Buffer Granulation

different approaches to buffer granulation with gui examples

Extension

Description

As with many things in SC work can be done by language or server. Speaking about granulation in general this mainly concerns the control of single grains, their rate, timing, length and other parameters. Regarding buffer granulation specifically this task can e.g. be taken over by ugens like TGrains and GrainBuf, the latter additionally accepting a grain envelope arg. The SC plugin distribution contains further variations (see BhobUGens, JoshUGens). By using granulation ugens in a SynthDef granular textures can be produced with a single Synth, including grain parameter sequencing with demand rate ugens, the DX suite opens additional genuine options in this regard. Alternatively SynthDefs for single grains can be defined with PlayBuf or BufRd in order to control the whole buffer granulation process from language side, using Patterns, Tasks or Routines. What is the best way? To a large extent this is a question of personal preference. Regarding CPU performance granulation ugens have an advantage, whereas e.g. pattern-driven granulation allows concise control over the sequencing of granulation parameters in a clear syntax. If pattern-driven granulation is becoming CPU-critical you might want to consider the event type \grain, a lightweight variant of default type \note (see comment in the source file Event.sc). For granulation with Tasks see VarGui, Ex.3.

Also hybrid strategies are possible, e.g. language controlled setting of a single granulation Synth or involving extra control synths in a language-driven granulation process, further options are Pspawner / Pspawn and Wavesets. VarGui can be used to integrate these setups in single GUIs. Together with this file (miSCellaneous v0.7) I reinvented a color grouping option that overrides automatic color grouping (VarGui, Ex.7). The latter is based on the logical structure of ordinary and array controls, synths and environments. This is useful for VarGuis up to a medium number of controls per Synth / Pattern / Task and also for a large number of such items, but it doesn't handle cases well where there are many controls per item. Not barely an aesthetic detail, it is much more convenient for experimenting to group all sliders of a Synth that, say, have to do with a bandpass filter, within one color.

Types of variables

In general interpreter variables are prefered in examples below, wherever possible. If evaluated with example code only their values are passed and further changing of used variables doesn't affect already generated gui instances. On the contrary repeated evaluation of an interpreter variable, e.g. from a Pfunc, is unsafe - variable's value could change while gui has not yet been closed - so in concerned examples values are passed to the event definition, in Ex.2c a variable is declared for the same reason. The use of environmental variables is following the way VarGui is handling them. Per default every EventStreamPlayer derived from a passed Pattern is run in a separate newly generated Environment, where variables are being set and Streams from Pfuncs and PLx patterns are reading from. See Event patterns and Functions, PLx suite and VarGui.

WARNING:

1. Be careful with amplitudes, especially with buffers you haven't granulated before! Also keep in mind that a granular cloud moving through a buffer can suddenly become louder and other controls than amp (e.g. buffer position, trigger rate, bandpass parameters) can cause a raise of amplitude too.
2. I haven't used below setups for live performances. Although all of them work stable for me as they are, in general hangs can occasionally happen with pattern-driven setups. Often this can be tracked down to sequences of extremely short event durations (and/or long grain durations). Where this can happen as a side-effect, thresholds can be built in, e.g. Ex.2d (Wavesets) has a parameter maxTrigRate. Another possible source of hangs is careless deep nesting of Patterns where mistakes can easily occur. Starting with clear Pattern structures is recommended - and if more complications are involved: testing without sound first, after saving your patch, might be a good idea.

NOTE: All variants from this tutorial can be applied to a buffer, which is occasionally (or continuously) filled with live input. Vice versa variants from Live Granulation can of course be applied to any signal, thus also to any playback of a buffer.

All examples below expect mono buffers. Buffer paths refering to the included sample suppose that you have installed via quarks or moved miSCellaneous lib into the user or system extensions directory directly (not into a subfolder), if not so or you have moved the sample file somewhere else you'd have to change paths accordingly.

Due to a bug in SC 3.7 / 3.8 TGrains didn't response to amp changes, I changed the examples accordingly, the bug is fixed in 3.9.

While other parts of miSCellaneous lib were running fine I was unable to allocate buffers with the SC 3.5.4 Windows binary in August 2012. Meanwhile this issue has been solved (SC 3.6.6, February 2014). There also seemed to be accuracy issues with OffsetOut on Windows with SC 3.6 still, but not with newer versions.

Credits

Thanks for contributions and inspirations by SCers Alberto de Campo (Wavesets), James Harkins (Patterns), Ron Kuivila (Pspawner), Sergio Luque (stochastic distributions), Josh Parmenter (granulation plugins), Bhob Rainey (granulation plugins).

References

1. de Campo, Alberto. "Microsound" In: Wilson, S., Cottle, D. and Collins, N. (eds). 2011. The SuperCollider Book. Cambridge, MA: MIT Press, 463-504.
2. Luque, Sergio (2006). Stochastic Synthesis, Origins and Extensions. Institute of Sonology, Royal Conservatory, The Netherlands. http://sergioluque.com
3. Roads, Curtis (2001). Microsound. Cambridge, MA: MIT Press.
4. Seidl, Fabian (2016). Granularsynthese mit Wavesets für Live-Anwendungen. Master Thesis, TU Berlin. http://www2.ak.tu-berlin.de/~akgroup/ak_pub/abschlussarbeiten/2016/Seidl_MasA.pdf
5. Wishart, Trevor (1994). Audible Design. York: Orpheus The Pantomime Ltd.

6. Xenakis, Iannis (1992). Formalized Music. Hillsdale, NY: Pendragon Press, 2nd Revised edition.

1. Granulation with Ugens

Ex.1a: Basic buffer granulation Synth

```
(
s = Server.local;
Server.default = s;
s.boot;
)

// basic SynthDef suited for pitch-shift and time-stretch
// buffer position given relatively (posLo and posHi between 0 and 1)
// posDev: maximum amount of deviation from (moving) grain center position
// posDev = 0 can lead to comb filter effects (which may be nice sometimes)

// passing control specs as metadata allows for VarGui shortcut build method sVarGui
// metadata specs can be overwritten by arg ctrReplace
// alternatively control specs may be passed as synthCtr arg to a build with VarGui( ...
)

(
SynthDef(\gran_1a, { arg out = 0, bufNum = 0, posLo = 0.0, posHi = 1.0,
    posRate = 1, posDev = 0.01, trigRate = 100, granDur = 0.1, rate = 1.0,
    panMax = 1, amp = 0.1, interp = 4;

    var trig, pan, pos, bufDur, bufDurSection, posDif;

    posDif = posHi - posLo;
    bufDur = BufDur.kr(bufNum);
    bufDurSection = bufDur * posDif;
    trig = Impulse.kr(trigRate);
    pos = posLo * bufDur +
        (Phasor.ar(0, BufRateScale.kr(bufNum) * posRate / SampleRate.ir, posLo * bufDur,
posHi * bufDur) +
        (TRand.kr(-0.5 * posDev, 0.5 * posDev, trig) * bufDur)).mod(bufDurSection);
    pan = Demand.kr(trig, 0, Dseq([panMax, panMax.neg],inf) * 0.999);
    Out.ar(out, TGrains.ar(2, trig, bufNum, rate, pos, granDur, pan, 1, interp) * amp);
    }, metadata: (
        specs: (
            posLo: [0.01, 0.99, \lin, 0.01, 0],
            posHi: [0.01, 0.99, \lin, 0.01, 1],
            posRate: [0.1, 2, \lin, 0.01, 1],
            posDev: [0, 0.2, 5, 0, 0.01],
            granDur: [0.01, 0.3, \lin, 0.01, 0.1],
            trigRate: [1, 200, \lin, 0.01, 100],
            rate: [0.1, 2, \lin, 0.01, 1],
            panMax: [0.0, 1, \lin, 0.005, 0.8],
            amp: [0.0, 0.5, \lin, 0.005, 0.25]
        )
    )
).add;

b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
```

```
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.

)


// start from GUI

\gran_1a.sVarGui([\bufNum, b.bufnum]).gui;
```

