

The Ruben-OM patch library

<http://www.bek.no/~ruben/Research/Downloads/software.html>

Ruben Sverre Gjertsen 2013

A patch library for Open Music

The Ruben-OM user library is a collection of processes transforming musical materials in various ways, including a new rhythm quantification method. This patch library is developed as patches in Ircams Open Music and saved as generic functions. It will currently require version 6.7, beta 10, or later.¹

Included in the package

The folder "Ruben-OM-Workspace-2013" can be opened as an OM workspace. Alternatively, contents of the 'user' folder can be copied to the 'user' folder of the workspace you are using. Demo patches found in 'elements' can be copied to your 'elements' folder. The user functions will be loaded, and the folder "ruben-demo-patches" contains demonstrations of various functions. The patches can be used together with this reference text.

Score processing

Many of these user functions work on the multi-seq format (second output is list of chord-seqs). Maintaining the same input and output format through most of the functions makes it easy to place the transformations in any order, and keep other parameters in sync when manipulating only a particular parameter in each function. There could still be some problems processing empty voices, or voices with only one note. This is a challenge for the future. When preparing this shared library, I have tried to include only the most general tools, without too many personal preferences of material and intervals. Some older patches are kept for compatibility or used within the main functions. I will not go through all these low level functions.

Documentation

This function reference describes inputs and outputs of each function. Similar information can be found from the patches by selecting a function and typing 'd'. The demo patches show examples of use, which can be just as informative.

¹ Free download: <http://repmus.ircam.fr/openmusic/download>

FUNCTION REFERENCE

add-chords-multiseq

INPUTS

1. List of chord-seqs.
2. Textfile self: Containing intervals (2-note chords) and lists of intervals (chords with 3 or more notes) to be added as chords.
3. Number: Percent probability to have an added chord. The chords are selected randomly from the textfile.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Random addition of chords to an input material.

add-rests-multiseq

INPUTS

1. List of chord-seqs.
2. List of bpf's(1000*1000): The bpf's show trajectories for percentage of rests through the input music. If there are too few bpf's for the number of chord-seqs, last will be repeated. If there are too many, the list will be cropped.
3. List of 2 numbers: Maximum and minimum percentage of rests.
4. List of bpf's(1000*1000): The bpf's show trajectories for length of rests through the input music. If there are too few bpf's for the number of chord-seqs, last will be repeated. If there are too many, the list will be cropped.
5. List of 2 numbers: Maximum and minimum milliseconds length of rests.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Pierre Boulez has called inserted rests "pockets of silence"². *add-rests-multiseq* inserts random silences based of probability curves for how often rests are inserted, and how long they are. As the function has random operations, the result will be different at each evaluation.

DEMO PATCH

add-rests-multiseq

adjust-onset-dur

INPUTS

1. List of lists: Onsets directly from *multiseq-extract-param*.
2. List of lists: Durations directly from *multiseq-extract-param*.
3. Number: Minimum milliseconds duration between notes.

OUTPUTS

1. List of lists: New onsets for *rebuild-multiseq*.
2. List of lists: New durations for *rebuild-multiseq*.

DESCRIPTION

This function is used between *multiseq-extract-param* and *rebuild-multiseq*. Onsets and durations are adjusted after the minimum value, to make sure there are no zero duration notes, and no overlapping notes. This makes the information useful for quantification into a single voice.

² Pierre Boulez, 1971, Boulez on music today, p. 57.

Spectral analysis chord-seqs (obtained through the pm2 library³) will pose problems for direct quantification:

- Notes shorter than 1 millisecond could be rounded down to 0. The reason is that onsets and durations of a chord-seq are ints, not floats. This causes errors.
- There are frequent superpositions of different partials. If these could be spread to multiple voices, this would not be a problem. For a monophonic voice, overlaps need to be cut.

DEMO PATCH

quantify-hockets-multiple-demo

amplify-multiseq

INPUTS

1. List of chord-seqs.
2. Number added to velocity (positive or negative). The MIDI velocity range is 1-127. If the result exceeds 127, velocity is set to 127.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Adds to all dynamics (velocity) of the input music.

DEMO PATCH

dynamics-filter-demo

apply-channel-chordseq

INPUTS

1. Chord-seq self.
2. Midi channel number.

OUTPUTS

1. Chord-seq self.

DESCRIPTION

The same input channel is applied to every note and chord in the material. This is useful for playback, or for identifying different materials further down a chain of processes.

apply-channels-multiseq

INPUTS

1. List of chord-seqs.
2. List of midi channel numbers.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Each chord-seq is matched with a channel number, which is applied to every note and chord in that chord-seq. This is useful for playback, or for identifying different materials further down a chain of processes.

applychannel2voice

INPUTS

1. Voice self.
2. Midi channel number.

OUTPUTS

1. Voice self.

DESCRIPTION

The same input channel is applied to every note and chord in the material. This is useful for playback, or for identifying different materials further down a chain of processes.

³ <http://support.ircam.fr/docs/om-libraries/main/co/OM-pm2.html>

approx2mode-multiseq

INPUTS

1. List of chord-seqs.
2. List of numbers: Midicents in mode.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

The input notes will be approximated to nearest note of the input mode.

ascending-maj-seconds

INPUTS

1. List: Midicent.
2. Number: Milliseconds duration.
3. Number: Speed ratio. 1 is normal, .5 is half speed, 2 is double speed.

OUTPUTS

1. Chordseq-datalist. Connecting this to *rebuild-chordseq* will create a chord-seq from these lists.

DESCRIPTION

This is an example of the gesture patches. Use this directly or as an input abstraction in lambda mode (select function, type 'l'). A list of gesture-patches in lambda mode can serve as abstractions for the *gestures-chordseq* function. This will place the gestures into a context of harmony and timing.

We will create all the gesture patches with the same 3 inputs for compatibility, even though parameters may be ignored by the actual function.

DEMO PATCH

gestures-sciarrino-generator

ascending-min-seconds

INPUTS

1. List: Midicent.
2. Number: Milliseconds duration.
3. Number: Speed ratio. 1 is normal, .5 is half speed, 2 is double speed.

OUTPUTS

1. Chordseq-datalist. Connecting this to *rebuild-chordseq* will create a chord-seq from these lists.

DESCRIPTION

This is an example of the gesture patches. Use this directly or as an input abstraction in lambda mode (select function, type 'l'). A list of gesture-patches in lambda mode can serve as abstractions for the *gestures-chordseq* function. This will place the gestures into a context of harmony and timing.

We will create all the gesture patches with the same 3 inputs for compatibility, even though parameters may be ignored by the actual function.

DEMO PATCH

gestures-sciarrino-generator

ascending-thirds

INPUTS

1. List: Midicent.
2. Number: Milliseconds duration.
3. Number: Speed ratio. 1 is normal, .5 is half speed, 2 is double speed.

OUTPUTS

1. Chordseq-datalist. Connecting this to *rebuild-chordseq* will create a chord-seq from these lists.

DESCRIPTION

This is an example of the gesture patches. Use this directly or as an input abstraction in lambda mode (select function, type 'l'). A list of gesture-patches in lambda mode can serve as abstractions for the *gestures-chordseq* function. This will place the gestures into a context of harmony and timing.

We will create all the gesture patches with the same 3 inputs for compatibility, even though parameters may be ignored by the actual function.

DEMO PATCH

gestures-sciarrino-generator

augmentator

INPUTS

1. Number: 4ths length.
2. Fraction: Ratio before (ex. 1/3). 0 is also possible.
3. Fraction: Ratio after (ex. 1/3). 0 is also possible.
4. List of numbers: Rhythmic proportions.

OUTPUTS

1. List of numbers.

DESCRIPTION

augmentator can be used for calculation of rhythmic proportions or macro rhythms, with optional silences before and after. Negative numbers are rests. Together with *point2pointcalculator*, I have used this to create a network of augmented and diminished macro rhythms from a few series of proportions. This has formed a timing basis for many of my pieces.

DEMO PATCH

macro-rhythm-demo

bandpass-bpfs-multiseq

INPUTS

1. List of chord-seqs.
2. List of 2 sublists of 2 numbers: ((LopassMinimumMidicent LopassMaximum)(HipassMinimum HipassMaximum))
3. List of 2 bpfs (1000*1000): The first bpf is highpass curve within midicent range of input 2, the second bpf is a lopass curve.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is a time-variable bandpass filter, applied to every chord-seq of the input.

DEMO PATCH

pitch-bpf-filter-demo

bandpass-multiseq

INPUTS

1. List of chord-seqs.
2. List of 2 numbers: (LopassMidicent HighpassMidicent).

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is static bandpass filter, applied to every chord-seq of the input.

DEMO PATCH

pitch-filter-demo

bandpass-percent-multiseq

INPUTS

1. List of chord-seqs.
2. List of 2 numbers: (LowLimitMidicent HighLimitMidicent).
3. Number: Percent notes passing the filter.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is a "soft" bandpass filter, which doesn't remove everything; it will thin out the filtered region by passing only the indicated percent of notes by random selection. As this function includes random selection, the results will vary with every evaluation.

DEMO PATCH

pitch-softfilter-demo

bandreject-bpfs-multiseq

INPUTS

1. List of chord-seqs.
2. List of 2 sublists of 2 numbers: ((LowLimitMidicentMinimum LowLimitMaximum)(HighLimitMinimum HighLimitMaximum))
3. List of 2 bpfs (1000*1000): The first bpf is a low limit curve within midicent range of input 2, the second bpf is a high limit curve.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is a time-variable bandreject filter, applied to every chord-seq of the input.

DEMO PATCH

pitch-bpf-filter-demo

bandreject-multiseq

INPUTS

1. List of chord-seqs.
2. List of 2 numbers: (LowLimitMidicent HighLimitMidicent).

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is static bandreject filter, applied to every chord-seq of the input.

DEMO PATCH

pitch-filter-demo

bandreject-percent-multiseq

INPUTS

1. List of chord-seqs.
2. List of 2 numbers: (LowLimitMidicent HighLimitMidicent).
3. Number: Percent notes passing the filter.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is a "soft" bandreject filter, which doesn't remove everything; it will thin out the filtered region by passing only the indicated percent of notes by random selection. As this function includes random selection, the results will vary with every evaluation.

DEMO PATCH

pitch-sofftfilter-demo

bpfs-dyn-multiseq

INPUTS

1. List of chord-seqs.
2. List of bpf's (1000*1000).

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

The bpf's will apply a dynamics (velocity) curve to each voice in the input music. If there are fewer bpf's than chord-seqs, the last one will be repeated as necessary. If there are too many, the list will be cropped.

DEMO PATCH

dynamics-multiseq

cent-approx-multiseq

INPUTS

1. List of chord-seqs.
2. Cents interval size. This can be any number. 100 is a halftone, 50 is a quarter-tone, 25 is an eight-tone, 200 is a wholetone.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Approximates notes and chords to interval sizes within an equal tempered system.

DEMO PATCH

approximation-demo

celestarange

INPUTS

1. Dummy, nothing needs to be connected.

OUTPUTS

1. List of possible midicents for celesta.

DESCRIPTION

This is a simple list of notes for a particular instrument, useful together with *my-instrument-multiseq*.

DEMO PATCH

quantify-instrumentalpart-demo

chain-chordseqlists-rests

INPUTS

1. Lists of chord-seq lists. A 'chord-seq list' is the output of the function *instead-of-chordseq*.
2. Matching list of milliseconds between each chord-seq.

OUTPUTS

1. Chord-seq datalist (*instead-of chordseq* list).

DESCRIPTION

This is a low-level function used inside *gestures-chordseq*. A bug in Om 6.5.1 would create a wrong last onset in the multi-seq object. This would create unintentional long notes within the material. Working with list of lists of lists instead of chord-seqs is a workaround for this problem. *instead-of-chordseq* creates a list which can be joined with other lists. The "last

onset" problem was fixed with Om 6.6.

DEMO PATCH

gestures-sciarrino-generator

channel-bpf-multiseq

INPUTS

1. List of chord-seqs.
2. Midi channels range (min. max.).
3. List of bpf (1000*1000). This should match the number of chord-seqs. If too few, last bpf will be repeated. If too many, the list will be cropped.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

A sequence of MIDI channels, within the indicated range, is created from the bpf curves. This is applied to the chord-seqs in the input material. This can be used to create a timbre melody, or with channel-hocket-multiseq to split a chord-seq into multiple chord-seq, sorted again by channel number.

DEMO PATCH

channel-bpf-env-demo

channel-hocketing-demo

channel-hocket-multiseq

INPUTS

1. List of chord-seqs.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Notes and chord are sorted into separate chord-seqs by MIDI channel number. There will be one chord-seq for each channel number in ascending order.

DEMO PATCH

channel-bpf-env-demo

channel-hocketing-demo

chordseq-chain-rests

INPUTS

1. List of chord-seqs.
2. List of milliseconds before each chord-seq.

OUTPUTS

1. Chord-seq.

DESCRIPTION

Multiple chord-seqs will be added in a sequence with possible silences before each.

chordseq-chordseqlists-rests

INPUTS

1. Lists of chord-seq lists. A 'chord-seq list' is the output of the function *instead-of-chordseq*.
2. Matching list of milliseconds between each chord-seq.

OUTPUTS

1. Chord-seq self.

DESCRIPTION

This function is almost identical to *chain-chordseqlists-rests*, except the output is a chord-seq and not the alternative *instead-of-chordseq* list format.

DEMO PATCH

gestures-sciarrino-generator

delay-ms-multiseq

INPUTS

1. List of chord-seqs.
2. Number: Milliseconds delay.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

The input music is delayed by n milliseconds.

ensemble-circles

INPUTS

1. Bpf (1000*1000): Trigger curve for arpeggio between multiple parts. The curve is divided into number of parts on the y-axis. A new note or event will start each time the curve crosses the border between parts.
2. Number: Overall duration seconds.
3. List of 2 numbers: Attack blur seconds minimum/maximum. (0 0) means no change. Numbers will be kept positive, and attacks will not extend past next attack.
4. List of 2 numbers: Duration blur seconds minimum/maximum. (0 0) means no change. Negative numbers will create staccato, positive numbers will create overlaps. However, the note will not extend past the next attack of a part, in order not to obstruct the curve shapes.
5. Number: Number of curve samples. 1000 could be a good compromise. With small numbers, the function will not detect all movements of the curve.
6. List of patches in lambda mode. These must have the following standard structure:
INPUTS: 1. List of lists of midicents. 2. Number: Milliseconds duration. 3. Number: Speed ratio (1 is default speed, .5 is half speed). OUTPUT Chord-seq datalist.
Check the prototype patches *gen-simple-note*, *gen-simple-chord*, *sciarrino-passage*, *ascending-thirds* for clarification. As long as they have similar inputs and outputs, new patches can expand a repertoire of gestures. Patches making no, or just approximate, use of the duration input, will bring parts out of sync with the overall ensemble-circles shapes. In this version, each gesture patch will correspond to one chord-seq of the output.
7. Chord-seq self: Each note or chord will be the base of the input gestures. Number of notes or chords is the same as the number of parts in the ensemble-circles texture.
8. List of numbers: Speed factors as inputs for the gesture patches. Lists of input 6, 7 and 8 should have the same length.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

ensemble-circles will create orchestral arpeggio shapes with input gestures and a fixed chord base. If pitches are selected in order by register, it will be a conventional register arpeggio. If instrumental pitches are ordered by spatial position, the curve shape could create spatial movements. *random-sawtooth-bpfs* can generate shapes with multiple rotations.

