1: Comb delay + phase modulation (seq, param sequencing)

Part 2: Rectangular comb (FFT fx, param sequencing)

Part 3: Resampling (param sequencing)

Part 4: Spectral complementing (FFT fx, param sequencing)

Part 5: Frequency shift with feedback (param sequencing)

Part 6: Bandpass (param sequencing)

Links

http://daniel-mayer.at

→ Software → miSCellaneous_lib
→ Works → *kitchen studies, Lokale Orbits, Matters*

https://github.com/dkmayer/miSCellaneous_lib

http://researchcatalogue.net

Search for → *kitchen studies*

http://supercollider.github.io