Ex.1b: More deviations

```
// In example 1a only a deviation from the grain center position was implemented.
// With additional deviation controls a greater plasticity of sound can be achieved,
// here deviations are added for trigRate (LFO with oscillation freq and deviation max),
// grain duration and rate (TRand, equally weighted random deviation with given max).
// Deviations intervals could be defined alternatively,
// e.g. (1/(1+maxDev), 1+maxDev) with 0 < maxDev
// instead of (1-maxDev, 1+mexDev) with 0 < maxDev < 1

// posRate control range is widened by inventing two controls for
// mantissa and exponent, so posRate = 1 for init param pair
// posRateE = 0 and posRateM = 1

(
SynthDef(\gran_1b, { arg out = 0, bufNum = 0, posLo = 0.0, posHi = 1.0,
    posRateE = 0, posRateM = 1, posDev = 0.01, trigRate = 100, trigRateDev = 0,
    trigRateOsc = 1, granDur = 0.1, granDurDev = 0, rate = 1.0, rateDev = 0,
    panMax = 1, amp = 0.1, interp = 4;

    var trig, pan, pos, bufDur, bufDurSection, posDif, posRate;

    posDif = posHi - posLo;
    bufDur = BufDur.kr(bufNum);
    bufDurSection = bufDur * posDif;
    trig = Impulse.kr(LFDNoise3.kr(trigRateOsc, trigRate * trigRateDev, trigRate));
    posRate = 10 ** posRateE * posRateM;
    pos = posLo * bufDur +
        (Phasor.ar(0, BufRateScale.kr(bufNum) * posRate / SampleRate.ir, posLo * bufDur,
posHi * bufDur) +
        (TRand.kr(-0.5, 0.5, trig) * posDev * bufDur)).mod(bufDurSection);
    pan = Demand.kr(trig, 0, Dseq([panMax, panMax.neg],inf) * 0.999);
    Out.ar(out, TGrains.ar(2, trig, bufNum, rate * (TRand.kr(-1, 1.0, trig) * rateDev +
1), pos,
        granDur * (TRand.kr(-1, 1.0, trig) * granDurDev + 1), pan, 1, interp) * amp);
    }, metadata: (
        specs: (
            posLo: [0.01, 0.99, \lin, 0.01, 0],
            posHi: [0.01, 0.99, \lin, 0.01, 1],
            posRateE: [-3, 4, \lin, 1, 0],
            posRateM: [0.1, 10, \exp, 0.01, 1],
            posDev: [0, 0.2, 5, 0, 0.05],
            trigRate: [1, 200, \lin, 0.01, 100],
            trigRateDev: [0.0, 1, \lin, 0.01, 0],
            trigRateOsc: [0.1, 2, \lin, 0.01, 3],
            granDur: [0.01, 0.3, \lin, 0.01, 0.1],
            granDurDev: [0.0, 0.95, \lin, 0.01, 0],

            rate: [0.1, 2, \lin, 0.01, 1],
            rateDev: [0.0, 0.99, \linear, 0.01, 0.05],
            panMax: [0.0, 1, \lin, 0.005, 0.8],
```

```
            amp: [0.0, 0.5, \lin, 0.005, 0.25]
        )
    )
).add;

b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.
)


// start from GUI
// use color grouping for better overview

\gran_1b.sVarGui([\bufNum, b.bufnum]).gui(synthColorGroups:
(0..14).clumps([1,5,3,2,2,1,1]) )
```

Ex.1c: Buffer granulation Synth with external control synths

```
// This is a more modular approach, once having a basic granulation synth
// it can be linked with arbitrary control synths, here
// also moving through the buffer is controlled externally.
// Depending on LFOs it might be worth defining
// audio buses for control to get higher precision.


(
SynthDef(\gran_1c, { arg out = 0, bufNum = 0, amp = 0.1, pos = 0.5, posDev = 0.01,
    trigRate = 100, granDur = 0.1, rate = 1, panMax = 1, interp = 4;
    var trig, pan;
    trig = Impulse.kr(trigRate);
    pan = Demand.kr(trig, 0, Dseq([panMax, panMax.neg],inf) * 0.999);
    pos = (pos * BufDur.kr(bufNum) * WhiteNoise.kr(posDev, 1));
    Out.ar(out, TGrains.ar(2, trig, bufNum, rate, pos, granDur, pan, 1, interp) * amp);
}).add;


// LFO synthdef for switching between 3 types:
// 0: LFDNoise3 (smooth random movement)
// 1: SinOsc
// 2: LFSaw (works as phasor for position)

SynthDef(\lfo, { |out = 0, lfoType = 0, freq = 1, lo = 0, hi = 1|
    var ctrl;
    ctrl = Select.kr(lfoType, [
        LFDNoise3,
        SinOsc,
        LFSaw
    ].collect { |x| x.kr(freq, mul: hi-lo/2, add: hi+lo/2) });
    Out.kr(out, ctrl);
}).add;

// multichannel control bus
c = Bus.control(s, 5);

b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.
)
```

```
// In contrast to Ex. 1a and 1b VarGui is called explicitely
// as control specs should differ.

// Also here the granulation Synth is generated explicitely and not by the interface
// as its controls have to be mapped to buses

(
// start granulation synth paused and register
// to let VarGui know its state

x = Synth.newPaused(\gran_1c, [\bufNum, b]).register;

// map args to be controlled to consecutive subbuses
x.map(*[[\pos, \trigRate, \granDur, \rate, \panMax],
(0..4).collect(c.subBus(_))].flop.flat)
)

(
// Open VarGui interface, granulation synth (#5) is paused
// (orange button on yellow background).
// Shift-clicking one of the green buttons runs
// granulation synth and control synths (#0 - #4).
// All synths can be paused (orange button) and resumed.

// NOTE: When stopping the granulation synth
// the linkage with control synths is lost!
// A new synth generated by the interface
// (by pressing the blue button of #5) is not
// automatically mapped to control buses.
// On the other hand control synths might be stopped and
// newly generated.


VarGui(synthCtr: [[
        \pos, 0,    // dummy spec for labelling
        \out, c.index,
        \lfoType, [0, 2, \lin, 1, 0],
        // as LFOs are unified, original movement tempo is
        // indicated not by 1 but by 1 / buffer duration
        \freq, [0.01, 20, \exp, 0.0, 1/b.duration],
        \lo, [0.0, 1, \lin, 0, 0.1],
        \hi, [0.0, 1, \lin, 0, 0.9]
    ],[
        \trigRate, 1,
        \out, c.index + 1,
        \lfoType, [0, 2, \lin, 1, 0],
        \freq, [0.001, 0.5, \exp, 0.0, 0.2],
        \lo, [1, 100, \lin, 0, 7],
        \hi, [1, 100, \lin, 0, 40]
    ],[
        \granDur, 2,
        \out, c.index + 2,
        \lfoType, [0, 2, \lin, 1, 1],
        \freq, [0.001, 0.5, \exp, 0.0, 0.4],
        \lo, [0.01, 0.2, \lin, 0, 0.1],
        \hi, [0.01, 0.2, \lin, 0, 0.15]
    ],[
        \rate, 3,
        \out, c.index + 3,
        \lfoType, [0, 2, \lin, 1, 0],
        \freq, [0.001, 0.1, \exp, 0.0, 0.03],
        \lo, [0.1, 3, \lin, 0, 0.6],
```

```
        \hi, [0.1, 3, \lin, 0, 1.1]
    ],[
        \panMax, 4,
        \out, c.index + 4,
        \lfoType, [0, 2, \lin, 1, 0],
        \freq, [0.001, 0.5, \exp, 0.0, 0.5],
        \lo, [0.0, 1, \lin, 0, 0.06],
        \hi, [0.0, 1, \lin, 0, 0.9]
    ],[
        \out, 0,
        \bufNum, b.bufnum,
        \posDev, [0.001, 0.2, \exp, 0, 0.02],
        \amp, [0.0, 2, \lin, 0, 0.5]
    ]],
     // 5 lfo synths are generated by the interface (synthdef name passed)
    // but granulation Synth is passed directly as object
    synth: \lfo!5 ++ x
).gui(sliderPriority: \synth, playerPriority: \synth);
)
```