Iannis Xenakis' sketches for Terretektorh include shapes for instrumental spatial movements through the hall⁴, and they have been an inspiration for the work on *ensemble-circles*.

ensemble-circles uses one gesture patch for each part, and a fixed chord base.

ensemble-circles-lists allows a flexible gesture script for each part, and a fixed chord base.

ensemble-circles-lists-gliss allows a flexible gesture script for each part, and an interpolated chord base.

DEMO PATCH

ensemble-circles

⁴ Maria Anna Harley, 1994, Space and Spatialization in Contemporary Music: History and Analysis, Ideas..., p. 289.

ensemble-circles-lists

INPUTS

1. Bpf (1000*1000): Trigger curve for arpeggio between multiple parts. The curve is divided into number of parts on the y-axis. A new note or event will start each time the curve crosses the border between parts.
2. Number: Overall duration seconds.
3. List of 2 numbers: Attack blur seconds minimum/maximum. (0 0) means no change. Numbers will be kept positive, and attacks will not extend past next attack.
4. List of 2 numbers: Duration blur seconds minimum/maximum. (0 0) means no change. Negative numbers will create staccato, positive numbers will create overlaps. However, the note will not extend past the next attack of a part, in order not to obstruct the curve shapes.
5. Number: Number of curve samples. 1000 could be a good compromise. With small numbers, the function will not detect all movements of the curve.
6. List of patches in lambda mode. These must have the following standard structure:
INPUTS: 1. List of lists of midicents. 2. Number: Milliseconds duration. 3. Number: Speed ratio (1 is default speed, .5 is half speed). OUTPUT Chord-seq datalist.
Check the prototype patches *gen-simple-note*, *gen-simple-chord*, *sciarrino-passage*, *ascending-thirds* for clarification. As long as they have similar inputs and outputs, new patches can expand a repertoire of gestures. Patches making no, or just approximate, use of the duration input, will bring parts out of sync with the overall ensemble-circles shapes. In this version, the list of patches can be of any length. The script at input 7 will create gesture indexes for each part.
7. Textfile self: The file will contain one list of index numbers for each part (0-x). 0 is the first gesture of input 6. The lists can be of any length. If there are too few gesture indexes, the rest will be randomly chosen from the list. If there are too many, the list will be cropped. If a part is missing a gesture list, the default will be (0).
8. Chord-seq self: Each note or chord will be the base of the input gestures. Number of notes or chords is the same as the number of parts in the ensemble-circles texture.
9. List of numbers: Speed factors as inputs for the gesture patches.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

ensemble-circles-lists will create orchestral arpeggio shapes with a gesture script and a fixed chord base. If pitches are selected in order by register, it will be a conventional register arpeggio. If instrumental pitches are ordered by spatial position, the curve shape could create spatial movements. *random-sawtooth-bpfs* can generate shapes with multiple rotations.

ensemble-circles uses one gesture patch for each part, and a fixed chord base.

ensemble-circles-lists allows a flexible gesture script for each part, and a fixed chord base.

ensemble-circles-lists-gliss allows a flexible gesture script for each part, and an interpolated chord base.

DEMO PATCH

ensemble-circles-lists

ensemble-circles-lists-gliss

INPUTS

1. Bpf (1000*1000): Trigger curve for arpeggio between multiple parts. The curve is divided into number of parts on the y-axis. A new note or event will start each time the curve crosses the border between parts.
2. Number: Overall duration seconds.
3. List of 2 numbers: Attack blur seconds minimum/maximum. (0 0) means no change. Numbers will be kept positive, and attacks will not extend past next attack.

4. List of 2 numbers: Duration blur seconds minimum/maximum. (0 0) means no change. Negative numbers will create staccato, positive numbers will create overlaps. However, the note will not extend past the next attack of a part, in order not to obstruct the curve shapes.
5. Number: Number of curve samples. 1000 could be a good compromise. With small numbers, the function will not detect all movements of the curve.
6. List of patches in lambda mode. These must have the following standard structure:
 INPUTS: 1. List of lists of midicents. 2. Number: Milliseconds duration. 3. Number: Speed ratio (1 is default speed, .5 is half speed). OUTPUT Chord-seq datalist.
 Check the prototype patches *gen-simple-note*, *gen-simple-chord*, *sciarrino-passage*, *ascending-thirds* for clarification. As long as they have similar inputs and outputs, new patches can expand a repertoire of gestures. Patches making no, or just approximate, use of the duration input, will bring parts out of sync with the overall ensemble-circles shapes. In this version, the list of patches can be of any length. The script at input 7 will create gesture indexes for each part.
7. Textfile self: The file will contain one list of index numbers for each part (0-x). 0 is the first gesture of input 6. The lists can be of any length. If there are too few gesture indexes, the rest will be randomly chosen from the list. If there are too many, the list will be cropped. If a part is missing a gesture list, the default will be (0).
8. List of numbers: Speed factors as inputs for the gesture patches.
9. Chord-seq self: Initial pitches. There should be one note or chord for each part.
10. Chord-seq self: Final pitches. This is the destination of the chord interpolation, and there should be as many final pitches as initial pitches. Order matters for glissando directions.
11. List of bpf's (1000*1000): Chord interpolation shapes, one for each part. These shapes can help making the interpolations less linear and predictable. If there are too few bpf's, last will be repeated. If there are too many, the list will be cropped.
12. List of numbers: Cents size of the bpf modulation. This decides the effect of bpf's from input 11.
13. Number: Cents intervalsize for the base chords with interpolation. Gesture patches can still add other interval sizes to the base chords.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

ensemble-circles-lists-gliss will create orchestral arpeggio shapes with a gesture script and a interpolating chord base. If the bpf pitch modulation has a large interval size, the interpolation will be less predictable.

ensemble-circles uses one gesture patch for each part, and a fixed chord base.

ensemble-circles-lists allows a flexible gesture script for each part, and a fixed chord base.

ensemble-circles-lists-gliss allows a flexible gesture script for each part, and an interpolated chord base.

DEMO PATCH

ensemble-circles-lists-gliss

extract-modelist-multiseq

INPUTS

1. List of chord-seqs.

OUTPUTS

1. List of midicents.

DESCRIPTION

Returns notes present in the input music without repetitions. This is useful to identify tuning system and tune another fragment into the same tuning. It could also be used to extract spectral analysis from pm2⁵ and use it as intonation for musical materials.

⁵ <http://support.ircam.fr/docs/om-libraries/main/co/OM-pm2.html>

extract-onset-fragment-multiseq

INPUTS

1. List of chord-seqs.
2. List of 2 numbers: (MinimumMillisecondsOnset MaximumOnset).

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

extract-onset-fragment-multiseq extracts only notes within the onset range, with original onsets. *extract-onset-range-multiseq* is similar, except that onsets will start at 0.

DEMO PATCH

extract-onset-range-demo

extract-onset-range-multiseq

INPUTS

1. List of chord-seqs.
2. List of 2 numbers: (MinimumMillisecondsOnset MaximumOnset).

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

extract-onset-range-multiseq extracts only notes within the onset range, with onsets starting at 0. Used in loops, this function could split musical materials into multiple tiny parts, forming a type of musical granulation. *extract-onset-fragment-multiseq* is similar, except that it keeps the original onsets.

DEMO PATCH

extract-onset-range-demo

float-random

INPUTS

1. List of 2 floats: Minimum and maximum number.

OUTPUTS

1. Floating number.

DESCRIPTION

This function returns a random floating number within a range.

gen-simple-chord

INPUTS

1. List of lists: Midicents of chord.
2. Number: Milliseconds duration.
3. Number: Speed ratio. 1 is normal, but this parameter is ignored in this particular gesture function.

OUTPUTS

1. Chordseq-datalist. Connecting this to *rebuild-chordseq* will create a chord-seq from these lists.

DESCRIPTION

This is a simple example of gesture patches. Use this directly to create only one chord, or as an input abstraction in lambda mode (select function, type 'l'). A list of gesture-patches in lambda mode can serve as abstractions for the *gestures-chordseq* function. This will place

the gestures into a context of harmony and timing.

We will create all the gesture patches with the same 3 inputs for compatibility, even though parameters may be ignored by the actual function.

gen-simple-chord creates a chord from the first input chord, with indicated direction. Speed is irrelevant in this case, as there is only one duration, and not a phrase which need indication of speed.

DEMO PATCH

gestures-sciarrino-generator (this function is not used in this patch, but it could easily be added to the list of lambda patches, and be referred to by the script)

gen-simple-note

INPUTS

1. List: Midicent (only the first value will be used).
2. Number: Milliseconds duration.
3. Number: Speed ratio. 1 is normal, but this parameter is ignored in this particular gesture function.

OUTPUTS

1. Chordseq-datalist. Connecting this to *rebuild-chordseq* will create a chord-seq from these lists.

DESCRIPTION

This is a simple example of gesture patches. Use this directly to create only one note, or as an input abstraction in lambda mode (select function, type 'l'). A list of gesture-patches in lambda mode can serve as abstractions for the *gestures-chordseq* function. This will place the gestures into a context of harmony and timing.

We will create all the gesture patches with the same 3 inputs for compatibility, even though parameters may be ignored by the actual function.

gen-simple-note creates a note with indicated direction. Speed is irrelevant in this case, as there is only one duration, and not a phrase which need indication of speed.

DEMO PATCH

gestures-sciarrino-generator (this function is not used in this patch, but it could easily be added to the list of lambda patches, and be referred to by the script)

gestures-chordseq

INPUTS

1. List of patches in lambda mode. These must have the following standard structure:
INPUTS: 1. List of lists of midicents. 2. Number: Milliseconds duration. 3. Number: Speed ratio (1 is default speed, .5 is half speed). OUTPUT Chord-seq datalist.
Check the prototype patches *gen-simple-note*, *gen-simple-chord*, *sciarrino-passage*, *ascending-thirds* for clarification. As long as they have similar inputs and outputs, new patches can expand a repertoire of gestures.
2. Textfile self: Gesture index (0 is the first lambda gesture patch, 1 is the second). This is the overall script for order of different gesture types. The following textfile inputs must all have the same number of numbers for this to work correctly.
3. Chord-seq self: A macro melody. Each note or chord will be the input for lambda patches pointed by the gesture index script. Make sure it has the same length.
4. Textfile-self: Milliseconds duration of each gesture.
5. Textfile-self: Speed ratios for each gesture.
6. Textfile-self: Milliseconds silence between each gesture. A chord-seq cannot transfer information of silence before and after, as it contains only information about *notes and chords*. For this reason a separate script of rests has been added.

OUTPUTS

1. Chord-seq self.

DESCRIPTION

gestures-chordseq creates a chordseq from a scripted sequence of gestures and a macro melody. This can use simple patches on a compositional meta level. I have chosen to

demonstrate this composition of gestures through a Sciarrino-generator. The piano part of *Un'immagine di Arpocrate* for piano and orchestra shows the use of figure types in variable durations and transpositions, among them 2 gestures, here named *sciarrino-passage* and *ascending-thirds*. These are signature gestures appearing in many of his piano works. Using only 2 different gestures is of course a simplification of his musical language. One could analyze each piece and recognize a different palette of gestures. Gesture scripts could be generated by a set of rules: Which type of gestures are possible endings, which gestures need to be followed by something else?

DEMO PATCH

gestures-sciarrino-generator

get-hocket-channel

INPUTS

1. List of chord-seqs.

OUTPUTS

1. List of channels

DESCRIPTION

get-hocket-channel returns the first MIDI channel number from each chord-seq. This is used to identify instruments by MIDI channel, and assumes there is only one MIDI channel for each chord-seq. If this isn't true, let the chord-seqs pass through *channel-hocket-multiseq* first.

get-multiseq-dur

INPUTS

1. List of chord-seqs.

OUTPUTS

1. Seconds duration.

DESCRIPTION

This function finds the total duration of the longest chord-seq. This is useful if you want to calculate a stretch factor related to original speed, instead of a fixed duration, for instance as input to *multiseq-env-poly*.

gliss-chords

INPUTS

1. Chord self: Initial pitches.
2. Chord self: Finale pitches. Order matters for glissando directions.
3. List of bps(1000*1000): Curve modulation of the pitch interpolation.
4. List of numbers: Cents range of the bpf modulation.
5. List of numbers: Number of notes in each part.
6. Number: Seconds duration.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

gliss-chords create a glissando from one chord to another. Bpf-shapes allow the transitions to happen in other shapes than straight lines. The function makes it possible to listen to harmonic situations during this transition, even though a glissando notation for a group of strings would only contain the turning points.

DEMO PATCH

gliss-chords

group-chordseqs

INPUTS

1. List of chord-seqs.
2. Number of chord-seqs in the output list.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is an editing tool reducing the number of chord-seqs. If input number 2 is 5, the maximum number of voices is 5. If there are 12 input chord-seqs, the first output chordseq will be filled with input chordseq 1, 6 and 11. This can join materials from multiple processes into a fixed number of instrumental parts.

harmonize-multiseq

INPUTS

1. List of chord-seqs.
2. List of numbers: Cent transpositions (positive or negative) from each note of the input music.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

The interval lists will be added as a chord to every note. This works like an organ mixture, or a static version of the Boulez 'chord multiplication'. If the second input was a list of lists (or another list of chords-seqs), with some sort of indexing matrix for chords to 'multiply'⁶ with each other, the patch could be expanded to work on the techniques used by Boulez in parts of *Le Marteau sans maître*. The patch inside could be expanded for this purpose.

DEMO PATCH

harmonize-multiseq-demo

hipass-bpf-multiseq

INPUTS

1. List of chord-seqs.
2. List of 2 numbers: (HipassMidicentMinimum HipassMaximum).
3. Bpf (1000*1000): Hipass curve within range of input 2.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is a time-variable hipass filter, applied to every chord-seq of the input.

DEMO PATCH

pitch-bpf-filter-demo

hipass-multiseq

INPUTS

1. List of chord-seqs.
2. Number: Highpass midicents.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is static hipass filter, applied to every chord-seq of the input.

DEMO PATCH

pitch-filter-demo

hipass-percent-multiseq

INPUTS

1. List of chord-seqs.
2. Number: Highpass note midicent.
3. Number: Percent notes passing the filter.

OUTPUTS

1. List of chord-seqs.

⁶ In a Boulezian sense.

DESCRIPTION

This is a "soft" hipass filter, which doesn't remove everything; it will thin out the filtered region by passing only the indicated percent of notes by random selection. As this function includes random selection, the results will vary with every evaluation.

DEMO PATCH

pitch-sofffilter-demo

hocket-multiseq

INPUTS

1. List of chord-seqs.
2. Number: Cents approximation (50 for 1/4-tone, 25 for 1/8-tone, 12.5 for 1/16-tone etc.).

OUTPUTS

1. List of chord-seqs without note approximation.
2. List of chord-seqs with note approximation.

DESCRIPTION

hocket-multiseq will split a microtonal material into parts where each musician could play only semitones at different concert pitches. Tristan Murail has used this approach to microtonal orchestral writing in the work *Le Partage des eaux*, and this is my implementation of this idea. The patch can be designed to create one playback score and one score for Finale export. The patch will not react to values larger than 100 cents (semitone). 50 will separate the chord-seq into 2 hockets: the first around the semitone, the second 1/4-tones above. 25 cent will give 4 hockets where each is 1/8-tone higher than the preceding. Any other semitone division is possible. If the division does not add up to 100, the last range will be increased. This means no notes will be lost.

DEMO PATCH

hocket-spectrum-demo

instead-of-chordseq

INPUTS

1. List of lists: Midicent notes and chords.
2. List: Onsets milliseconds.
3. List: Durations milliseconds.
4. List: Velocities (1-127).
5. List: Offsets.
6. List: Channels
7. Number: Legato.