Ex.1d: Buffer granulation Synth with demand rate ugens

```
// Repeated grain triggering within a synth can be defined by demand rate ugens,
// which is comfortable in connection with granular ugens.

// E.g. with TGrains you can trigger parameters like grain duration, playback rate,
// position, panning and amp, single grains will keep their params if they overlap.
// (Note that in the below implementation changes of demand rate array fields will apply
// also not before the next call of those fields in the synth)

// More refined per-grain control, e.g. per-grain filtering with filter parameter
streams,
// can be done by using a multichannel trick, see Ex. 1e.

(
// length of demand rate sequence, you might want to check larger sizes
// in connection with gui arg tryColumnNum > 1
~n = 5;

// posRate control range is widened by inventing two controls for
// mantissa and exponent, so posRate = 1 for init param pair
// posRateE = 0 and posRateM = 1

SynthDef(\gran_1d, { |out = 0, soundBuf, posLo = 0.1, posHi = 0.3,
    posRateE = 0, posRateM = 1, granDurMul = 1, rateMul = 1, panMul = 1, amp = 0.5,
interp = 2|

    var signal, bufDur, granDur, granGate, relGranDurs, pos, overlap, overlaps,
overlapSeq,
        pan, rate, relRates, rateSeq, posRate, granDurSeq;

    // array args for demand rate sequencing, short form of NamedControl
    relGranDurs = \relGranDurs.kr(0.1!~n);
    relRates = \relRates.kr(1!~n);
    overlaps = \overlaps.kr(1!~n);

    // Dstutter (or Dunique) necessary as granDurSeq is polled twice: granGate and
granDur
    granDurSeq = Dstutter(2, Dseq(relGranDurs, inf));
    rateSeq = Dseq(relRates, inf);
    overlapSeq = Dseq(overlaps, inf);

    granGate = TDuty.ar(granDurSeq * granDurMul);
```

```
    granDur = Demand.ar(granGate, 0, granDurSeq * granDurMul);
    rate = Demand.ar(granGate, 0, rateSeq) * rateMul;
    pan = Demand.ar(granGate, 0, Dseq([1, -1], inf)) * 0.999 * panMul;
    overlap = Demand.ar(granGate, 0, overlapSeq);

    bufDur = BufDur.kr(soundBuf);
    posRate = 10 ** posRateE * posRateM;

    pos = Phasor.ar(0, BufRateScale.kr(soundBuf) * posRate / (SampleRate.ir * bufDur),
posLo, posHi);
    signal = TGrains.ar(2, granGate, soundBuf, rate, pos * bufDur, granDur * overlap,
pan, 1, interp);

    Out.ar(out, signal * amp);
}
).add;

b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.
)

(
// relGranDurs are multiplied with granDurMul, analogously relRates with rateMul.
// Changes of these params apply immediately in contrast to the array args
// used by demand rate ugens which are used with next demand.

// check out moving a number of sliders of one array by using
// Shift, Alt + Shift and Ctrl + Shift, see VarGui help Ex. 1c.


VarGui(synthCtr: [
    soundBuf: b.bufnum,

    posLo: [0, 1, \lin, 0, 0.1],
    posHi: [0, 1, \lin, 0, 0.9],
    posRateE: [-3, 4, \lin, -1, 0],
    posRateM: [0.1, 10, \exp, 0.01, 0.5],

    // generating control specifications depending on index
    relGranDurs: { |i| [0.01, 0.1, \lin, 0, i * 0.005 + 0.02] } ! ~n,
    granDurMul: [0.03, 2, \lin, 0, 0.25],
    overlaps: [0.1, 5, \lin, 0, 1.5] ! ~n,

    relRates: { |i| [0.1, 1.5, \lin, 0, (5-i) * 0.1 + 0.5] } ! ~n,
    rateMul: [0.1, 2, \lin, 0, 0.5],

    panMul: [0, 1, \lin, 0, 0.6],
    amp: [0.0, 3, \lin, 0, 2]
],
    synth: \gran_1d
).gui(
    tryColumnNum: 1,
    // color grouping for better overview
    synthColorGroups: (0..(~n*3+8)).clumps([1,4,~n+1,~n,~n+1,1,1]),
    labelWidth: 90,
    sliderWidth: 280
);
)
```

Ex.1e: Buffer granulation Synth with per-grain effect processing (TGrains)

```
// TGrains and other granular ugens allow per-grain processing
// for a limited number of parameters (pos, rate etc.)
// Nevertheless it is possible to apply arbitrary effects with
// per-grain parameter changes, even if grains overlap.
// This can be achieved by defining the granulation output
// as a multichannel signal and appropriate triggering of fx parameters.
// The example is adapted from a recommendation of Julian Rohrhuber.

// The method is elegant but also a bit tricky in terms of
// multichannel triggering and channel routing.
// See Ex.1f for achieving the same with DX ugens

(
// the multichannel size and equivalently:
// the maximum number of overlapping grains that might get
// different fx parameters have to be fixed.
// For convenience of later L/R-spatialization we take an
// even number ~n = 2 * ~m

~m = 5;
~n = 2 * ~m;

SynthDef(\gran_1e, { |out = 0, soundBuf, posLo = 0.1, posHi = 0.9,
    posRateE = 0, posRateM = 1, rate = 1, panMax = 0.8, bpRQ = 0.1, bpLo = 50, bpHi =
5000,
    amp = 1, bpFund = 100, overlap = 2, trigRate = 1, interp = 2|
    var sig, sigL, sigR, bpFreq, chan, bpFreqSeqs, dUgen,
        trig, trigs, bufDur, pos, posRate;

    trig = Impulse.ar(trigRate);
    // we need a multichannel trigger that steps through all consecutive channels
    trigs = { |i| PulseDivider.ar(trig, ~n, ~n-1-i) } ! ~n;

    chan = Demand.ar(trig, 0, Dseq((0..~n-1), inf));

    posRate = 10 ** posRateE * posRateM;
    bufDur = BufDur.kr(soundBuf);
    pos = Phasor.ar(0, BufRateScale.kr(soundBuf) * posRate * SampleDur.ir / bufDur,
posLo, posHi);

    sig = TGrains.ar(~n, trig, soundBuf, rate, pos * bufDur, overlap/trigRate,
        // Panning convention is that from PanAz,
        // speakers should be from 0 to 2, but (orientation)
        // 1/n has to be substracted for n speakers.
        // If this isn't done correctly grains are spread onto more than one channel
        // and per-grain application of fxs fails.
        chan.linlin(0, ~n-1, -1/~n, (2*~n - 3)/~n), 1, interp);

    dUgen = Dwhite(0.0, 1);
    sig = sig.collect { |ch, i|
        // this is the place to define fxs per channel/grain
        // multichannel trigger is polling from a single demand ugen
        bpFreq = Demand.ar(trigs[i], 0, dUgen).linlin(0, 1, bpLo, bpHi);

        // amplitude compensation for lower rq of bandpass filter
        BPF.ar(ch, bpFreq, bpRQ, (bpRQ ** -1) * (400 / bpFreq ** 0.5));
    };

    // routing to two channels ...
    sigL = Mix(((0..(~m-1)) * 2).collect(sig[_]));
    sigR = Mix(((0..(~m-1)) * 2 + 1).collect(sig[_]));
```

```
    // ... in order to have L/R-spreading with panMax as in other examples
    Out.ar(0, Pan2.ar(sigL, panMax.neg) + Pan2.ar(sigR, panMax) * amp)
}).add;


b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.
)


(
VarGui(synthCtr: [
    soundBuf: b.bufnum,
    posLo: [0, 1, \lin, 0, 0.2],
    posHi: [0, 1, \lin, 0, 0.5],
    posRateE: [-3, 4, \lin, -1, -1],
    posRateM: [0.1, 10, \exp, 0.01, 0.8],
    overlap: [0.1, ~n, \lin, 0, 12],
    trigRate: [1, 100, \lin, 0, 45],
    rate: [0.1, 2, \lin, 0, 1],

    bpRQ: [0.05, 1, \lin, 0, 0.25],
    bpLo: [50.0, 5000, \exp, 0, 50],
    bpHi: [50.0, 5000, \exp, 0, 5000],
    panMax: [0, 1, \lin, 0, 0.85],
    amp: [0.0, 3, \lin, 0, 1]
],
    synth: \gran_1e
).gui(
    tryColumnNum: 1,
    synthColorGroups: (0..12).clumps([1,4,2,1,3,1,1])
);
)
```

Ex.1f: Buffer granulation Synth with per-grain effect processing (DXEnvFan)

```
// DXEnvFan generates a multichannel envelope which can be used as trigger for
granulation and fxs on grains.
// It encapsulates the trigger logic, which has been used explicitely in Ex. 1e.
// DX ugens can be used for a variety of microsound techniques, see their help files.

(
~maxOverlap = 12;
a = Bus.audio(s, ~maxOverlap);

// overlap only settable in SC versions >= 3.9

SynthDef(\gran_1f, { |out = 0, soundBuf, bus = 0, posLo = 0.1, posHi = 0.9,
    posRateE = 0, posRateM = 1, overlap = 2, trigRate = 1, rate = 1,
    bpRQ = 0.1, bpLo = 50, bpHi = 5000, panMax = 0.8, amp = 1|
    var sig, bpFreq, dUgen, bufDur, pos, posRate, playbuf, env, maxOverlap = ~maxOverlap;

    posRate = 10 ** posRateE * posRateM;
    bufDur = BufDur.kr(soundBuf);
    pos = Phasor.ar(0, BufRateScale.kr(soundBuf) * posRate * SampleDur.ir / bufDur,
posLo, posHi);

    // multichannel trigger
```

```
    env = DXEnvFan.ar(
        Dseq((0..maxOverlap-1), inf),
        trigRate.reciprocal,
        size: maxOverlap,
        maxWidth: maxOverlap,
        width: (Main.versionAtLeast(3, 9)).if { overlap }{ 2 },
        // option to avoid unwanted triggers
        zeroThr: 0.002,
        // take equalPower = 0 for non-squared sine envelopes
        // more efficient with helper bus
        equalPower: 0,
        bus: a
    );
    // multichannel playback, pos is triggered for each grain
    playbuf = PlayBuf.ar(1, soundBuf, rate, env, pos * BufFrames.ir(soundBuf), 1);

    dUgen = Dwhite(0, 1);
    // multichannel trigger used to poll values from drate ugen
    bpFreq = Demand.ar(env, 0, dUgen).linlin(0, 1, bpLo, bpHi);

    // generate grains by multiplying with envelope
    sig = playbuf * env;

    // different frequency on each grain channel
    sig = BPF.ar(sig, bpFreq, bpRQ, (bpRQ ** -1) * (400 / bpFreq ** 0.5));

    // generate array of 5 stereo signals
    sig = Pan2.ar(sig, Demand.ar(env, 0, Dseq([-1, 1], inf) * panMax));

    // mix to out
    Out.ar(0, Mix(sig) * amp)
}, metadata: (
    specs: (
        posLo: [0.01, 0.99, \lin, 0.01, 0],
        posHi: [0.01, 0.99, \lin, 0.01, 0.5],
        posRateE: [-3, 4, \lin, 1, -1],
        posRateM: [0.1, 10, \exp, 0.01, 1.35],
        trigRate: [1, 200, \lin, 0.01, 90],
        overlap: [0.2, 12, \lin, 0.01, 7],
        rate: [0.1, 2, \lin, 0.01, 0.75],
        panMax: [0.0, 1, \lin, 0.005, 0.75],
        bpLo: [100, 5000, \lin, 0, 300],
        bpHi: [100, 5000, \lin, 0, 3000],
        bpRQ: [0.05, 1, \lin, 0, 0.18],
        amp: [0.0, 3, \lin, 0.005, 1]
    )
)).add;

b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.
)

(
\gran_1f.sVarGui([\soundBuf, b.bufnum]).gui(
    tryColumnNum: 1,
    synthColorGroups: (0..12).clumps([1,4,2,1,3,1,1])
)
)
```

Ex.1g: Buffer granulation with (half) wavesets: ZeroXBufRd

```
// movement through the buffer of zero crossings
// with a slow pos rate we get repetitions of half wavesets

// load sound buffer

// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.

b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");


// allocate zeroX buffer

(
z = Buffer.alloc(s, b.duration * 44100 / 5, 1);
s.scope;
)


// write zero crossings, but no need to overwrite sound buffer
// this is caused by setting adjustZeroXs to -1

(
{
    // use LeakDC to avoid extremely long half wavesets
    var src = LeakDC.ar(PlayBuf.ar(1, b, BufRateScale.ir(b)));
    ZeroXBufWr.ar(src, b, z, adjustZeroXs: -1, doneAction: 2);
}.play
)


// instead adjust from lang and get zeroXs

b.adjustZeroXs(z, { |z| ~zeroXs = z.reject(_==0) })

// get number of zeroXs

~zeroXNum = ~zeroXs.size


(
SynthDef(\gran_1g, { |out = 0, soundBuf, zeroXBuf, zeroXNum, posLo = 0.1, posHi = 0.9,
    posRateE = 0, posRateM = 1, rate = 1, amp = 1|
    var sig, bufDur, pos, posRate;

    posRate = 10 ** posRateE * posRateM;
    bufDur = BufDur.kr(soundBuf);

    // move through the buffer of zero crossings
    // the tempo (rate) refers to the sound buffer
    pos = Phasor.ar(
        0,
        BufRateScale.kr(soundBuf) * posRate * SampleDur.ir / bufDur,
        posLo,
        posHi
    ) * zeroXNum;

    sig = ZeroXBufRd.ar(
        soundBuf,
        zeroXBuf,
```

```
        0!2,
        pos,
        mul: amp,
        rate: rate * [1, 1.01]
    );
    Out.ar(0, sig * amp)
}, metadata: (
    specs: (
        posLo: [0.01, 0.99, \lin, 0.01, 0.1],
        posHi: [0.01, 0.99, \lin, 0.01, 0.5],
        posRateE: [-2, 2, \lin, 1, -1],
        posRateM: [0.1, 10, \exp, 0.01, 3],
        rate: [0.3, 3, \lin, 0.01, 1],
        amp: [0.0, 2, \lin, 0.005, 1]
    )
)
).add;

\gran_1g.sVarGui([\soundBuf, b.bufnum, \zeroXBuf, z.bufnum, \zeroXNum, ~zeroXNum]).gui
)
```

Ex.1h: Buffer granulation with (half) wavesets: TZeroXBufRd

```
// buffer preparations from Ex. 1g

// again a movement through the buffer of zero crossings is implemented
// the repetition of wavesets though is defined by xNum and xRep

// with high trigger rates, xNum and xRep values and low rates you might
// have to increase overlapSize, see TZeroXBufRd help for a discussion of this topic

(
SynthDef(\gran_1h, { |out = 0, soundBuf, zeroXBuf, zeroXNum, posLo = 0.1, posHi = 0.9,
    posRateE = 0, posRateM = 1, xNum = 1, xRep = 1, rate = 1, trigRate = 100, amp = 1|
    var sig, bufDur, pos, posRate;

    posRate = 10 ** posRateE * posRateM;

    bufDur = BufDur.kr(soundBuf);
    pos = Phasor.ar(
        0,
        BufRateScale.kr(soundBuf) * posRate * SampleDur.ir / bufDur,
        posLo,
        posHi
    )  * zeroXNum;

    // move through the buffer of zero crossings
    // the tempo (rate) refers to the sound buffer
    sig = TZeroXBufRd.ar(
        soundBuf,
        zeroXBuf,
        0!2, // bufMix arg triggers stereo
        trig: Impulse.ar(trigRate),
        zeroX: pos,
        xNum: xNum,
        xRep: xRep,
        mul: amp,
        rate: rate * [1, 1.01], // a bit of decorrelation
        overlapSize: 20 // larger overlapSize
    );
    // by overlapping a DC can easily accumulate, so do leak
```

```
    Out.ar(0, LeakDC.ar(sig) * amp)
}, metadata: (
    specs: (
        posLo: [0.01, 0.99, \lin, 0.01, 0.1],
        posHi: [0.01, 0.99, \lin, 0.01, 0.5],
        posRateE: [-2, 2, \lin, 1, -1],
        posRateM: [0.1, 10, \exp, 0.01, 1],
        rate: [0.3, 2, \lin, 0.01, 1],
        trigRate: [5, 120, \lin, 0, 50],
        xNum: [1, 5, \lin, 1, 2],
        xRep: [1, 5, \lin, 1, 3],
        amp: [0.0, 2, \lin, 0, 1]
    )
)
).add;

\gran_1h.sVarGui([\soundBuf, b.bufnum, \zeroXBuf, z.bufnum, \zeroXNum, ~zeroXNum]).gui
)
```

2. Granulation driven by language

Ex.2a: Basic buffer granulation Pbind

> **NOTE:** Language-driven sequencing is not sample-exact in realtime (with NRT synthesis it is). This is related to hardware control and cannot be overcome currently. However for most practical purposes this might not be relevant. It is anyway a far less strong effect than the inaccuracies related to Out.ar and the combination of OffsetOut.ar and In.ar (see examples 2a-d in Live Granulation)

```
// Control parameters like in Ex.1a, but implemented with a SynthDef for
// playing single grains and an appropriate Pbind,
// OffsetOut used for exact timing.

(
SynthDef(\gran_2a, { |out = 0, pos = 0, sndBuf = 0, windowBuf = 1, granDur = 0.1,
    rate = 1, loop = 1, panMax = 0, amp = 1|
    var window, src;
    src = PlayBuf.ar(1, sndBuf, BufRateScale.kr(sndBuf) * rate,
        1, round(pos * BufFrames.kr(sndBuf)), loop, 2);
    window = BufRd.ar(1, windowBuf,
        EnvGen.ar(Env([0, BufFrames.kr(windowBuf)], [granDur]),
        doneAction: 2), loop, 4);
    OffsetOut.ar(out, Pan2.ar(src, panMax, amp) * window);
}).add;

b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.

w = Buffer.sendCollection(s, Signal.hanningWindow(1024));
)


// Determining the correct buffer position depending on
```

```
// posRate, posLo and posHi needs a little calculation.
// In 1a this was done inside the ugen by a Phasor.
// See Ex.3b, 3c for doing position movement with a separate Synth.

// PL placeholder patterns used, could also be Pfunc { ~ ... }

(
p = Pbind(
    \instrument, \gran_2a,
    \sndBuf, b,
    \windowBuf, w,

    \dur, 1 / PL(\trigRate),
    \granDur, PL(\granDur),
    \time, Ptime(),
    \pos, Pfunc { |e|
        var relTime = ~posRate * e.time / e.sndBuf.duration, relDif;
        relDif = ~posHi - ~posLo;
        relTime + rand2(~posDev) % relDif + ~posLo;
    },
    \rate, PL(\rate),
    \amp, PL(\amp),
    \panMax, PLseq([-1,1]) * PL(\panMax),
    \out, 0
);

VarGui([
    \posLo, [0.0, 0.99, \lin, 0.01, 0],
    \posHi, [0.0, 0.99, \lin, 0.01, 1],
    \posRate, [0.1, 2, \lin, 0.01, 1],
    \posDev, [0, 0.2, 5, 0, 0.01],
    \trigRate, [1, 200, \lin, 0.01, 120],
    \granDur, [0.01, 0.3, \lin, 0.005, 0.06],
    \rate, [0.1, 3, \lin, 0.01, 1],
    \panMax, [0.0, 1, \lin, 0.0, 0.8],
    \amp, [0.0, 1, \lin, 0.01, 0.25]
    ], stream: p
).gui(varColorGroups: (0..8).clumps([4,1,1,1,1,1]))
)
```