OUTPUTS

1. List of lists, used instead of chord-seq.

DESCRIPTION

This function creates a list equivalent to the chord-seq inputs. OM 6.5.1 had some problems with last onsets. This format of lists is a workaround to avoid these errors. *rebuild-chordseq* is used to get this list back to the chord-seq format. The lambda gesture patches, like for instance *sciarrino-passage*, have *instead-of-chordseq* inside.

DEMO PATCH

gestures-sciarrino-generator

instrument-region

INPUTS

1. List: Possible midicents of an instrument.
2. List of 2 numbers: Minimum and maximum percent (0-100) of register.

OUTPUTS

1. List of numbers: Midicents within the indicated register.

DESCRIPTION

instrument-region is useful if you don't want to use the whole possible register of an instrument. (30 60) will return only the middle register, (0 100) will return everything.

interval-multiply-multiseq

INPUTS

1. List of chord-seqs.
2. Interval scaling factor. 1 is original pitch, 2 is double interval size, .5 is half interval size. Any floating number is possible.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

interval-multiply-multiseq literally multiplies intervals of the input music by a factor. Multiplication of intervals has been used by Tristan Murail to create inharmonic stretched spectrums.⁷ This is different from the Boulez 'chord multiplication', which implies pure transpositions of intervals. For a static version of the Boulez technique, check *harmonize-multiseq*.

DEMO PATCH

harmonize-multiseq-demo

invert-spectre-multiseq

INPUTS

1. List of chord-seqs.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This function inverts all intervals of the input music; high becomes low, and low becomes high. If the input is a simple harmonic series, the result will be a sub-harmonic spectra, as used by Grisey in the work *Modulations*.

knoppskyting

INPUTS

1. Chord-seq self: Macro melody.
2. List of 2 numbers: Number of gestures for each macro melody note, minimum and maximum.
3. List of 2 numbers: Speed ratio, minimum and maximum.
4. List of 2 durations: Millisecond duration, minimum and maximum.
5. List of 2 numbers: Cents random deviations from macro melody note, minimum and maximum.
6. Number: Cents interval size.
7. Lambda patch with 4 inputs:

INPUTS: 1. List of lists of midicents. 2. Number: Milliseconds duration. 3. Number: Speed ratio (1 is default speed, .5 is half speed). 4. Number: Cents interval size.

OUTPUT Chord-seq datalist.

This is an early version of the gesture patch idea. Having more control over interval size for all the gesture patches could be a good idea. But it would need adjustment to many patches, including *gestures-chordseq*, *ensemble-circles*, *ensemble-circles-lists*, *ensemble-circles-lists-gliss*.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

knoppskyting creates orchestral textures from a macromelody. 1 or more gestures will start on each note of the macro melody, with possible inflections around the pitch. A tree with a network of branches could serve as a metaphor. There will be only one gesture in each chord-seq, and a long macro melody would create a huge score. The *ensemble-circles* functions are actually improvements to this idea, as they offer more control over the number of parts over a

⁷ Rozalie Hirs, 2009, Contemporary compositional techniques and openmusic, p. 104.

longer period of time. *knoppskyting* could still be used for more specific and short orchestral outbursts.

DEMO PATCH

knoppskyting-demo

lists-to-bpf-lib

INPUTS

1. List of lists: Any sequence of numbers (length more than 1) can be used as control data.
2. Number: Resolution of the bpf. 1000 will be compatible with many of the other functions within this library.

OUTPUTS

1. Bpf-lib self.

DESCRIPTION

This function is used to read any kind of number sequence as control data into a bpf. In the demo patch, *multiseq-extract-param* is used to read pitch or velocities into bpf. Further possibilities are numerous, as many functions use bpf as controlling inputs.

DEMO PATCH

controldata-multiseq-demo

lop-pass-bpf-multiseq

INPUTS

1. List of chord-seqs.
2. List of 2 numbers: (LowpassMidicentMinimum LowpassMaximum).
3. Bpf (1000*1000): Lowpass curve within range of input 3.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is a time-variable lowpass filter, applied to every chord-seq of the input.

DEMO PATCH

pitch-bpf-filter-demo

lop-pass-multiseq

INPUTS

1. List of chord-seqs.
2. Number: Lowpass midicent.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is static lowpass filter, applied to every chord-seq of the input.

DEMO PATCH

pitch-filter-demo

lop-pass-percent-multiseq

INPUTS

1. List of chord-seqs.
2. Number: Lowpass note midicent.
3. Number: Percent notes passing the filter.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is a "soft" lowpass filter, which doesn't remove everything; it will thin out the filtered region by passing only the indicated percent of notes by random selection. As this function includes random selection, the results will vary with every evaluation.

DEMO PATCH

make-ratio-intervals

INPUTS

1. List of 2 numbers: Partial number minimum and maximum, ex: (1 128).
2. Number of neighbor tones to be considered, ex: (1 2 3 4 5 6 7 8 9 10 11)

OUTPUTS

1. Flat list of midicent intervals calculated from overtone ratios.

DESCRIPTION

This function generates a reservoir of possible just intonation intervals as midicents. Inputs will decide the length of this list.

multi-env-poly

INPUTS

1. List of multi-seq selfs. If number of chord-seqs within don't match, the smallest amount will be used.
2. Number: Tempo.
3. Bpf-lib (1000*1000) defining shape of textural attack and decay between each multi-seq. There must be 1 more bpf than there are multi-seqs, as the decay of last texture also needs to be defined.
4. List of numbers: Percent duration of each attack and decay (.1 is 1/10 of the overall duration).
5. List of numbers: Texture durations between attack and decay. The length of this list equals length of multiseq list. If the numbers don't add up to 1, overall duration will not be as defined at input 7.
6. List of time signatures. The quantification method within the patch is *tree-quant*.
7. Number: Seconds total duration.
8. Complexity factor (low number for simplicity, high number for precision, 10 is an average compromise).

OUTPUTS

1. List of voices.

DESCRIPTION

The input texture and transistions between textures, are rhythmically sculpted by a series of attack/decay-envelopes. This function is not limited to a single attack/decay pair, but can handle texture lists of any length.

The quantification method inside is *tree-quant*.

DEMO PATCH

env-multi-demo

multi-env-poly-legato-tie

INPUTS

1. List of multi-seq selfs. If number of chord-seqs within don't match, the smallest amount will be used.
2. Number: Tempo.
3. Bpf-lib (1000*1000) defining shape of textural attack and decay between each multi-seq. There must be 1 more bpf than there are multi-seqs, as the decay of last texture also needs to be defined.
4. List of numbers: Percent duration of each attack and decay (.1 is 1/10 of the overall duration).
5. List of numbers: Texture durations between attack and decay. The length of this list equals

length of multiseq list. If the numbers don't add up to 1, overall duration will not be as defined at input 7.

6. List of time signatures. The quantification method within the patch is tree-quant.

7. Number: Seconds total duration.

8. Complexity factor (low number for simplicity, high number for precision, 10 is an average compromise).

9. Milliseconds rest threshold.

10. Milliseconds tie threshold.

OUTPUTS

1. List of voices.

DESCRIPTION

The input texture and transitions between textures, are rhythmically sculpted by a series of attack/decay-envelopes. This function is not limited to a single attack/decay pair, but can handle texture lists of any length. This is the most recent version, with the best control of each texture duration.

The quantification method inside is *tree-quant-legato-tie*.

DEMO PATCH

env-multi-demo

multiseq-env-poly

INPUTS

1. List of chord-seq's.

2. Number: Tempo.

3. bpf-lib self with 2 bpf's: attack shape and decay shape (1000*1000).

4. List of 2 numbers: Percent attack and percent decay.

This should not add up to more than 1.

5. List of lists: Time signatures.

6. Number: Seconds duration.

7. Number: Complexity factor (low is simplicity, high is precision, 10 is a compromise).

OUTPUTS

1. List of voices.

DESCRIPTION

This function scales durations of each chord-seq to fit with an orchestral attack and decay line. This will work with a simple chord, or more elaborated textures.

The quantification method inside is *tree-quant*.

DEMO PATCH

envelope-music-demo

envelope-chord-demo

multiseq-env-poly-legato

INPUTS

1. List of chord-seq's.

2. Number: Tempo.

3. bpf-lib self with 2 bpf's: attack shape and decay shape (1000*1000).

4. List of 2 numbers: Percent attack and percent decay.

This should not add up to more than 1.

5. List of lists: Time signatures.

6. Number: Seconds duration.

7. Number: Complexity factor (low is simplicity, high is precision, 10 is a compromise).

8. Number: Milliseconds rest threshold.

OUTPUTS

1. List of voices.

DESCRIPTION

This function scales durations of each chord-seq to fit with an orchestral attack and decay line. This will work with a simple chord, or more elaborated textures.

The quantification method inside is *tree-quant-legato*.

DEMO PATCH

envelope-music-demo
envelope-chord-demo

multiseq-env-poly-legato-tie

INPUTS

1. List of chord-seq's.
2. Number: Tempo.
3. bpf-lib self with 2 bpf's: attack shape and decay shape (1000*1000).
4. List of 2 numbers: Percent attack and percent decay.
This should not add up to more than 1.
5. List of lists: Time signatures.
6. Number: Seconds duration.
7. Number: Complexity factor (low is simplicity, high is precision, 10 is a compromise).
8. Number: Milliseconds rest treshold.
9. Number: Milliseconds tie treshold.

OUTPUTS

1. List of voices.

DESCRIPTION

This function scales durations of each chord-seq to fit with an orchestral attack and decay line. This will work with a simple chord, or more elaborated textures.

The quantification method inside is *tree-quant-legato-tie*.

DEMO PATCH

envelope-music-demo
envelope-chord-demo

multiseq-extract-param

INPUTS

1. List of chord-seqs.

OUTPUTS

1. List of lists of midicents.
2. List of lists of onsets.
3. List of lists of durations.
4. List of lists of velocities.
5. List of lists of offsets.
6. List of lists of channels.
7. List of legato.

DESCRIPTION

multiseq-extract-param makes parameters separately available for further processes and combinations. The multi-seq is rebuilt through *rebuild-multiseq*, or used as control data for other functions. The example patch *cross-multiseq-demo* uses this as a kind of cross synthesis: Pitches of tam-tam analysis 1 are combined with rhythms of a different tam-tam spectrum 2. In spectral cross synthesis of sounds similar principles are used, and they are here available for both musical material and spectral analysis.

DEMO PATCH

cross-multiseq-demo
controldata-multiseq-demo

multiseq-pointer

INPUTS

1. List of chord-seqs.
2. Bpf (1000*1000): Time pointer. The y-axis points to time of the original, while the x-axis is

the chronological time of the output.

3. Number: Number of time windows (ex. 1000).

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

multiseq-pointer reorders the input music through a time pointer. A straight line 0-1000 returns the original, a straight line 1000-0 returns the retrograde. Any other shape is possible, and will create non-linear scannings of the material.

DEMO PATCH

poly-pointer-demo

poly-pointer-nonlinear-demo

multiseq2poly

INPUTS

1. List of chord-seqs.

2. Number: Tempo.

3. List of lists: Time signatures (if too few, last will be repeated).

4. Float: Complexity factor. Low number means simplicity, large means precision.

OUTPUTS

1. List of voices.

DESCRIPTION

This function will perform a rhythm quantification of the input chord-seqs using the method *tree-quant*. Onsets and duration can be processed in advance to remove zero durations and overlapping notes, as shown in the patch *quantify-instrumentalpart-demo*.

DEMO PATCH

ensemble-circles-lists-gliss

multiseq2poly-legato

INPUTS

1. List of chord-seqs.

2. Number: Tempo.

3. List of lists: Time signatures (if too few, last will be repeated).

4. Float: Complexity factor. Low number means simplicity, large means precision.

5. Number: Milliseconds rest threshold. Rests below the threshold are deleted.

OUTPUTS

1. List of voices.

DESCRIPTION

This function will perform a rhythm quantification of the input chord-seqs using the method *tree-quant-legato*. Onsets and duration can be processed in advance to remove zero durations and overlapping notes, as shown in the demo patch.

DEMO PATCH

quantify-hockets-multiple-tie

multiseq2poly-legato-tie

INPUTS

1. List of chord-seqs.

2. Number: Tempo.

3. List of lists: Time signatures (if too few, last will be repeated).

4. Float: Complexity factor. Low number means simplicity, large means precision.

5. Number: Milliseconds rest threshold. Rests below the threshold are deleted.

6. Number: Milliseconds tie threshold. If time after beat is less than threshold, there will be no tie.

OUTPUTS

1. List of voices.

DESCRIPTION

This function will perform a rhythm quantification of the input chord-seqs using the method *tree-quant-legato-tie*. Onsets and duration can be processed in advance to remove zero durations and overlapping notes, as shown in the demo patch.

DEMO PATCH

quantify-hockets-multiple-tie

multiseq-start-at-0

INPUTS

1. List of chord-seqs.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This function finds the earliest onset, and subtracts it from all other onsets, to make the multi-seq start at 0.

DEMO PATCH

extract-onset-range-demo

multiseq-2-chordseq

INPUTS

1. List of chord-seqs.

OUTPUTS

1. Chord-seq.

DESCRIPTION

This will join all notes and chords into a single chord-seq.

my-instrument-multiseq

INPUTS

1. List of chord-seqs.

2. List of possible midicents on the instrument.

3. Maximum distance from notes within the mode. 50 cents will leave no gaps in a semitone scale, but duplicates may occur in scales containing smaller intervals. 50 will return notes if distance is equal or larger than -50 and smaller than 50.

OUTPUTS

1. List of chord-seqs without pitch approximation.

2. List of chord-seqs with pitch approximation.

DESCRIPTION

my-instrument-multiseq will keep only pitches within a certain distance (input 3) to a note of the midicent pitch-list (input 2). If these ranges overlap there may be duplicate notes. The function *remove-dup-multiseq* can be used to delete duplicate notes within each chord.

There are several possible uses of *my-instrument-multiseq*:

- Adapting a material to notes of a particular instrument. Creating a library of possible instrumental ranges will create a tool for orchestration. It is not given that the result automatically will be playable, as for instance microtonal fingering combinations are complicated on woodwind instruments. Elvio Cipollone has worked on an in progress database of possible fingering combinations,⁸ which could be a step towards creating more ideomatic raw materials from Open Music.
- Approximating a material to any microtonal tuning given as midicents. This is useful for synthesis through export to Csound, where the intonation can be exact. The Open Music musical notation can show a maximum division of 1/16-tones.

DEMO PATCH

quantify-instrumentalpart-demo

piano-range

⁸ [Multiple_authors], 2008, The OM composer's book.2, p. 107.

INPUTS

1. Dummy, nothing needs to be connected.

OUTPUTS

1. List of possible midicents for the piano.

DESCRIPTION

This is a simple list of notes for a particular instrument, useful together with *my-instrument-multiseq*.

DEMO PATCH

quantify-instrumentalpart-demo

pitchshift-multiseq

INPUTS

1. List of chord-seqs.
2. List of 2 bpf's: For modulation of the lower and upper limit of the pitchshift range.
3. List of list of 4 numbers/Second output of chord-seq: 4 notes for pitchshift range: ((InitialMidicentMinimum)(InitialMaximum)(FinalMinimum)(FinalMaximum))
4. Number: Scaling of the bpf modulation. If 0, there will be a straight line from initial to final range.
5. Number: Cents interval size. 50 is 1/4-tone, 25 is 1/8-tone, 1 is also possible, for high pitch resolution and export to synthesis.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Within sound processing, a transposition would be moving the sound to a new fundamental while keeping the original intervals within the spectrum. Pitchshifting involves distorting the spectrum by also changing interval sizes. *pitchshift-multiseq* works as a transposition if the pitchshift range is of the same size as the total range of the input music. Any other ranges will create interval distortions which form a type of musical pitchshifting.