Ex.2b: Switching between stochastic distributions

```
// Extended basic SynthDef from Ex.2a with bandpass filter

(
SynthDef(\gran_2b, { |out = 0, pos = 0, sndBuf = 0, windowBuf = 1, granDur = 0.1,
    rate = 1, loop = 1, panMax = 0, amp = 1, bpFreq = 500, bpRQ = 0.5, bpWet = 1|
    var window, granSrc, src;
    granSrc = PlayBuf.ar(1, sndBuf, BufRateScale.kr(sndBuf) * rate,
        1, round(pos * BufFrames.kr(sndBuf)), loop, 2);
    window = BufRd.ar(1, windowBuf,
        EnvGen.ar(Env([0, BufFrames.kr(windowBuf)], [granDur]),
        doneAction: 2), loop, 4);
    // do amplitude compensation, estimation like in Wavesets example by Alberto de Campo
    src = (BPF.ar(granSrc, bpFreq, bpRQ, mul: (bpRQ ** -1) * (400 / bpFreq ** 0.5)) *
        bpWet + (granSrc * (1 - bpWet)));

    OffsetOut.ar(out, Pan2.ar(src, panMax, amp) * window);
}).add;
```

```
b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.

w = Buffer.sendCollection(s, Signal.hanningWindow(1024));
)


// random types

// 0: low value
// 1: low or high value, evenly distributed
// 2: evenly distributed
// 3: linear decrease from mean value
// 4: exponential distribution
// 5: beta distribution, default parameter 0.3 centers value at the borders
// 6: brownian movement (of first order)
// 7: brownian movement of second order (stepsize itself generated by brownian movement)

// A second order brownian movement much more tends to get stuck at the
// borders than a normal (first order) brownian movement
// E.g. see Sergio Luque's presentation of Xenakis's stochastic synthesis:
// "Stochastic Synthesis, Origins and Extensions", pp 25-28
// http://sergioluque.com


// Function that generates an array of PLx patterns
// of different random types (see PLx suite).
// These placeholders can refer to environmental variables
// to be set by the VarGui interface later on.

(
d = { |keyLo, keyHi, betaProb = 0.3, brownStepFac = 0.01,
    brown2Ratio = 1, brown2StepFac = 0.01|
    // keyLo and keyHi must be Symbols,
    // other args may be Symbols
    var patLo, patHi, patDif;

    // avoid lo-hi reversing with Pbrown
    patLo = min(PL(keyLo), PL(keyHi));
    patHi = max(PL(keyLo), PL(keyHi));
    patDif = patHi - patLo;
    [
        PL(keyLo),
        Pfunc { currentEnvironment[[keyLo, keyHi].choose] },
        PLwhite(keyLo, keyHi),
        PLmeanrand(keyLo, keyHi),
        PLexprand(keyLo, keyHi),
        PLbeta(keyLo, keyHi, betaProb, betaProb),
        PLbrown(patLo, patHi, patDif * PL(brownStepFac)),
        PLbrown(patLo, patHi,
            PLbrown(
                patDif.neg * PL(brown2Ratio) / 2,
                patDif * PL(brown2Ratio) / 2,
                patDif * PL(brown2Ratio) * PL(brown2StepFac)
            )
        )
    ]
};
)

// trigrate, granDur, rate and bpFreq are
```

```
// chosen between bounds according to the
// random distribution type notated with suffix D

// single grains are filtered with a bandpass
// amount of effect controlled with bpWet

(
p = Pbind(
    \instrument, \gran_2b,
    \sndBuf, b,
    \windowBuf, w,

    \dur, 1 / PLswitch1(d.(\trigRateLo, \trigRateHi), \trigRateD),
    \granDur, PLswitch1(d.(\granDurLo, \granDurHi), \granDurD),
    \time, Ptime(),
    \posRate, PL(\posRate),
    \pos, Pfunc { |e|
        var relTime = ~posRate * e.time / e.sndBuf.duration, relDif;
        relDif = ~posHi - ~posLo;
        relTime + rand2(~posDev) % relDif + ~posLo;
    },
    \rate, PLswitch1(d.(\rateLo, \rateHi), \rateD),
    \bpFreq, PLswitch1(d.(\bpFreqLo, \bpFreqHi), \bpFreqD),
    \bpRQ, PL(\bpRQ),
    \bpWet, PL(\bpWet),

    \amp, PL(\amp),
    \panMax, PLseq([-1,1]) * PL(\panMax),
    \out, 0
);

VarGui([
    \posLo, [0.0, 0.99, \lin, 0.01, 0.21],
    \posHi, [0.0, 0.99, \lin, 0.01, 0.47],
    \posRate, [0.1, 2, \lin, 0.01, 0.2],
    \posDev, [0, 0.2, 5, 0, 0.002],

    \trigRateLo, [1, 200, \lin, 0.01, 21],
    \trigRateHi, [1, 200, \lin, 0.01, 155],
    \trigRateD, [0, 7, \lin, 1, 6],

    \granDurLo, [0.01, 0.6, \exp, 0.0, 0.037],
    \granDurHi, [0.01, 0.6, \exp, 0.0, 0.4],
    \granDurD, [0, 7, \lin, 1, 6],

    \rateLo, [0.1, 3, \lin, 0.01, 1.09],
    \rateHi, [0.1, 3, \lin, 0.01, 1.63],
    \rateD, [0, 7, \lin, 1, 1],

    \bpFreqLo, [50, 10000, \exp, 0.1, 54],
    \bpFreqHi, [50, 10000, \exp, 0.1,8275],
    \bpFreqD, [0, 7, \lin, 1, 1],
    \bpRQ, [0.01, 0.99, \lin, 0.0, 0.07],
    \bpWet, [0.0, 1, \linear, 0.0, 0.23],

    \panMax, [0.0, 1, \lin, 0.0, 0.85],
    \amp, [0.0, 1, \lin, 0.01, 0.25]
    ], stream: p
).gui(varColorGroups: (0..19).clumps([4,3,3,3,5,1,1]))
)
```