DEMO PATCH

pitchshift-demo

pitchshift-poly

INPUTS

1. List of voices.
2. List of 2 bpf's: For modulation of the lower and upper limit of the pitchshift range.
3. List of list of 4 numbers/Second output of chord-seq: 4 notes for pitchshift range: ((InitialMidicentMinimum)(InitialMaximum)(FinalMinimum)(FinalMaximum))
4. Number: Scaling of the bpf modulation. If 0, there will be a straight line from initial to final range.
5. Number: Cents interval size. 50 is 1/4-tone, 25 is 1/8-tone, 1 is also possible, for high pitch resolution and export to synthesis.

OUTPUTS

1. List of voices.

DESCRIPTION

Within sound processing, a transposition would be moving the sound to a new fundamental while keeping the original intervals within the spectrum. Pitchshifting involves distorting the spectrum by also changing interval sizes. *pitchshift-poly* works as a transposition if the pitchshift range is of the same size as the total range of the input music. Any other ranges will create interval distortions which form a type of musical pitchshifting.

DEMO PATCH

pitchshift-demo

point2pointcalculator

INPUTS

1. List of numbers: First proportion list.
2. Number: Position in first proportion list (index starting at 1).
3. List of numbers: Second proportion list.
4. Number: Position in second proportion list (index starting at 1).

OUTPUTS

1. Number: Ratio before.
2. Number: Ratio after.

DESCRIPTION

point2pointcalculator calculates ratios before and after between points within 2 lists of proportions. These ratios are used as inputs to *augmentator*. The goal was starting with main macro rhythms for a whole piece, then finding diminishment of the proportions spanning between exact points of the main macro rhythms. I have used this as an organizing principle for several pieces.

DEMO PATCH

macro-rhythm-demo

poly-pointer

1. List of chord-seqs.
2. Bpf (1000*1000): Time pointer. The y-axis points to time of the original, while the x-axis is the chronological time of the original.
3. Number: Length scaling (1 is original speed).
4. Number: Number of time windows (ex. 1000).

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

poly-pointer reorders the input music through a time pointer. A straight line 0-1000 returns the original, a straight line 1000-0 returns the retrograde. Any other shape is possible, and will create non-linear scanings of the material.

DEMO PATCH

poly-pointer-demo

poly-pointer-nonlinear-demo

poly2multiseq

INPUTS

1. List of voices.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Removes rhythm quantification. This is useful for processing the music as chord-seqs through the other functions. The result can be quantified again using *multiseq2poly*, *multiseq2poly-legato* or *multiseq2poly-legato-tie*.

profile-dur-multiseq

INPUTS

1. List of chord-seqs
2. Bpf-lib self: 1 bpf(1000*1000) for each chord-seq. If there are too few, the last will be repeated. If there are too many, the list will be cropped.

OUTPUTS

1. List of chord-seq.

DESCRIPTION

Pitches of the input material will be sorted by duration, with the bpf curve as an index. 0 means the shortest duration. 1000 means the longest duration. Onsets will not be changed, only the order of notes (including channel and velocity information). These functions are especially interesting sorting a spectral analysis, as all parameters will naturally be in flux.

DEMO PATCH

profile-dur-multiseq-demo

profile-dyn-multiseq

INPUTS

1. List of chord-seqs
2. Bpf-lib self: 1 bpf(1000*1000) for each chord-seq. If there are too few, the last will be repeated. If there are too many, the list will be cropped.

OUTPUTS

1. List of chord-seq.

DESCRIPTION

Pitches of the input material will be sorted by dynamic level (MIDI velocity 0-127), with the bpf curve as an index. 0 means the softest note. 1000 means the loudest note. Onsets will not be changed, only the order of notes (including channel and velocity information). These functions are especially interesting sorting a spectral analysis, as all parameters will naturally be in flux.

DEMO PATCH

profile-dyn-multiseq-demo

profile-multiseq

INPUTS

1. List of chord-seqs
2. Bpf-lib self: 1 bpf(1000*1000) for each chord-seq. If there are too few, the last will be repeated. If there are too many, the list will be cropped.

OUTPUTS

1. List of chord-seq.

DESCRIPTION

Pitches of the input material will be sorted again, with the bpf curve as the new pitch curve. 0 means the lowest note. 1000 means the highest note. Onsets will not be changed, only the order of notes (including channel and velocity information).

DEMO PATCH

profile-multiseq-demo

random-bpfs

INPUTS

1. List of 2 numbers: Y-minimum, y-maximum.
2. List of 2 numbers: Points-minimum, points-maximum.
3. Number: Number of bpfs.

OUTPUTS

1. List of bpfs.

DESCRIPTION

This function creates a list of randomized bpfs for bpf-lib. X will always be between 0 and 1000, y is defined by input 1. Numbers are floats, as this will give more precision for many processes.

DEMO PATCH

ensemble-circles

random-sawtooth-bpfs

INPUTS

1. List of 2 numbers: Y-minimum, y-maximum.
2. List of 2 numbers: Points-minimum, points-maximum.
3. Number: Number of rotations.
4. Number: Number of bpfs.

OUTPUTS

1. List of bpfs.

DESCRIPTION

This function creates a list of randomized bpf's for bpf-lib, with modulo operation on the y-axis. This is useful for spatial movements, as a curve can be split into multiple rotations. Split curves will have sawtooth shapes. X will always be between 0 and 1000, y is defined by input 1. Numbers are floats, as this will give more precision for many processes.

DEMO PATCH

ensemble-circles

random-selection-multiseq

INPUTS

1. List of chord-seqs.
2. Number: Percent (0-100) of notes to keep.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This function selects randomly a certain percent of the notes. The result will be different at each evaluation.

DEMO PATCH

pitch-manipulation-demo

random-spline-bpfs

INPUTS

1. List of 2 numbers: Y-minimum, y-maximum.
2. List of 2 numbers: Points-minimum, points-maximum.
3. Number: Number of bpf's.

OUTPUTS

1. List of bpf's.

DESCRIPTION

This function creates a list of randomized bpf's for bpf-lib, with a spline interpolation between each y-point. X will always be between 0 and 1000, y is defined by input 1. Numbers are floats, as this will give more precision for many processes.

DEMO PATCH

ensemble-circles

retune-poly

INPUTS

1. List of voices.
2. Flatlist of mode.

OUTPUTS

1. Voice approximated to mode.

DESCRIPTION

Approximates midicent values to nearest element of the input list. See also *my-instrument-multiseq*.

return-channels-multiseq

INPUTS

1. List of chord-seqs.
2. Number or list of MIDI channels.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Return every note and chord with the corresponding channel. The function will go through the MIDI channels input and search all chord-seqs for each channel. Empty results will be deleted. This is useful to sort instruments into a different score order.

retune-scordatura-multiseq

INPUTS

1. List of chord-seqs.
2. Chordseq self, containing 2-note chords. First note of the chord (show order while editing chord) is reference pitch, second chord note is sounding pitch after scordatura.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Chord-seqs are retuned to a scordatura tuning. This is useful for a 1/4-tone harp scordatura, while string instruments are more complicated, as multiple strings can be used to play the same pitch. Enharmonic harp notation cannot be solved directly, as Open Music does only display microtones as sharps.

rebuild-chordseq

INPUTS

Chord-seq datalist (from the *instead-of-chordseq* list format).

OUTPUTS

Chord-seq self.

DESCRIPTION

rebuild-chordseq restores the chord-seq from the alternative list format from *instead-of-chordseq*. Lambda patches used as input for *gestures-chordseq* output this type of lists. This function is thus useful for testing the gesture functions on a single chord-seq, before putting them in lambda mode.

rebuild-multiseq

INPUTS

1. List of lists of midicents.
2. List of lists of onsets.
3. List of lists of durations.
4. List of lists of velocities.
5. List of lists of offsets.
6. List of lists of channels.
7. List of legato.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

rebuild-chordseq used together with several *multiseq-extract-param* allows combining parameters of different multi-seqs. Lists lengths usually don't match. In these cases the shortest list length will be used. A reconnection of different parameters from different sources allows a type of musical 'morphing'.

DEMO PATCH

cross-multiseq-demo

reduce-multiseq-to-chord

INPUTS

1. List of chord-seqs.

OUTPUTS

1. Chord self.

DESCRIPTION

This function joins all notes and chords into a single chord. Notes in a chord do not have individual onsets, while other parameters are kept. To split chord notes into separate chord-seqs again, use *split-chord-to-multiseq*.

DEMO PATCH

envelope-chord-demo

remove-dup-multiseq

INPUTS

1. List of chord-seqs.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This function removes duplicate notes within each chord.

remove-emphy-chordseqs

INPUTS

1. List of chord-seqs.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Many processes are at the moment not programmed to handle chord-seqs with 0 or only 1 note. This function will remove chord-seqs without notes.

remove-emphy-voices

INPUTS

1. List of voices.

OUTPUTS

1. List of voices.

DESCRIPTION

Many processes are at the moment not programmed to handle voices with 0 or only 1 note. This function will remove voices without notes.

remove-singlenote-chordseqs

INPUTS

1. List of chord-seqs.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Many processes are at the moment not programmed to handle chord-seqs with 0 or only 1 note. This function will remove chord-seqs with only 1 note.

remove-twonote-chordseqs

INPUTS

1. List of chord-seqs.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

Many processes are at the moment not programmed to handle chord-seqs with 0 or only 1 note. Only 2 notes can also be a problem for *multiseq-pointer*. This function will remove chord-seqs with only 2 notes.

scale2inversion

INPUTS

1. List of voices.
2. Bpf self (y: -1000 to 1000, x: 0 to 2000)

OUTPUTS

1. List of voices.

DESCRIPTION

scale2inversion scales intervals of the input music by a bpf-curve. A static curve at -1000 will create a normal inversion, as known from music history. A static curve at 1000 will return the original. A static curve at 0 will give only note repetitions. 500 will diminish intervals from semitones to 1/4-tones. Any fluctuating curve will create a development of interval size where positive is scaling, and negative is inversion and scaling. In combination with *multiseq-pointer* or *poly-pointer*, these functions can create the Modus Quarternion, as well as any fluctuating state in between.

To process multi-seq the same way, check *scale2inv-multiseq*. *scale2inv-poly* is a newer version where x is kept between 0 and 1000.

DEMO PATCH

scale2inversion-demo

scale2inv-multiseq

INPUTS

1. List of chord-seqs.
2. Bpf self: (y: -1000 to 1000, x: 0 to 1000).

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

scale2inv-multiseq scales intervals of the input music by a bpf-curve. A static curve at -1000 will create a normal inversion, as known from music history. A static curve at 1000 will return the original. A static curve at 0 will give only note repetitions. 500 will diminish intervals from semitones to 1/4-tones. Any fluctuating curve will create a development of interval size where positive is scaling, and negative is inversion and scaling. In combination with *multiseq-pointer* or *poly-pointer*, these functions can create the Modus Quarternion, as well as any fluctuating state in between.

DEMO PATCH

scale2inversion-demo

scale2inv-poly

INPUTS

1. List of voices.
2. Bpf self: (y: -1000 to 1000, x: 0 to 1000).

OUTPUTS

1. List of voices.

DESCRIPTION

scale2inv-poly scales intervals of the input music by a bpf-curve. A static curve at -1000 will create a normal inversion, as known from music history. A static curve at 1000 will return the original. A static curve at 0 will give only note repetitions. 500 will diminish intervals from semitones to 1/4-tones. Any fluctuating curve will create a development of interval size where positive is scaling, and negative is inversion and scaling. In combination with *multiseq-pointer* or *poly-pointer*, these functions can create the Modus Quarternion, as well as any fluctuating state in between.

DEMO PATCH

scale2inversion-demo

sciarrino-passage

INPUTS

1. List: Midicent.
2. Number: Milliseconds duration.
3. Number: Speed ratio. 1 is normal, .5 is half speed, 2 is double speed.

OUTPUTS

1. Chordseq-datalist. Connecting this to *rebuild-chordseq* will create a chord-seq from these lists.

DESCRIPTION

This is an example of the gesture patches. Use this directly or as an input abstraction in lambda mode (select function, type 'l'). A list of gesture-patches in lambda mode can serve as abstractions for the *gestures-chordseq* function. This will place the gestures into a context of harmony and timing.

We will create all the gesture patches with the same 3 inputs for compatibility, even though parameters may be ignored by the actual function.

DEMO PATCH

gestures-sciarrino-generator

select-longest-multiseq

INPUTS

1. List of chord-seqs.
2. Number: Milliseconds minimum duration.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

select-longest-multiseq extracts only notes above minimum duration. This is useful to reduce the amount of information from the pm2 library,⁹ and keep only the most stable partials of a sound.¹⁰

DEMO PATCH

durations-filter-demo

select-loudest-multiseq

INPUTS

1. List of chord-seqs.
2. Number: Velocity treshold (within the midi range 1-127).

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

List of chord-seqs where only notes above dynamic treshold are maintained. This could be used to reduce vast amount of information within a spectral analysis.

DEMO PATCH

dynamics-filter-demo

select-multiseq-range

INPUTS

1. List of chord-seqs.
2. List of 2 numbers: Start milliseconds, end milliseconds.

OUTPUTS

1. List of chord-seqs

DESCRIPTION

Selects notes within the milliseconds range. See also *extract-onset-range-multiseq* and *extract-onset-fragment-multiseq*.

DEMO PATCH

extract-onset-range-demo

select-n-random-multiseq

INPUTS

1. List of chord-seq.

⁹ <http://support.ircam.fr/docs/om-libraries/main/co/OM-pm2.html>

¹⁰ The software Spear can do similar selection of partials below a certain duration on a sonogram: <http://www.klingbeil.com/spear>

2. Number: Number of notes to extract from each chord-seq.

OUTPUTS

1. List of chord-seq.

DESCRIPTION

Selects an exact number of notes to extract from each chord-seq. Use this instead of *random-selection-multiseq* when the number of notes is important.

DEMO PATCH

pitch-manipulation-demo

select-shortest-multiseq

INPUTS

1. List of chord-seqs.

2. Number: Milliseconds maximum duration.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

select-shortest-multiseq extract only notes below maximum duration. This can be used to extract the most noisy and unstable components from a spectrum.

DEMO PATCH

durations-filter-demo

select-softest-multiseq

INPUTS

1. List of chord-seqs.

2. Number: Velocity treshold (within the midi range 1-127).

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

List of chord-seqs where only notes below dynamic treshold are maintained.

DEMO PATCH

dynamics-filter-demo

sort-chord-notes

INPUTS

1. Chord self.

2. Bpf self (1000*1000): A straight line from 0 to 1000 will simply sort notes from high to low pitch. Any sorting shape is possible.

OUTPUTS

1. Chord self.

DESCRIPTION

An order of notes in a chord is sorted after a curve shape. While order is not immediately visible within the chord object, it will be significant when passing it through the function *split-chord-to-multiseq*.

DEMO PATCH

envelope-chord-demo

sort-chordseqs-by-pitch

INPUTS

1. List of chord-seqs.

2. Bpf self (1000*1000): A straight line from 0 to 1000 will simply sort notes from high to low pitch. Any sorting shape is possible. The first pitch of each chord-seq is used as the reference.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This sorts chord-seqs into score order from high to low. This is useful for instance to rearrange chord-seqs with diverse ranges to a score for strings.

DEMO PATCH

knoppskyting-demo

sort-onsets-multiseq

INPUTS

1. List of chord-seqs.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This is a fix for processes returning results where notes are not in the correct order. This sorting can solve the problem when other processes might be confused from this.

spectralize-rand-chordnote

INPUTS

1. Chord-seq or list of chord-seqs.

2. Number: Number of octaves down from chosen fundamental. If outside the midrange, it will be transposed up.

3. Number of partials from this fundamental.

OUTPUTS

1. Flatlist of midicents.

DESCRIPTION

This function chooses a random midicent value from the input material. This is used as the fundamental of an overtone series. This can be an input for *my-instrument-multiseq*, to approximate the material to a more resonant tuning. If input 2 is a high number, the virtual fundamental will be deep, and the grid intervals will be finer. Make sure input 3 is high enough to cover the whole register (hundreds of partials if the fundamental is deep).

spline-ascending-bpfs

INPUTS

1. List of 2 numbers: Y-minimum, y-maximum.

2. List of 2 numbers: Points-minimum, points-maximum.

3. Number: Number of bpfs.

OUTPUTS

1. List of bpfs.

DESCRIPTION

This function creates a list of randomized bpfs for bpf-lib. Random y-points are sorted in ascending order, with spline interpolations. Sketches by Iannis Xenakis for the orchestral work *Terretektorh* show a heterophony of ascending pitch curves, which could be simulated through this function. X will always be between 0 and 1000, y is defined by input 1. Numbers are floats, as this will give more precision for many processes.

DEMO PATCH

ensemble-circles

spline-descending-bpfs

INPUTS

1. List of 2 numbers: Y-minimum, y-maximum.

2. List of 2 numbers: Points-minimum, points-maximum.

3. Number: Number of bpfs.

OUTPUTS

1. List of bpfs.

DESCRIPTION

This function creates a list of randomized bpf's for bpf-lib. Random y-points are sorted in descending order, with spline interpolations. X will always be between 0 and 1000, y is defined by input 1. Numbers are floats, as this will give more precision for many processes.

DEMO PATCH

ensemble-circles

split-chord-to-multiseq

INPUTS

1. Chord self.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

This function creates one chord-seq for each note in the chord. This is a reverse process of *reduce-multiseq-to-chord*.

DEMO PATCH

envelope-chord-demo

surround-poly-rests

INPUTS

1. List of lists: Silent time signatures before fragment.

2. Poly self.

3. List of lists: Silent time signatures after fragment.

OUTPUTS

1. Poly self.

DESCRIPTION

This function will add empty measures at the beginning and end of a poly. There are several uses for this:

- Synchronizing materials from multiple sources into a larger score.
- Adding margin measures for the etf export. Opening etf-files still works in Finale 2011, while the format has been officially abandoned for future Finale versions. It will work, but there are some buggy behaviours. The first measure of first voice will usually disappear. For this reason it is a good idea to add empty measures before and after the music exported from Open Music. In some cases it will be very hard to place expressions and text correctly over imported measures in Finale. I cannot find a rule for this problem, but extra measures at the beginning and end often makes it easier. By selecting the middle measures, you avoid copying problem measures into your document.

time-scaler

INPUTS

1. List of chord-seqs.

2. List of 2 numbers: Scaling factor minimum and maximum. 1 is original speed, .5 is double speed, 2 is half speed.

3. Bpf self (1000*1000): Curve where the x-axis is time from beginning to end, and the y-axis is speed within the range of input 2.

4. Number: Window resolution (ex. 1000). The number of time windows need to be large enough to create a smooth tempo fluctuation. If the input is very long, the number of windows can be increased.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

time-scaler applies a tempo curve to the input music, solved as a long sequence of tempo changes. By using time windows, different parts of the music will always be kept in sync.

DEMO PATCH

time-scaler-demo

timescale-ligeti-canon

timesign-tempo-dur

INPUTS

1. List of lists: Time signatures.
2. Number: Tempo.

OUTPUTS

1. List: (x min y seconds).

DESCRIPTION

A simple calculation of length.

transpose-music

INPUTS

1. Note, chord, multi-seq, voice or poly self.
2. Number: Transposition interval as midicents, negative or positive.

OUTPUTS

1. Note, chord, multi-seq, voice or poly self.

DESCRIPTION

Simple transposition of all notes and chords, supporting different notation objects in Open Music.

tree-quant-legato-tie

INPUTS

1. List of numbers: Rhythmic proportions. Negative numbers are rests.
2. Number: Seconds duration.
3. Number: Tempo.
4. List of lists: Time signatures. If there are too few, the last will be repeated.
5. Floating number: Multiplication factor. Low number means simplicity, high number means precision. 1 is very simple, 10 is already quite complicated.
6. Number: Milliseconds rest treshold.
7. Number: Milliseconds tie treshold.

OUTPUTS

1. A rhythm tree as input to the voice object.

DESCRIPTION

This is a quantification method, which can use any series of numbers without loss of notes. Very small values will be rounded up. The duration parameter makes it easy to augment rhythms by irrational values. *tree-quant-legato-tie* offers additional control over rests and ties. Rests shorter than treshold will be deleted. If a duration ties across a beat is shorter than the treshold, next note will start on the beat. The basic *tree-quant* function will tie every note not stopping exactly at the beat. For rapid passages, this could create too many ties.

DEMO PATCH

tree-quant-legato-tie-demo

tree-quant-legato

INPUTS

1. List of numbers: Rhythmic proportions. Negative numbers are rests.
2. Number: Seconds duration.
3. Number: Tempo.
4. List of lists: Time signatures. If there are too few, the last will be repeated.
5. Floating number: Multiplication factor. Low number means simplicity, high number means precision. 1 is very simple, 10 is already quite complicated.
6. Number: Milliseconds rest treshold.

OUTPUTS

1. A rhythm tree as input to the voice object.

DESCRIPTION

This is a quantification method, which can use any series of numbers without loss of notes. Very small values will be rounded up. The duration parameter makes it easy to augment rhythms by irrational values. *tree-quant-legato* is *tree-quant* with additional control over rests. Rests shorter than threshold will be deleted. Some times unintentional rests occur, since onsets and durations in a chord-seq are rounded to integers. Other times *tree-quant-legato* can be helpful to create legato interpretations of the rhythms.

DEMO PATCH

tree-quant-legato-demo

tree-quant

INPUTS

1. List of numbers: Rhythmic proportions. Negative numbers are rests.
2. Number: Seconds duration.
3. Number: Tempo.
4. List of lists: Time signatures. If there are too few, the last will be repeated.
5. Floating number: Multiplication factor. Low number means simplicity, high number means precision. 1 is very simple, 10 is already quite complicated.

OUTPUTS

1. A rhythm tree as input to the voice object.

DESCRIPTION

This is a quantification method, which can use any series of numbers without loss of notes. Very small values will be rounded up. The duration parameter makes it easy to augment rhythms by irrational values.

DEMO PATCH

tree-quant-simple-demo

tree-quant-complex-demo

trill-chords

INPUTS

1. Chord self: Initial main chord.
2. Chord self: Initial trill chord.
3. Chord self: Final main chord.
4. Chord self: Final trill chord.
5. List of bpf (1000*1000): Curve added to glissando shape of main chords. If there are too few, last will be repeated. If there are too many, the list will be cropped.
6. List of bpf (1000*1000): Curve added to glissando shape of trill chords. If there are too few, last will be repeated. If there are too many, the list will be cropped.
7. List of numbers: Cents range of the bpf curves. Even positions are for main notes, odd positions are for trill notes.
8. List of numbers: Number of notes for each part.
9. Number: Seconds duration.

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

trill-chords creates a texture of trills between 2 different glissando trajectories. Bpf-shapes allow the transitions to happen in other shapes than straight lines. The function makes it possible to listen to harmonic situations during this transition, even though a glissando notation for a group of strings would only contain the turning points.

DEMO PATCH

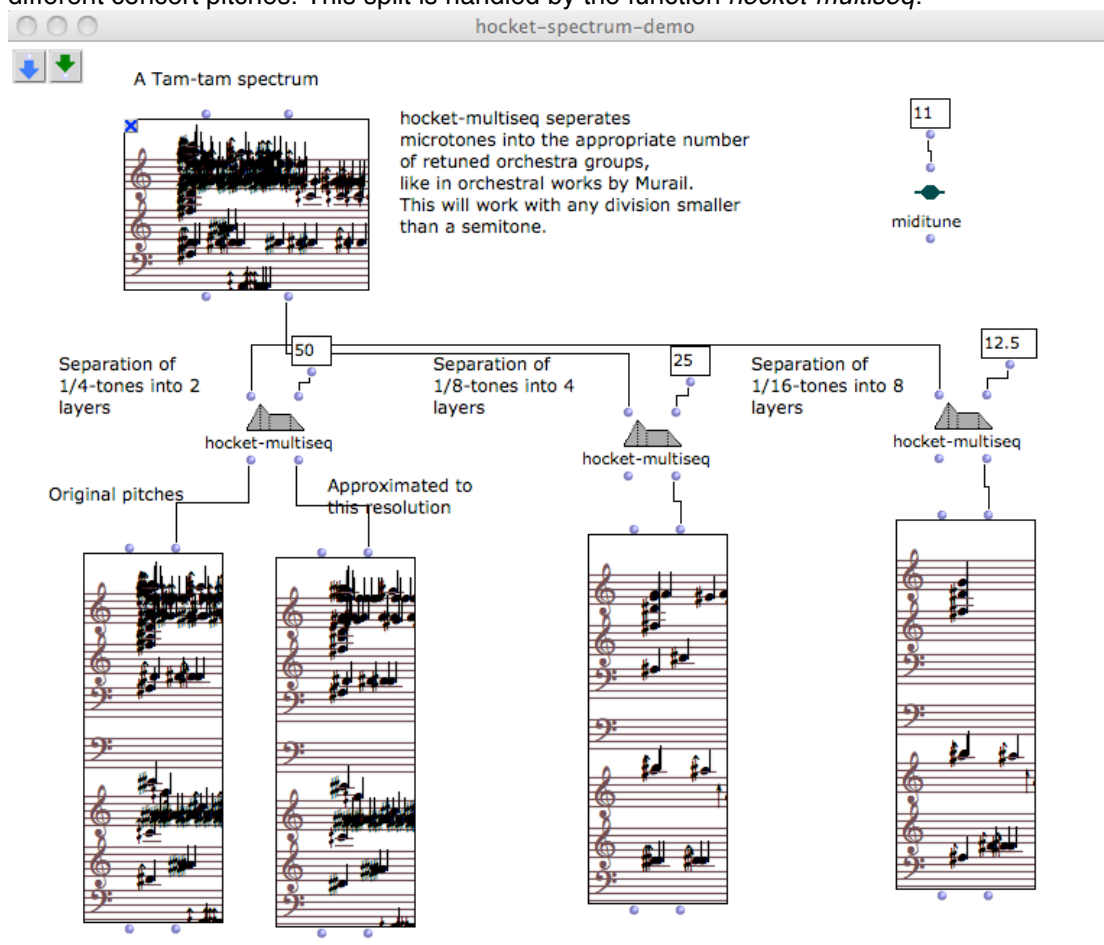
tr-gliss-chords

EXAMPLE PATCHES

The reference part of this manual explains inputs and outputs of the functions. The demo patches give more concrete examples of use. Open these demo patches from Ruben-OM to listen to materials. Evaluate the *miditune* functions to hear 1/8-tone playback from Midishare. If Microplayer is selected from the scores, divisions up to 1/16-tone are handled automatically. These chapters show transformation on different parameters, and some possible combinations of processes. Crowded scores in Open Music are not easy to read, Open Music is not primarily a score editor. Quantified scores can be exported as etf-files for editing with Finale. Lilypond export is available through OmLily¹¹.

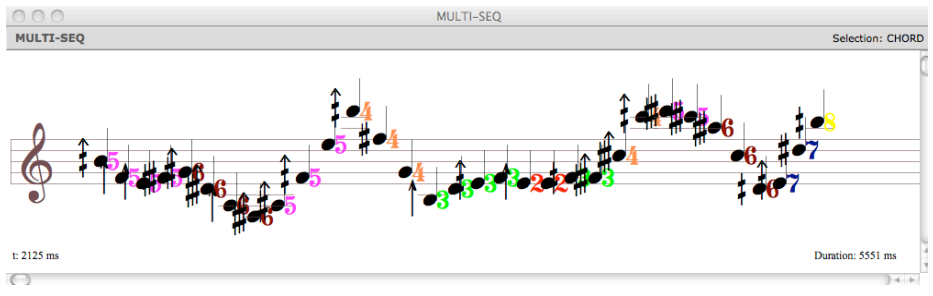
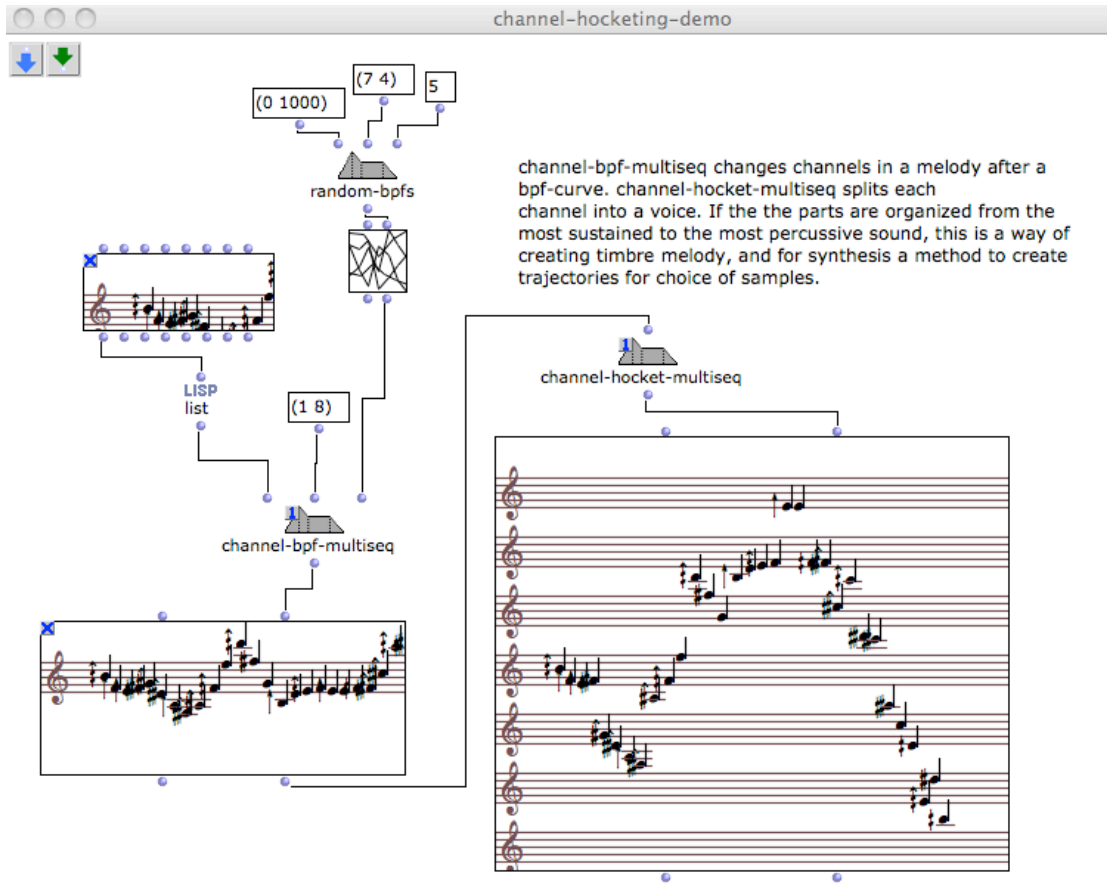
Hockets