Ex.2c: Generating granular phrases with Pspawner

```
// This example needs SynthDef \gran_2b and Function d to be taken from Ex. 2b

(
b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.

w = Buffer.sendCollection(s, Signal.hanningWindow(1024));
)

// A simple form of Pspawner is used to generate phrases.
// Phrase length params are taken a bit roughly as sustain and
// rest times also depend on randomly varying grain lengths.
// spSustain controls medium sustain time (without grain length overhead)
// spLegato controls medium legato factor (disregarding reduction by grain length
overhead)
// spDev is causing separate random deviation of spSustain and spLegato
// between 1/(1+spDev) and 1+spDev

// random distribution switching is restricted here to
// types 6 and 7 (random walks of first and second order)
// to force individual sound qualities of phrases.

(
// declare var here as pattern is repeatedly evaluated from within the Pspawner,
// interpreter variable would be unsafe if running examples in parallel

var p = Pbind(
    \instrument, \gran_2b,
    \sndBuf, b,
    \windowBuf, w,

    \dur, 1 / PLswitch1(d.(\trigRateLo, \trigRateHi), \trigRateD),
    \granDur, PLswitch1(d.(\granDurLo, \granDurHi), \granDurD),
    \time, Ptime(),
    \posRate, PL(\posRate),

    // random timeOffset added with each spawning
    \pos, Pfunc { |e|
        var relTime = ~posRate * e.time / e.sndBuf.duration + e.timeOffset, relDif;
        relDif = ~posHi - ~posLo;
        relTime + rand2(~posDev) % relDif + ~posLo;
    },
    \rate, PLswitch1(d.(\rateLo, \rateHi), \rateD),
    \bpFreq, PLswitch1(d.(\bpFreqLo, \bpFreqHi), \bpFreqD),
    \bpRQ, PL(\bpRQ),
    \bpWet, PL(\bpWet),

    \amp, PL(\amp),
    \panMax, PLseq([-1,1]) * PL(\panMax),
    \out, 0
);

q = Pspawner({ |sp|
    var randomizer = { |x| var y = rand(x); 0.5.coin.if { 1 + y }{ 1 / (1 + y) } },
        sus, legato, delta;

    loop {
        sus = ~spSustain * randomizer.(~spDev);
        legato = ~spLegato * randomizer.(~spDev);
```

```
        // take random offset for each phrase
        sp.par(Pfindur(sus, Psetpre(\timeOffset, 5.0.rand, p)));
        delta = sus / (~spLegato * randomizer.(~spDev));
        sp.wait(delta)
    }
});

VarGui([
    \posLo, [0.0, 0.99, \lin, 0.01, 0.16],
    \posHi, [0.0, 0.99, \lin, 0.01, 0.41],
    \posRate, [0.1, 2, \lin, 0.01, 1.4],
    \posDev, [0, 0.2, 5, 0, 0.0017],

    \trigRateLo, [1, 200, \lin, 0.01, 70],
    \trigRateHi, [1, 200, \lin, 0.01, 150],
    \trigRateD, [6, 7, \lin, 1, 7],

    \granDurLo, [0.01, 0.6, \exp, 0.0, 0.02],
    \granDurHi, [0.01, 0.6, \exp, 0.0, 0.11],
    \granDurD, [6, 7, \lin, 1, 6],

    \rateLo, [0.1, 3, \lin, 0.01, 0.2],
    \rateHi, [0.1, 3, \lin, 0.01, 1.86],
    \rateD, [6, 7, \lin, 1, 7],

    \bpFreqLo, [50, 10000, \exp, 0.1, 67],
    \bpFreqHi, [50, 10000, \exp, 0.1, 5885],
    \bpFreqD, [6, 7, \lin, 1, 6],
    \bpRQ, [0.01, 0.99, \lin, 0.0, 0.17],
    \bpWet, [0.0, 1, \linear, 0.0, 0.38],

    \spSustain, [0.2, 2, \linear, 0.0, 0.884],
    \spLegato, [0.6, 1.2, \linear, 0.0, 0.996],
    \spDev, [0.0, 1, \linear, 0.0, 0.41],

    \panMax, [0.0, 1, \lin, 0.0, 0.85],
    \amp, [0.0, 1, \lin, 0.01, 0.35]
    ], stream: q
).gui(varColorGroups: (0..22).clumps([4,3,3,3,5,3,1,1]) )
)
```

Ex.2d: Wave sets

```
// For this example you need Alberto de Campo's Wavesets class
// (Quark extension, implementation following definitions of Trevor Wishart)
// and the Function d from Ex.2b.

// the wave set player synth optionally adds a BPF applied to the signal,
// amount can be controlled with bpWet
// attack and release time > 0 for smoothening

(
SynthDef(\wsPlayer, { arg out = 0, buf = 0, start = 0, length = 441,
    rate = 1, att = 0.03, rel = 0.03, wvDur = 1, panMax = 0, amp = 0.2,
    delayL = 0.0, delayR = 0.0, bpFreq = 500, bpRQ = 0.5, bpWet = 1;
    var phasor, env, granSrc, src, attEff, relEff, sus;

    phasor = Phasor.ar(0, BufRateScale.ir(buf) * rate, 0, length) + start;
    attEff = min(att, wvDur/2);
    relEff = min(rel, wvDur/2);
```

```
    sus = wvDur - attEff - relEff;

    env = EnvGen.ar(Env([0, 1, 1, 0], [attEff, sus, relEff], \sine), doneAction: 2);
    granSrc = BufRd.ar(1, buf, phasor);
    src = (BPF.ar(granSrc, bpFreq, bpRQ, mul: (bpRQ ** -1) * (400 / bpFreq ** 0.5)) *
        bpWet + (granSrc * (1 - bpWet)));
    OffsetOut.ar(out, Pan2.ar(src, panMax, amp) * env);
}).add;

a = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.

b = Buffer.read(s, a);
w = Wavesets.from(a);

// relative positions of zero crossings
x = w.fracXings.drop(-1) / w.numFrames;
)


// Note that trigRate / event duration is not controlled directly in this setup,
// it's derived from wave set length (may vary) and legato factor.
// For short wave sets and a low legato value durs could become so short that
// a sudden mass of grains could cause hangs.
// To avoid this a parameter maxTrigRate is invented.

// As wavesets start at distinguished positions in the buffer and
// - together with legato - durations are determined,
// an additionally given position rate in general cannot result in a "correct"
// looping through the buffer, there will be a deviation.
// Here two types of approximation can be choosen with posType:
// Type 0 takes the waveset nearest to the calculated exact position.
// Type 1 linearly maps positions to wave set indices.
// As wave sets are of different length the latter is a rough heuristic
// accelerating buffer parts with relatively low frequencies.

(
p = Pbind(
    \instrument, \wsPlayer,

    // refering to interpreter variables within Pfuncs
    // when running several examples in parallel is unsafe,
    // so pass them here, then access from within the event

    \b, b,
    \w, w,
    \x, x,

    \time, Ptime(),
    \posLo, PL(\posLo),
    \posHi, PL(\posHi),
    \posRate, PL(\posRate),

    \pos, Pfunc { |e| (e.time * e.posRate / e.b.duration) %
        (e.posHi - e.posLo) + e.posLo },

    // Estimation of ws index from relative position, see explanation above,
    // indexIn might be a bottleneck with large buffers,
    // also a more rough estimation of ws index could be used:
    // \startWs, Pfunc { |e| e.pos.linlin(0, 1, 0, e.w.xings.size - 2).round.asInteger }
```

```
    \startWs, Pfunc { |e|
            (~posType == 0).if {
                e.x.indexIn(e.pos)
            }{
                e.pos.linlin(e.posLo, e.posHi, e.x.indexIn(e.posLo),
e.x.indexIn(e.posHi))
                    .round.asInteger
            }
        },

    \numWs, PLswitch1(d.(\numWsLo, \numWsHi), \numWsD),
    \repeats, PLswitch1(d.(\repeatsLo, \repeatsHi), \repeatsD),

    \bpFreq, PLswitch1(d.(\bpFreqLo, \bpFreqHi), \bpFreqD),
    \bpRQ, PL(\bpRQ),
    \bpWet, PL(\bpWet),

    \rate, PLswitch1(d.(\rateLo, \rateHi), \rateD),
    \data, Pfunc { |e| e.w.frameFor(e.startWs, e.numWs)  },

    \buf, b.bufnum,
    \start, Pkey(\data).collect(_[0]), // startFrame
    \length, Pkey(\data).collect(_[1]), // length (frameNum)
    \wvDur, Pkey(\data).collect(_[2]) * Pkey(\repeats), // sustain time

    \calculatedDur, Pkey(\wvDur) / PLswitch1(d.(\legatoLo, \legatoHi), \legatoD),
    \dur, Pfunc { |e| max(e.calculatedDur, 1 / ~maxTrigRate) },

    \panMax, PLseq([-1,1]) * PL(\panMax),
    \amp, PL(\amp),
    \out, 0
);


VarGui([
    \posLo, [0, 1, \lin, 0.0, 0.15],
    \posHi, [0, 1, \lin, 0.0, 0.45],
    \posRate, [0.0, 2, \lin, 0.01, 0.25],
    \posType, [0, 1, \lin, 1, 0],

    \numWsLo, [1, 100, \lin, 1, 5],
    \numWsHi, [1, 100, \lin, 1, 23],
    \numWsD, [0, 7, \lin, 1, 6],

    \repeatsLo, [1, 4, \lin, 1, 1],
    \repeatsHi, [1, 4, \lin, 1, 2],
    \repeatsD, [0, 7, \lin, 1, 6],

    \maxTrigRate, [1, 250, \lin, 1, 200],

    \bpFreqLo, [50, 10000, \exp, 0.1, 67],
    \bpFreqHi, [50, 10000, \exp, 0.1, 9600],
    \bpFreqD, [0, 7, \lin, 1, 7],
    \bpRQ, [0.01, 0.99, \lin, 0.005, 0.22],
    \bpWet, [0.0, 1, \lin, 0.005, 0.0],

    \rateLo, [0.05, 2, \lin, 0.0, 0.32],
    \rateHi, [0.05, 2, \lin, 0.0, 1.3],
    \rateD, [0, 7, \lin, 1, 7],

    \att, [0.0, 0.05, \lin, 0.001, 0.001],
    \rel, [0.0, 0.05, \lin, 0.001, 0.001],

    \panMax, [0.0, 1, \lin, 0, 0.8],
```

```
    \legatoLo, [0.3, 25, \exp, 0, 0.4],
    \legatoHi, [0.3, 25, \exp, 0, 5.5],
    \legatoD, [0, 7, \lin, 1, 7],
    \amp, [0.0, 2.0, \lin, 0.0, 0.7]
    ],
    stream: p
).gui(varColorGroups: (0..25).clumps([4,3,3,1,5,3,2,1,3,1]))
)
```