Hocket is a medieval compositional technique where materials alternate between multiple voices. A dense microtonal material can be distributed between instrumental groups with different concert pitches. This split is handled by the function *hocket-multiseq*:



¹¹ <http://karim.haddad.free.fr/pages/downloads.html>

A different hocket technique is possible, using a curve shape to split a melodic line between multiple midi channels, and finally separate parts:



The shape of the midi channel curve can be recognized in the resulting score:

t: 3476 ms Duration: 5551 ms

Adding orchestral attack/decay envelopes with *multiseq-env-poly* after the hockets can distort the timing. A monophonic line is folded into polyphony or heterophony.

channel-bpf-env-demo

channel-bpf-multiseq channel-hocket-multiseq multiseq-env-poly

The initial melodic line.



t: 1111 ms

Duration: 5426 ms

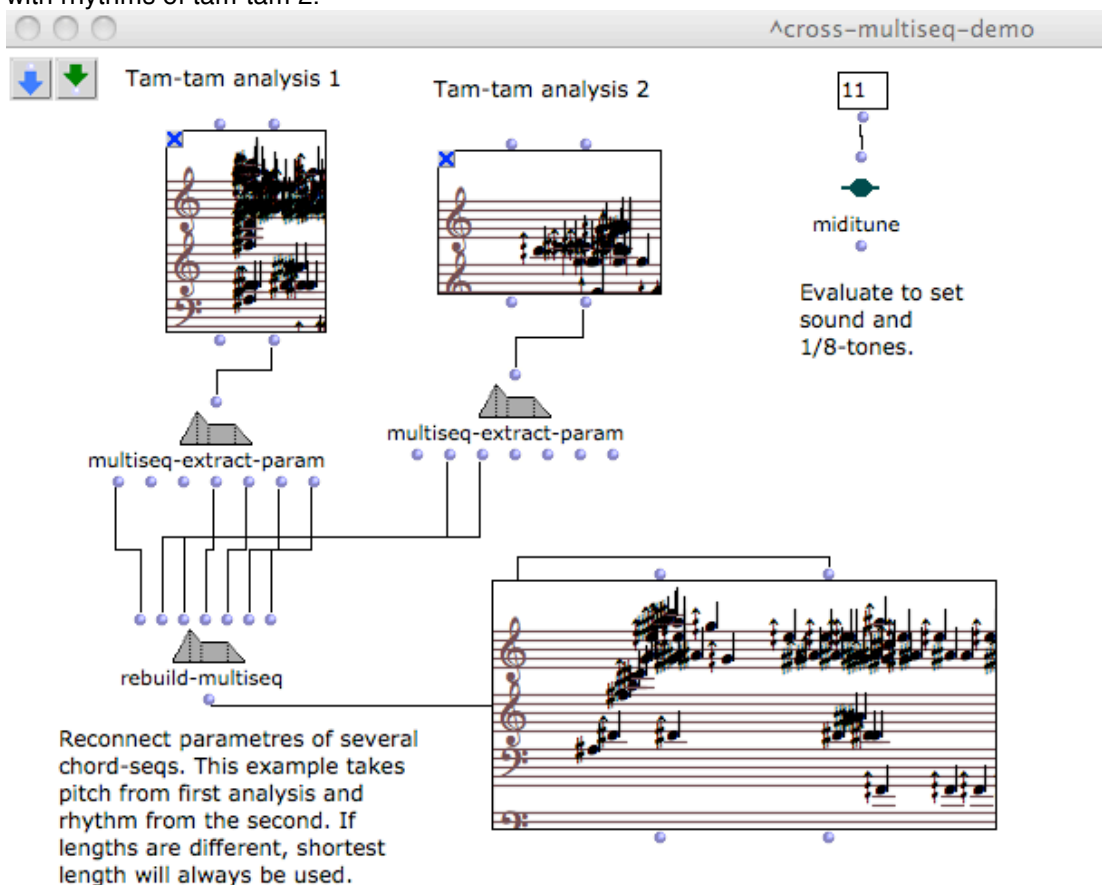
The same line split between multiple parts, and time distorted through these envelopes.

POLY

A complex musical score for a polyphonic piece. It features six systems of staves, each with a treble and bass clef. The notation is highly detailed, with many notes beamed together and some notes having stems pointing in different directions. The score is divided into two measures, with a '2' above the second measure. The overall appearance is that of a dense, multi-layered musical composition.

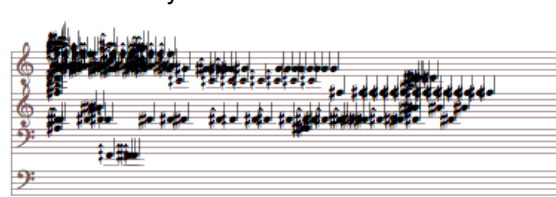
Morphing

Morphing within sound processing is available in Audiosculpt, Soundhack, Csound and other software. Properties of one sound is be applied to a second sound to form hybrid sounds. There are numerous morphing techniques. One of them is to match amplitudes from one sound with remaining data (pitch and time) of another sound. In csound, this is done with the function pvcross or pvcross¹². The example below extracts parameters of 2 tam-tam sounds, performed with different sticks and techniques. The hybrid tam-tam has pitch of tam-tam 1 with rhythms of tam-tam 2.



¹² <http://www.csounds.com/manual/html/pvcross.html>

Tam-tam analysis 1.



t: 2583 ms Duration: 6124 ms

The image shows a musical score for Tam-tam analysis 1. It consists of four staves: two treble clefs and two bass clefs. The notation is dense, featuring many beamed notes and rests, characteristic of a complex rhythmic pattern. The score is contained within a rectangular box.

Tam-tam analysis 2.



t: 9562 ms

The image shows a musical score for Tam-tam analysis 2. It consists of four staves: two treble clefs and two bass clefs. The notation is dense, featuring many beamed notes and rests, characteristic of a complex rhythmic pattern.

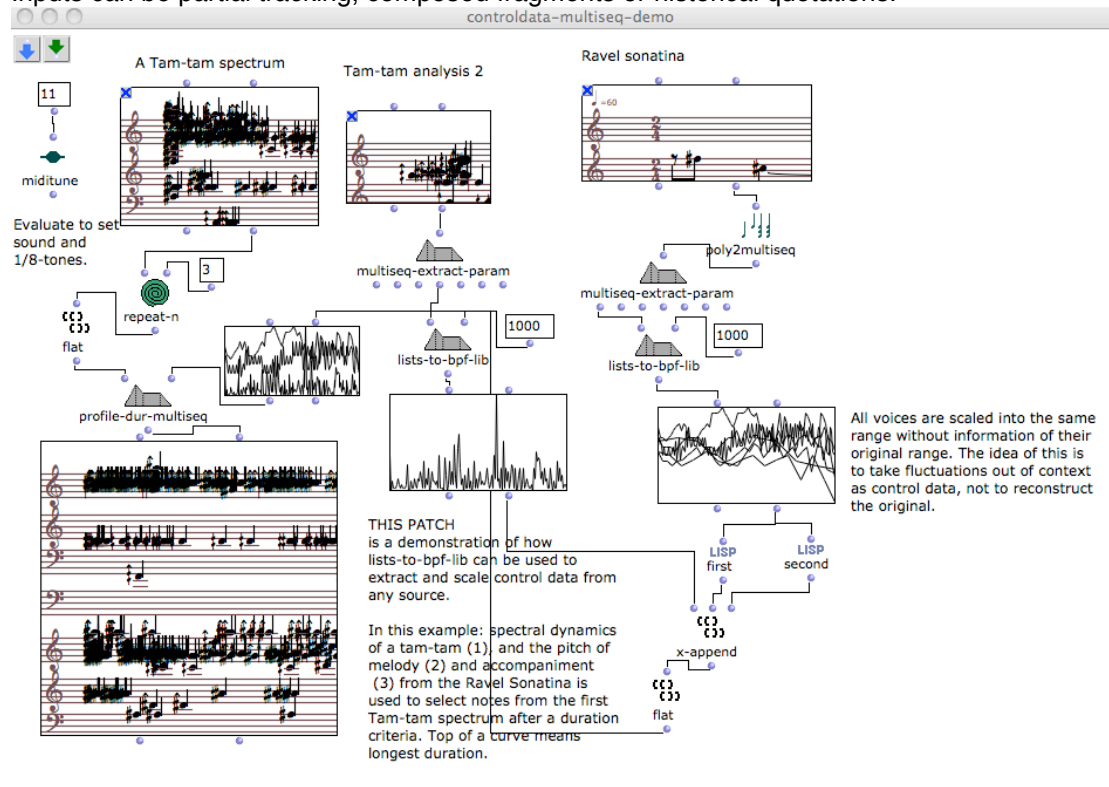
The result is a hybrid tam-tam sound.



t: 3875 ms Duration: 11285 ms

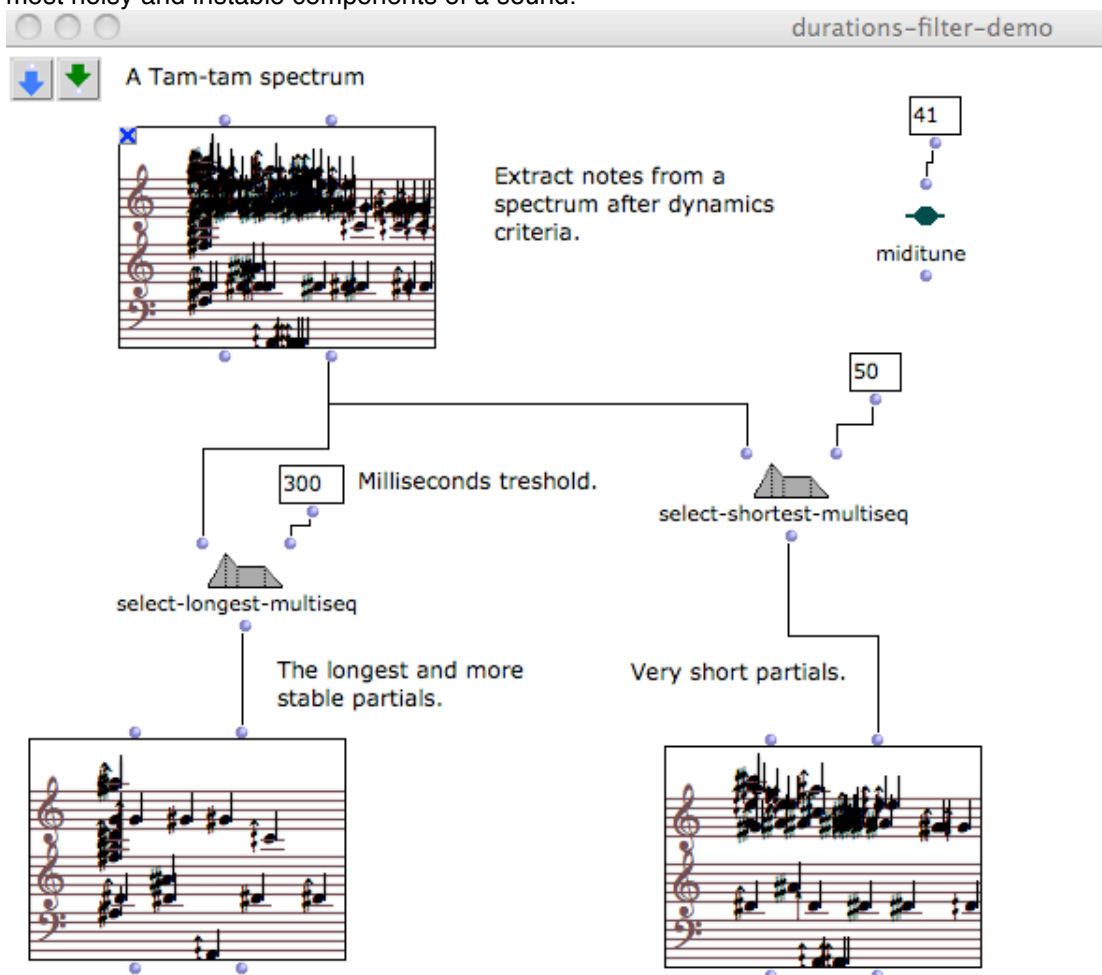
The image shows a musical score for a hybrid tam-tam sound. It consists of four staves: two treble clefs and two bass clefs. The notation is dense, featuring many beamed notes and rests, characteristic of a complex rhythmic pattern.

Partial trackings can easily be treated the same way as other musical materials. Control data is gathered from multiple sources: The input material is reordered by new pitch curves. These curves come from spectral dynamics of a second tam-tam sound, and melodic curves from Ravel's Sonatina, used in many of the following examples. Such connections can be considered 'morphing' of musical materials. We can use the term in a wider sense for a range of fluid transformations of musical materials available through the Ruben-OM patch library. Inputs can be partial tracking, composed fragments or historical quotations.



Filtering by criteria

Partial trackings from the pm2¹³ library can contain a huge amount of information. We can select duration longer than a minimum.¹⁴ This is helpful to remove background noise from the analysis of timbrally stable sounds. Keeping only the short partials means keeping only the most noisy and instable components of a sound.

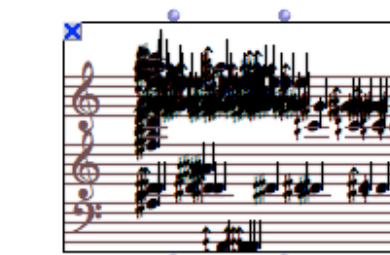


¹³ <http://forumnet.ircam.fr/product/openmusic-libraries/>

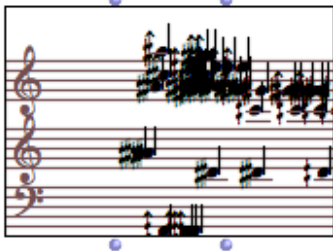
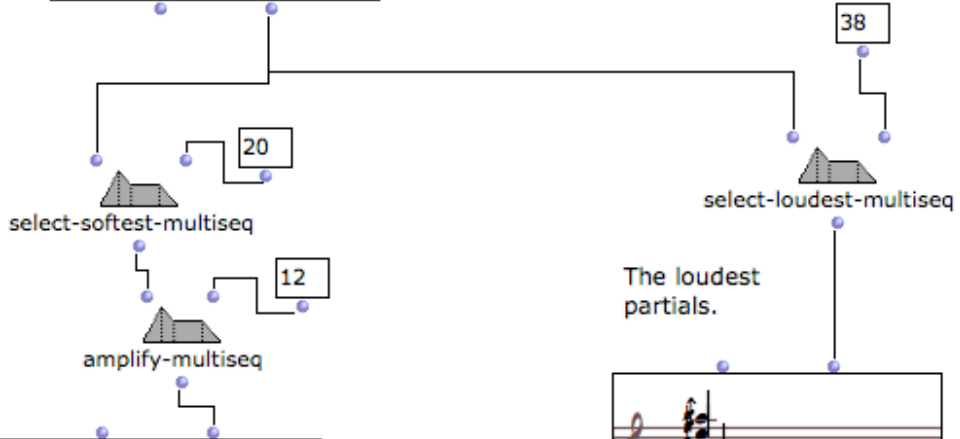
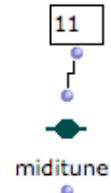
¹⁴ The software Spear can do similar selection of partials below a certain duration on a sonogram: <http://www.klingbeil.com/spear>

Dynamic levels of partials can be a selection criteria.