3. Hybrid Implementations

Ex.3a: Granulation with ugen plus step sequencing

```
// Here the trigger for the TGrains ugen comes from a Pbind
// which also generates rates like a step sequencer


(
SynthDef(\gran_3a, { arg out = 0, posLo = 0.0, posHi = 1.0,
    posRate = 1, posDev = 0.01, bufNum = 0, t_trig = 0,
    granDur = 0.1, t_rate = 1.0, rateDev = 0,
    panMax = 1, amp = 0.1, interp = 4;

    var pan, pos, bufDur, bufDurSection, posDif;

    posDif = posHi - posLo;
    bufDur = BufDur.kr(bufNum);
    bufDurSection = bufDur * posDif;
    pos = posLo * bufDur +
        (Phasor.ar(0, BufRateScale.kr(bufNum) * posRate / SampleRate.ir, posLo * bufDur,
posHi * bufDur) +
        (TRand.kr(-0.5, 0.5, t_trig) * posDev * bufDur)).mod(bufDurSection);
    pan = Demand.kr(t_trig, 0, Dseq([panMax, panMax.neg], inf) * 0.999);
    Out.ar(out, TGrains.ar(2, t_trig, bufNum, t_rate, pos, granDur, pan, 1, interp) *
amp);
    }, metadata: (
        specs: (
            posLo: [0.01, 0.99, \lin, 0.01, 0],
            posHi: [0.01, 0.99, \lin, 0.01, 1],
            posRate: [0.1, 2, \lin, 0.01, 1],
            posDev: [0, 0.2, 5, 0, 0.01],
            panMax: [0.0, 1, \lin, 0.005, 0.8],
            amp: [0.0, 1, \lin, 0.005, 0.5]
        )
    )
).add;

b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.
)


// As the setting Pbind needs to know the Synth's nodeID
// the Synth has to be started explicitely and passed to the VarGui later on
```

```
// (VarGui takes Synths as well as SynthDefs, passing a SynthDef is recommended in
general).
// The Synth starts silently as t_trig defaults to 0.


(
x = Synth(\gran_3a, [\bufNum, b]).register;

p = Pbind(
    \type, \set,
    \id, x,
    \args, [\t_trig, \t_rate, \granDur],

    \dur, PL(\dur),
    \granDur, Pkey(\dur) * PL(\legato),
    \t_trig, 1,
    \t_rate, PLseq(\midi).midiratio
);
)

// Do start and pause with the Pbind (EventStreamPlayer) player.
// If you stop the Synth you cannot resume audio with a new Synth
// as the EventStreamPlayer has lost the correct nodeID
// (however the Synth can be paused and resumed).

(
VarGui(varCtr: [
    \dur, [0.01, 0.1, \lin, 0, 0.05],
    \legato, [0.3, 3, \lin, 0, 1],
    \midi, [-12, 12, \lin, 1, 1] ! 8
    ], synth: x, stream: p
).gui;
)
```

Ex.3b: Using external control synths

```
// Example needs SynthDef from Ex.2a

// Also with language-driven granulation
// controls can be delegated to separate Synths,
// which output to control buses.
// Control inputs of single synths can read from
// these buses (comfortably use aBus.asMap in the Pbind)
// or synths can read from buses with In.kr (needs extra definition).

// A nearby parameter to determine with a separate synth is grain position

(
SynthDef(\bufPhasor, { |out = 0, sndBuf = 0, posRate = 1, posLo = 0, posHi = 1, posDev =
0.01|
    var pos, posDif;
    posDif = posHi - posLo;
    pos = Phasor.ar(1, posRate * BufRateScale.kr(sndBuf) / BufFrames.kr(sndBuf), 0,
posDif)
        + WhiteNoise.kr(posDev / 2) % posDif + posLo;
    Out.kr(out, A2K.kr(pos));
    }
).add;

b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
```

```
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.

w = Buffer.sendCollection(s, Signal.hanningWindow(1024));
c = Bus.control(s,1);
)

// in GUI start EventStreamPlayer and bufPhasor Synth

(
p = Pbind(
    \instrument, \gran_2a,
    \sndBuf, b,
    \windowBuf, w,

    \dur, 1 / PL(\trigRate),
    \granDur, PL(\granDur),

    \pos, c.asMap,
    \rate, PL(\rate),
    \amp, PL(\amp),
    \panMax, PLseq([-1,1]) * PL(\panMax),
    \out, 0
);

VarGui([
        \trigRate, [1, 200, \lin, 0.01, 50],
        \granDur, [0.01, 0.3, \lin, 0.005, 0.12],
        \rate, [0.1, 3, \lin, 0.01, 1],
        \panMax, [0.0, 1, \lin, 0.0, 0.8],
        \amp, [0.0, 1, \lin, 0.01, 0.3]
    ],[
        \out, c.index,
        \sndBuf, b.bufnum,
        \posLo, [0, 1, \linear, 0.005, 0],
        \posHi, [0, 1, \linear, 0.005, 1],
        \posRate, [0.1, 2, \linear, 0.01, 1],
        \posDev, [0, 0.2, 5, 0, 0.01]
    ],
    p, \bufPhasor
).gui(sliderPriority: \synth, playerPriority: \synth);
)
```

Ex.3c: Switching between ugens of external control synths

```
// Example needs SynthDef from Ex.2a

// external control synth for position as in Ex.3a, but with
// different types of movement to select

(
SynthDef(\bufPosLFO, { |out = 0, lfoType = 0, freq = 1, posLo = 0, posHi = 1, posDev =
0.01|
    var pos, posDif;
    posDif = posHi - posLo;
    pos = WhiteNoise.kr(posDev / 2) + Select.kr(lfoType,
        [LFDNoise0, LFDNoise1, LFDNoise3].collect(_.kr(freq, posDif))) % posDif + posLo;
    Out.kr(out, pos);
}).add;

b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
```

```
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.

w = Buffer.sendCollection(s, Signal.hanningWindow(1024));
c = Bus.control(s,1);
)


// added parallel grains
// in GUI start bufPhasor Synth before or together with EventStreamPlayer

// buffer position movement can be forward and backward
// lfoType 0: LFDNoise0 (jumps)
// lfoType 1: LFDNoise1 (linear interpolation)
// lfoType 2: LFDNoise3 (cubic interpolation, smooth, useful in many control contexts)

(
p = Pbind(
    \instrument, \gran_2a,
    \sndBuf, b,
    \windowBuf, w,

    \dur, 1 / PL(\trigRate),
    \granDur, PL(\granDur),

    \pos, c.asMap,
    \rate, PL(\rate) * PL(\midiAdd).midiratio,
    \amp, PL(\amp),
    \panMax, PLseq([-1,1]) * PL(\panMax),
    \out, 0
);

VarGui([
        \trigRate, [1, 200, \lin, 0.01, 80],
        \granDur, [0.01, 0.3, \lin, 0.005, 0.195],
        \rate, [0.1, 1.5, \lin, 0.01, 0.6],
        \midiAdd, [-5, 0].collect([-12, 12, \lin, 1, _]),
        \panMax, [0.0, 1, \lin, 0.0, 0.95],
        \amp, [0.0, 1, \lin, 0.01, 0.15]
    ],[
        \out, c.index,
        \posLo, [0, 1, \linear, 0.005, 0.15],
        \posHi, [0, 1, \linear, 0.005, 0.43],
        \posDev, [0, 0.2, 5, 0, 0.0],
        \freq, [0.01, 2, \lin, 0, 0.96],
        \lfoType, [0, 2, \lin, 1, 2]
    ],
    p, \bufPosLFO
).gui(sliderPriority: \synth, playerPriority: \synth);
)
```

Ex.3d: Pattern-driven sequencing of granular events using demand rate ugens

```
// This is basically the same as SynthDef \gran_1d with an additional envelope,
// grain position phasor is left out, position is determined by a phasor synth via bus
and mapping.

(
// length of demand rate sequence, you might want to check larger sizes
// in connection with gui arg tryColumnNum > 1
~n = 5;
```

```
SynthDef(\gran_3d, { |out = 0, soundBuf, envBuf, att = 0.1, sus = 1, rel = 1, pos = 0.5,
    granDurMul = 1, rateMul = 1, overlapMul = 1, panMul = 1, amp = 0.5, interp = 2|

    var signal, bufDur, granDur, granGate, relGranDurs, overlap, relOverlaps, overlapSeq,
        pan, rate, relRates, rateSeq, granDurSeq;

    // array args for demand rate sequencing, short form of NamedControl
    relGranDurs = \relGranDurs.kr(0.1!~n);
    relRates = \relRates.kr(1!~n);
    relOverlaps = \relOverlaps.kr(1!~n);

    // Dstutter (or Dunique) necessary as granDurSeq is polled twice: granGate and
granDur
    granDurSeq = Dstutter(2, Dseq(relGranDurs, inf));
    rateSeq = Dseq(relRates, inf);
    overlapSeq = Dseq(relOverlaps, inf);

    granGate = TDuty.ar(granDurSeq * granDurMul);
    granDur = Demand.ar(granGate, 0, granDurSeq * granDurMul);
    rate = Demand.ar(granGate, 0, rateSeq) * rateMul;
    pan = Demand.ar(granGate, 0, Dseq([1, -1], inf)) * 0.999 * panMul;
    overlap = Demand.ar(granGate, 0, overlapSeq) * overlapMul;

    bufDur = BufDur.kr(soundBuf);
    signal = TGrains.ar(2, granGate, soundBuf, rate, pos * bufDur, granDur * overlap,
pan, 1, interp) *
        EnvGen.kr(Env([0, 1, 1, 0], [att, sus, rel]), doneAction: 2);

    Out.ar(out, signal * amp);
}
).add;

SynthDef(\bufPhasor_2, { |out = 0, sndBuf = 0, posRateE = 0, posRateM = 1,
    posLo = 0, posHi = 1, posDev = 0.01|

    var pos, posDif, posRate;
    posDif = posHi - posLo;
    posRate = 10 ** posRateE * posRateM;
    pos = Phasor.ar(1, posRate * BufRateScale.kr(sndBuf) / BufFrames.kr(sndBuf), 0,
posDif)
        + WhiteNoise.kr(posDev / 2) % posDif + posLo;
    Out.kr(out, A2K.kr(pos));
}
).add;

b = Buffer.read(s, Platform.miSCellaneousDirs[0] +/+ "Sounds" +/+
"kitchen_sounds_1.wav");
// This searches the most likely extension places for the miSCellaneous folder.
// In case of an extraordinary install situation or a removed sound file, pass the
concerned path.

c = Bus.control(s,1);
)


// Pattern control of granular events

// Every granular event gets a duration between durLo and durHi (exp distribution)
// and an envelope according to att, sus and rel.
// One of four events is randomly defined as rest.

// arg arrays relGranDurs, relRates and relOverlaps determine
// demand rate sequencing for granulation as in Ex. 1d.
// Per event they are multiplied with corresponding factors
```

```
// limited by granDurMulLo/Hi, rateMulLo/Hi and overlapMulLo/Hi

// start phasor synth (first row in player console) before event stream player
// event stream player might start with rest


(
// Passing arrayed args with an event pattern requires
// wrapping them into an array or Ref object.
// This is necessary to distinguish from triggering
// several synths per event.
// .collect(`_) is short for .collect { |x| Ref(x) }
// .collect([_]) or .collect { |x| [x] } would also be possible

p = Pbind(
    \instrument, \gran_3d,
    \soundBuf, b,
    \pos, c.asMap,

    \dur, PLexprand(\durLo, \durHi),
    \type, PLshufn(\note!3 ++ \rest),
    \att, PL(\att),
    \sus, PL(\sus),
    \rel, PL(\rel),


    \relGranDurs, PL(\relGranDurs).collect(`_),
    \granDurMul, PLwhite(\granDurMulLo, \granDurMulHi),

    \relOverlaps, PL(\relOverlaps).collect(`_),
    \overlapMul, PLwhite(\overlapMulLo, \overlapMulHi),

    \relRates, PL(\relRates).collect(`_),
    \rateMul, PLwhite(\rateMulLo, \rateMulHi),

    \panMul, PL(\panMul),
    \amp, PL(\amp)
);

VarGui([
        \durLo, [0.05, 1.5, \lin, 0, 0.16],
        \durHi, [0.05, 1.5, \lin, 0, 1.2],
        \att, [0.01, 0.6, \lin, 0, 0.4],
        \sus, [0.01, 0.6, \lin, 0, 0.05],
        \rel, [0.01, 0.6, \lin, 0, 0.5],
        \relGranDurs, { |i| [0.01, 0.1, \lin, 0, i * 0.005 + 0.02] } ! ~n,
        \granDurMulLo, [0.03, 2, \lin, 0, 0.05],
        \granDurMulHi, [0.03, 2, \lin, 0, 1.8],

        \relRates, { |i| [0.1, 1.5, \lin, 0, (5-i) * 0.1 + 0.5] } ! ~n,
        \rateMulLo, [0.1, 2, \lin, 0, 0.5],
        \rateMulHi, [0.1, 2, \lin, 0, 1.5],

        \relOverlaps, [0.5, 3, \lin, 0, 2] ! ~n,
        \overlapMulLo, [0.1, 2, \lin, 0, 0.25],
        \overlapMulHi, [0.1, 2, \lin, 0, 1.8],

        \panMul, [0.0, 1, \lin, 0.0, 0.8],
        \amp, [0.0, 3, \lin, 0.01, 1.8]
    ],[
        \out, c.index,
        \sndBuf, b.bufnum,
        \posLo, [0, 1, \lin, 0.005, 0.1],
        \posHi, [0, 1, \lin, 0.005, 0.9],
```

```
        \posRateE, [-3, 4, \lin, -1, 0],
        \posRateM, [0.1, 10, \exp, 0.01, 1],
        \posDev, [0, 0.2, 5, 0, 0.01]
    ],
    p, \bufPhasor_2
).gui(
    tryColumnNum: 2,
    sliderPriority: \synth,
    playerPriority: \synth,
    varColorGroups: (0..(~n*3+12)).clumps([2,3,~n+2,~n+2,~n+2,1,1]),
    synthColorGroups: (0..6).clumps([1,1,2,2,1]),
    labelWidth: 90,
    sliderWidth: 300
);
)
```

4. Extensions of Setups

- **Changing ranges and scaling**

  This concerns all control parameters in question. E.g. the layering of long grains (> 200 ms) in connection with small rates of position changes (posRate) often has interesting effects. One may want to drop the term granulation in that case, though it's the same structure of synthesis. For such parameters with a very large coefficient boundHi / boundLo one may take exponential scaling, and if this is not fine enough you can invent a control pair of mantissa and exponent (as for posRate in Ex.1b).

- **Parameter linkage**

  On the one hand a logical restriction, on the other hand it can make sense from a musical / perceptional point of view. E.g. shortening of grains could be linked with a raise of rate (as low frequencies might fail to unfold in short grains). Anyway parameter linkage is reducing complexity - it's a trade-off between simplicity of the interface and exclusion of certain constellations which should be considered from case to case.

- **Inventing and extending controls and LFO changes dependant on specific buffers and parameter sets**

  Say one has started playing around with a certain buffer and a general granulation patch. A parameter set that gives an interesting sound may react in a very interesting way on a change of a single parameter e.g., trigger rate. Then it may be an option to build in a control or a LFO specifically designed for that parameter - LFO is meant here in a general sense, it could be a LFO in a Synth, a dedicated LFO synth or defined by a rapidly sequencing Pattern.

- **Iterated granulation**

  Granulation of buffers themselves resulting from buffer granulation can give interesting effects.

- **Spatialization**

For the sake of ease and comparison a L/R-switch per grain with one panning parameter was used in the examples above. Needless to say that spatial scattering of grains remains a large field of experimentation. Generally spoken spatialization can be part of the synthesis process or carried through independently afterwards, but also a combination of both approaches is feasible.

- **Effects**

  Effect processing can be applied to all or single grains in a language-driven granulation setup. There can be one or more effects, serial or in parallel, defined outside or inside the Synth playing the buffer, as with the BPF in Ex.2b, Ex.2c, Ex.2d. In these examples the bandpass is applied in general, but also a sequencing of effects can be done. And even in the case of just one effect (e.g. reverb) a decent sequencing of Fx- / noFx-events can sound very interesting. This can be done with PbindFx (as in the project kitchen studies), sequencing not more than one effect per grain can be done straightly: continously running effect synths read from different buses and events with different out values cause the player synths to output to these buses.

- **Micro rhythm**

  See Xenakis' suggestion of Sieves with rhythm generation as a special application, a granulation example is contained in the example section of: Sieves and Psieve patterns. PSPdiv, a dynamic multi-layer pulse divider, which is based on Pspawner, can also be used in this context, see the last example of PSPdiv.