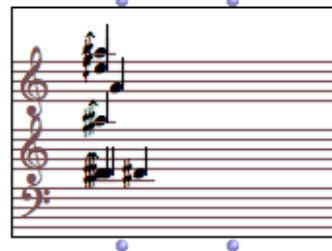
A Tam-tam spectrum



Extract notes from a spectrum after dynamics criteria.



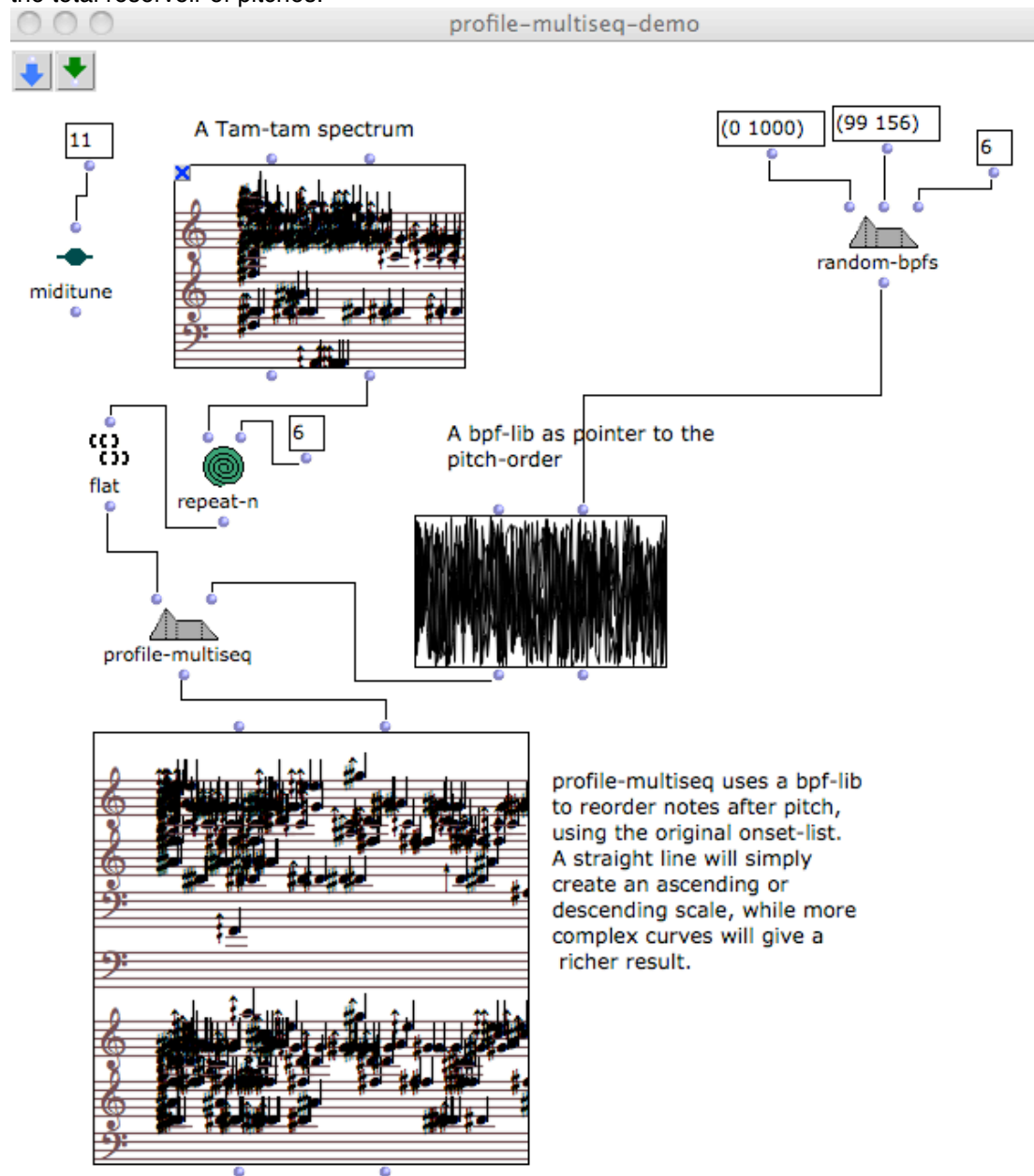
Select the most quiet partials and amplify them.



Sorting curves

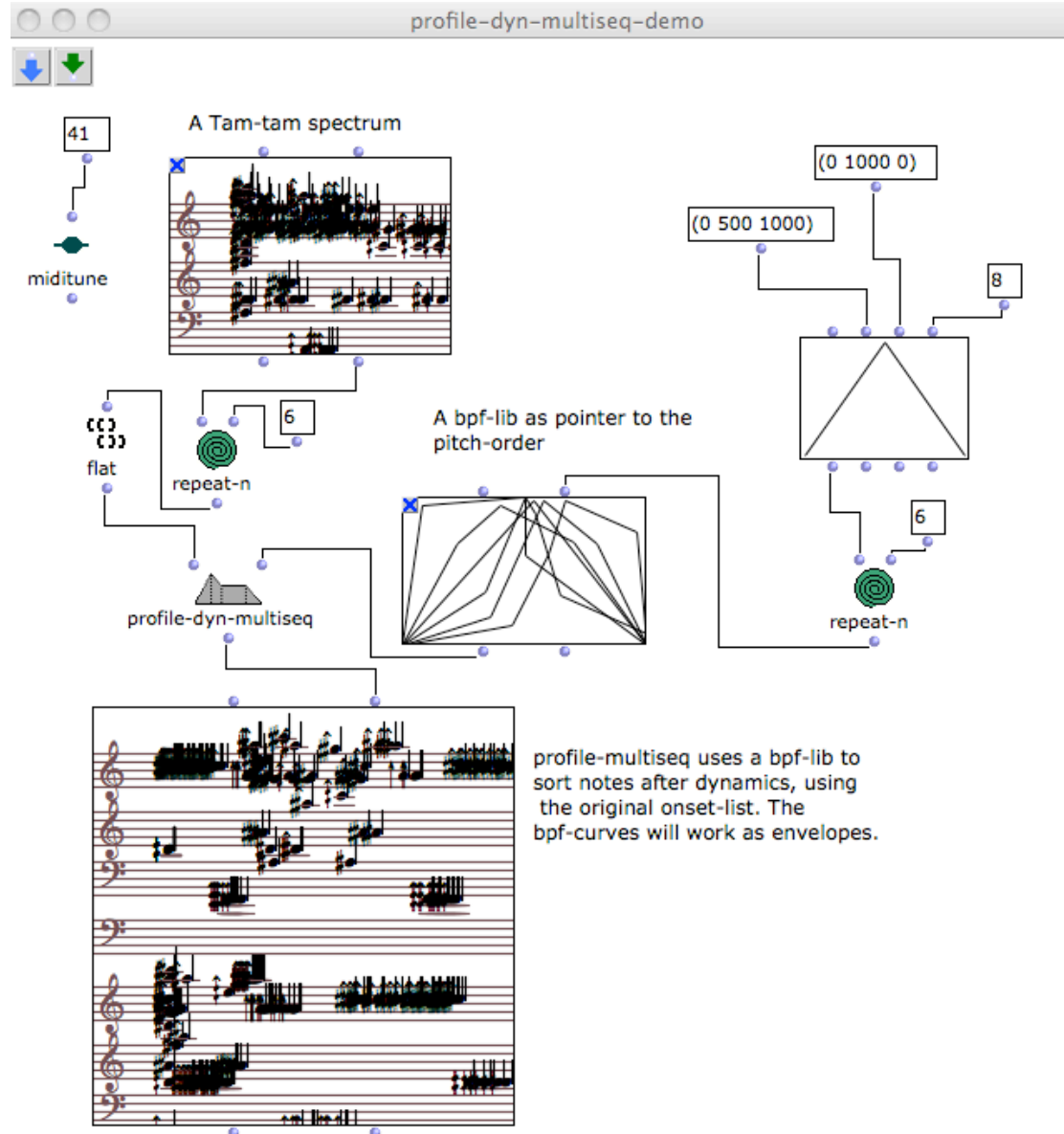
While the previous examples were static filters, the following patches reorder information by curves. These processes work well on spectral analysis, due to their multidimensional nature. Sorting by pitch, duration or dynamics criteria will return different results.

In this example, a chaotic curve is used as the new pitch curve, pointing up or down through the total reservoir of pitches.



The image displays a page of musical notation, likely a score for a string ensemble or orchestra. It consists of four systems of staves. Each system includes a grand staff (treble and bass clefs) and a single bass clef staff. The notation is dense with notes and rests, and includes blue horizontal lines extending from the right side of the staves, possibly indicating a specific performance instruction or a continuation of the piece. The notation is written in black ink on a white background.

Finally, curves can be pointers to dynamic levels of the material. 6 different simple shapes will create a heterophonic crescendo and decrescendo. The attack portion of a percussion instrument is the strongest, so what comes at the top of the curves, comes from the beginning of the tam-tam sound.

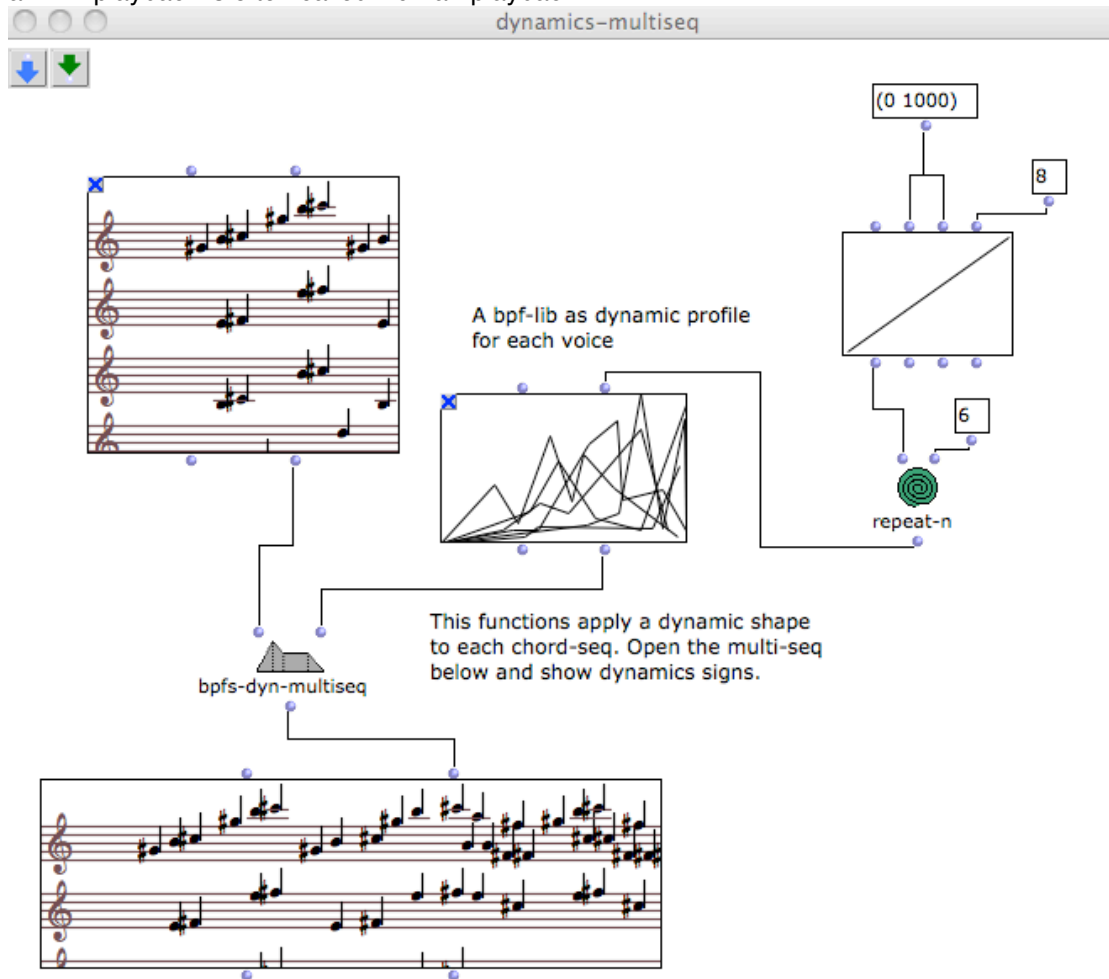


The image displays a musical score for a piano piece, consisting of eight systems of staves. Each system includes a grand staff with a treble clef on the upper staff and a bass clef on the lower staff. The notation is dense, featuring numerous notes, rests, and dynamic markings. The dynamic markings, written in blue, include *ppp* (pianississimo), *pp* (pianissimo), *p* (piano), and *f* (forte). The score is heavily annotated with these markings, indicating a wide range of dynamic control. At the bottom left of the page, a time stamp reads "t: 8708 ms".

Dynamic curves

Examples using partial trackings have a natural variation on all parameters. Manually edited scores have pitch and rhythm, while dynamics are static. It is possible, though tedious, to change midi velocities manually. *bpf-dyn-multiseq* applies dynamic curves to each chord-seq. This is useful to find a more musical MIDI playback, and directly relevant if the material is exported to Csound synthesis. The Om2Csound¹⁵ library exports Csound scores from Open Music.

Time scaling could be considered for a less rigid performance. Introduction of irregularities to a MIDI playback is often called 'human playback'.



Score with dynamic shapes:



¹⁵ <http://support.ircam.fr/docs/om-libraries/main/co/OM2Csound.html>

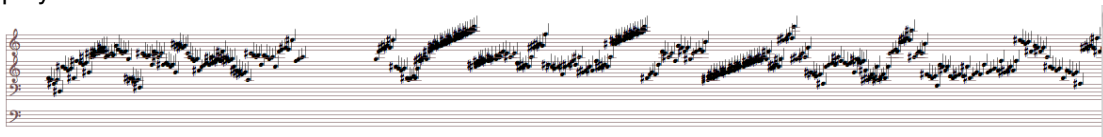
Gestures

Basic Open music patches can deal with particular types of musical gestures, while the maquette is designed to work on a macro level. I have chosen not to work with maquette, as rhythmic synchronization and communication between different patches are complicated to handle.

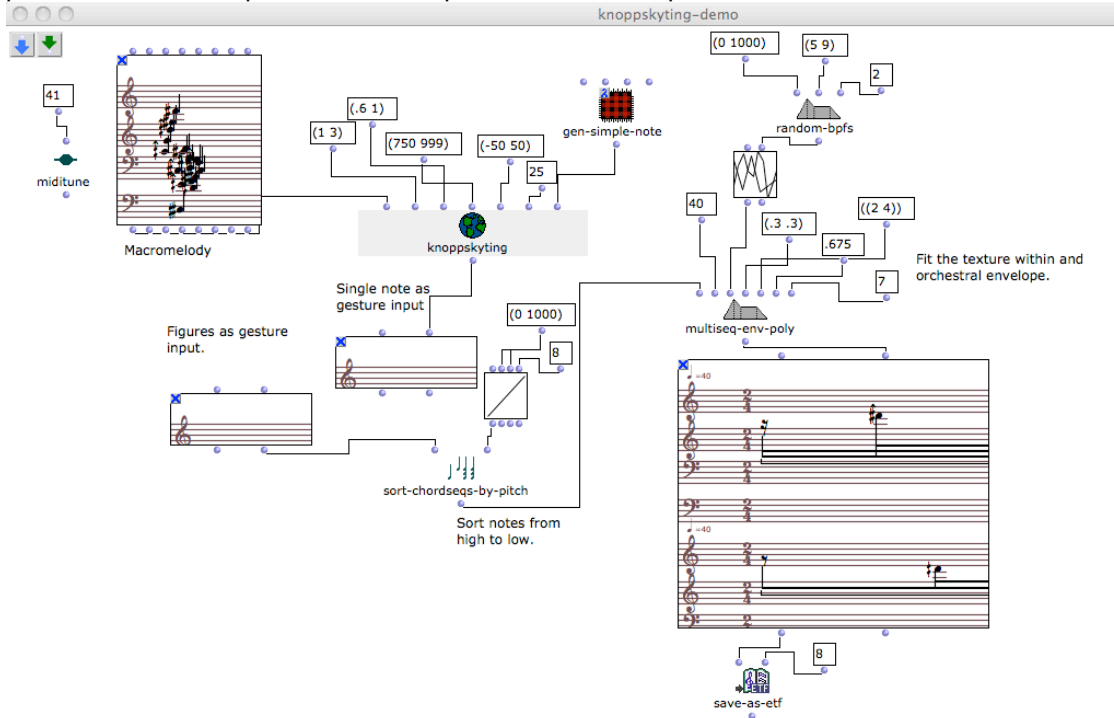
gestures-chordseq organizes a composition of patches through a macro melody, a repertoire of gestures, and scripts for gesture choices, duration of each, speed and silence before each. Two piano gestures used by Salvatore Sciarrino has been used as an example. The input patches are lambda abstractions, and it's possible to expand a repertoire of specific gestures.

This is an example of work of gesture patches on a meta level. I use 2 types of gestures from Sciarrino's piano music as an example. A macro melody and scripts control the development of gestures, randomized on a high level or planned. This is useful as patch for ideomatic gestures, and very particular musical ideas. Keep creating lambda patches of gestures to work with this high level control.

This is the "Sciarrino" score. The scripts are randomized in this patch, and the score will be different at each evaluation. The gestures have dynamic shapes, for a more expressive playback.



Another idea could be using a macro melody, and let a number of new melodies start in proximity to each note of the macro melody. *knoppskyting* was an early version of the gesture patch idea. Gesture patches with 4 inputs are used, it is possible to control interval sizes.



Principles of these tree branches are:

- Each submelody start simultaneously with it's source note (time blur could be added for the future).
- The melody pitchrange equals the interval to next note of the macro melody, except for the last macro note (which repeats the last interval).
- The resulting submelody is transposed to a proximity to it's source note.
- The macro melody is there as an organizing principle, but it is not itself present in the output. It has been split into a heterophony.

The result of this patch is a short orchestral outburst created with knoppskyting and orchestral envelopes.

The image displays a musical score for an orchestral outburst, organized into four main sections: Violins, Violas, Celli, and Contrabasses. Each section contains multiple staves, with some staves featuring a thick black bar indicating a specific musical event or envelope. The score is written in a standard musical notation style, including clefs, time signatures, and various musical symbols. The overall structure suggests a complex, multi-layered orchestral texture.

A long macro melody would create a vast amount of parts. The *ensemble-circles* functions makes is easier to control number of parts.

Ensemble circles

ensemble-circles contains the *gestures-chordseq* function, with additional structural organization to handle multiple parts. Curve shapes create orchestral arpeggios of chords, or serve as spatialization trajectories for a spatially distributed ensemble. Blurring parameters can distort the attack envelopes or control overlaps between parts.

ensemble-circles

Use bpf to create circling movements between different instruments in an ensemble.

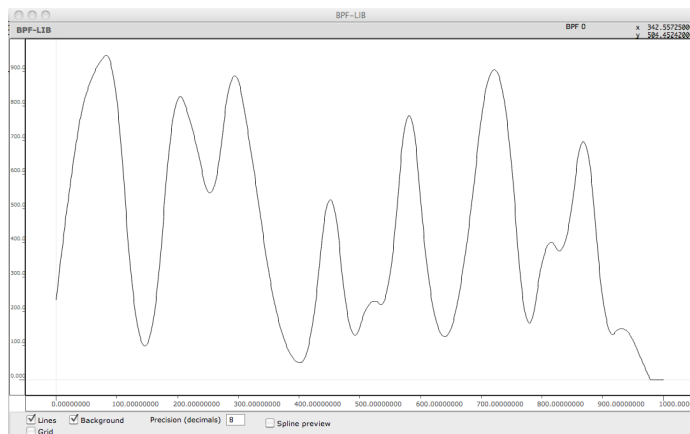
INPUTS

1. Bpf (1000*1000): Trigger curve for arpeggio between multiple parts. The curve is divided into number of parts on the y-axis. A new note or event will start each time the curve crosses the border between parts.
2. Number: Overall duration seconds.
3. List of 2 numbers: Attack blur seconds minimum/maximum. (0 0) means no change. Numbers will be kept positive, and attacks will not extend past next attack.
4. List of 2 numbers: Duration blur seconds minimum/maximum. (0 0) means no change. Negative numbers will create staccato, positive numbers will create overlaps. However, the note will not extend past the next attack of a part, in order not to obstruct the curve shapes.
5. Number: Number of curve samples. 1000 could be a good compromise. With small numbers, the function will not detect all movements of the curve.
6. List of patches in lambda mode. These must have the following standard structure:
 - INPUTS: 1. List of lists of midicents.
 - 2. Number: Milliseconds duration.
 - 3. Number: Speed ratio (1 is default speed, .5 is half speed).
7. Chord-seq self: Each note or chord will be the base of the input gestures. Number of notes or chords is the same as the number of parts in the ensemble-circles texture.
8. List of numbers: Speed factors as inputs for the gesture patches. Lists of input 6, 7 and 8 should have the same length.

OUTPUTS

1. List of chord-seqs.

This triggering curve...



...can be recognized as attack trigger for this score:

The image shows a multi-staff musical score. It consists of approximately 12 staves, each with a clef and a key signature. The notation includes various note values, rests, and dynamic markings. A vertical bar line is present, dividing the score into two systems. The score is presented in a standard musical notation style, with notes, stems, and beams clearly visible. The overall appearance is that of a professional musical score, possibly for a piano or a similar instrument.

ensemble-circles-lists makes it possible to use more than one gesture for each part.

Use bpf to create circling movements between different instruments in an ensemble.

INPUTS 1. Bpf (1000*1000): Trigger curve for arpeggio between multiple parts. The curve is divided into number of parts on the y-axis. A new note or event will start each time the curve crosses the border between parts. 2. Number: Overall duration seconds. 3. List of 2 numbers: Attack blur seconds minimum/maximum. (0 0) means no change. Numbers will be kept positive, and attacks will not extend past next attack. 4. List of 2 numbers: Duration blur seconds minimum/maximum. (0 0) means no change. Negative numbers will create staccato, positive numbers will create overlaps. However, the note will not extend past the next attack of a part, in order not to obstruct the curve shapes. 5. Number: Number of curve samples. 1000 could be a good compromise. With small numbers, the function will not detect all movements of the curve. 6. List of patches in lambda mode. These must have the following standard structure:
 INPUTS: 1. List of lists of midicents. 2. Number: Milliseconds duration. 3. Number: Speed ratio (1 is default speed, .5 is half speed).
 OUTPUT Chord-seq datalist. Check the prototype patches gen-simple-note, gen-simple-chord, sciarino-passage, ascending-thirds for clarification. As long as they have similar inputs and outputs, new patches can expand a repertoire of gestures. Patches making no, or just approximate, use of the duration input, will bring parts out of sync with the overall ensemble-circles shapes. In this version, the list of patches can be of any length. The script at input 7 will create gesture indexes for each part.
 7. Textfile self: The file will contain one list of index numbers for each part (0-x). 0 is the first gesture of input 6. The lists can be of any length. If there are too few gesture indexes, the rest will be randomly chosen from the list. If there are too many, the list will be cropped. If a part is missing a gesture list, the default will be (0). 8. Chord-seq self: Each note or chord will be the base of the input gestures. Number of notes or chords is the same as the number of parts in the ensemble-circles texture. 9. List of numbers: Speed factors as inputs for the gesture patches.

OUTPUTS 1. List of chord-seqs.

A script will for each part will select from the available gesture patches.

```

Te:
(0 0 1 0 0 0 1 1 0)
(0 0 0 0 1 1 1 0 1)
(0 0 1 1 0 0 1)
(0 0 0 1)
(0 0 0 1 1 0)
(1 0 0 1 1 0 0)
(0 0 0 1 0)
(0 0 0 0 0 1 1 1 1)
(0 0 0 0 0 1 1 0 0 0)
  
```

ensemble-circles-lists-gliss contains the gesture script, and an additional interpolation of the base notes.

Use bpf to create circling movements between different instruments in an ensemble.

INPUTS 1. Bpf (1000*1000): Trigger curve for arpeggio between multiple parts. The curve is divided into number of parts on the y-axis. A new note or event will start each time the curve crosses the border between parts. 2. Number: Overall duration seconds. 3. List of 2 numbers: Attack blur seconds minimum/maximum. (0 0) means no change. Numbers will be kept positive, and attacks will not extend past next attack. 4. List of 2 numbers: Duration blur seconds minimum/maximum. (0 0) means no change. Negative numbers will create staccato, positive numbers will create overlaps. However, the note will not extend past the next attack of a part, in order not to obstruct the curve shapes. 5. Number: Number of curve samples. 1000 could be a good compromise. With small numbers, the function will not detect all movements of the curve. 6. List of patches in lambda mode. These must have the following standard structure:
 INPUTS: 1. List of lists of midicents. 2. Number: Milliseconds duration. 3. Number: Speed ratio (1 is default speed, .5 is half speed).
 OUTPUT Chord-seq datalist. Check the prototype patches gen-simple-note, gen-simple-chord, sciarino-passage, ascending-thirds for clarification. As long as they have similar inputs and outputs, new patches can expand a repertoire of gestures. Patches making no, or just approximate use of the duration input, will bring parts out of sync with the overall ensemble-circles shapes. In this version, the list of patches can be of any length. The script at input 7 will create gesture indexes for each part.
 7. Textfile self: The file will contain one list of index numbers for each part (0-x). 0 is the first gesture of input 6. The lists can be of any length. If there are too few gesture indexes, the rest will be randomly chosen from the list. If there are too many, the list will be cropped. If a part is missing a gesture list, the default will be (0).
 8. List of numbers: Speed factors as inputs for the gesture patches. 9. Chord-seq self: Initial pitches. There should be one note or chord for each part. 10. Chord-seq self: Final pitches. This is the destination of the chord interpolation, and there should be as many final pitches as initial pitches. Order matters for glissando directions. 11. List of bpf (1000*1000): Chord interpolation shapes, one for each part. These shapes can help making the interpolations less linear and predictable. If there are too few bpf, last will be repeated. If there are too many, the list will be cropped. 12. List of numbers: Cents size of the bpf modulation. This decides the effect of bpf's from input 11. 13. Number: Cents intervalsize for the base chords with interpolation. Gesture patches can still add other interval sizes to the base chords.

Orchestral envelopes

The envelope functions in Ruben-OM do not generate material, but scale existing material. Orchestral sculptures are shaped by fitting input parts within envelopes of attack and decay. If the input is a chord with pitches sorted from high to low, the envelopes serve as lo-pass and hi-pass filters.

envelope-chord-demo

STABLE partials of a Tam-tam

reduce-multiseq-to-chord

sort-chord-notes

split-chord-to-multiseq

x-append

make an instance of the class multi-seq.

multiseq-env-poly

miditune

41

DESCRIPTION This function scales durations of each chord-seq to fit with an orchestral attack and decay line. This will work with a simple chord, or more elaborated textures. The quantification method inside is tree-quant.

INPUTS 1. List of chord-seq's. 2. Number: Tempo. 3. bpf-lib self with 2 bpf's: attack shape and decay shape (1000*1000). 4. List of 2 numbers: Percent attack and percent decay. This should not add up to more than 1. 5. List of lists: Time signatures. 6. Number: Seconds duration. 7. Number: Complexity factor (low is simplicity, high is precision, 10 is a compromise).

OUTPUTS 1. List of voices.

Musical fragments can be squeezed in between similar envelopes. This example shows the Ravel Sonatina, shifted and stretched with ascending envelope shapes.

envelope-music-demo

Ravel Sonatina

x-append

(0 1000) 8

(0 1000) 8

poly2multiseq

40 (.5 .5) ((3 8)) 34.45 9 100 100

multiseq-env-poly-legato-tie

41

miditune

multiseq-env-poly gives an attack and decay envelope to a musical texture.

This example makes a canonic distortion of Ravel. The music will be scaled to the duration available.

We will hear a canon where the original parts¹⁶ are out of sync.

¹⁶ I generally use the word 'parts' instead of 'voices' or instruments. The reason is possible confusion with the Open Music score objects: 'Chord-seq' is pure proportional notation, a 'voice' has quantification, 'poly' consists of a list of voices.

multi-env-poly-legato-tie joins a list textures through a list of envelopes. After the initial attack, the decays are the attack shapes of next texture, until a final decay. This can create fluid changes of textures within an orchestra section. The list of textures can be of any length.

env-multi-demo

multi-env-poly-legato-tie works on lists of musical textures, the transitions will be controlled by attack envelopes

multi-env-poly-legato-tie

INPUTS 1. List of multi-seq selfs. If number of chord-seqs within don't match, the smallest amount will be used. 2. Number: Tempo. 3. Bpf-lib (1000*1000) defining shape of textural attack and decay between each multi-seq. There must be 1 more bpf than there are multi-seqs, as the decay of last texture also needs to be defined. 4. List of numbers: Percent duration of each attack and decay (.1 is 1/10 of the overall duration). 5. List of numbers: Texture durations between attack and decay. The length of this list equals length of multiseq list. If the numbers don't add up to 1, overall duration will not be as defined at input 7. 6. List of time signatures. The quantification method within the patch is tree-quant. 7. Number: Seconds total duration. 8. Complexity factor (low number for simplicity, high number for precision, 10 is an average compromise). 9. Milliseconds rest threshold. 10. Milliseconds tie threshold.

OUTPUTS

1. List of voices.

DESCRIPTION

The input texture and transitions between textures, are rhythmically sculpted by a series of attack/decay-envelopes. This function is not limited to a single attack/decay pair, but can handle texture lists of any length. This is the most recent version, with the best control of each texture duration. The quantification method inside is tree-quant-legato-tie. DEMO PATCH env-multi-demo

Granulation

The principle of granulation of sound is extracting, reordering and superposing fragments of a sound. We can do a similar thing extracting random segments from Ravel's Sonata. This could be called 'musical granulation'. Through superposition of many extracts without synchronizing to common beats, a blurring of the music can be achieved. Used in a simpler way, *extract-onset-range-multiseq* can be used as an editing tool.

extract-onset-range-multiseq is useful for extracting parts of a piece while keeping the parts in sync. Every note before and after the onset range is deleted. With a chain of functions, this can be used to create collages from a piece.

Source piece: Ravel Sonata

extract-onset-fragment-multiseq is similar, except the extraction will start at original onset instead of being shifted to 0.

Randomized collages of extracts from the source piece

OM Loop - delay-them

Superposed fragments from Ravel's Sonata:

MULTI-SEQ 2

MULTI-SEQ

t: 5806 ms

Pitch approximation

A spectral analysis will contain precise intonations. Intervals can be approximated to a particular equal tempered interval size, which makes it easier to identify intervals in instrumental parts.

approximation-demo

↓

↑

A Tam-tam spectrum

Approximate a multi-seq to 1/8-tone, half tones, whole tones etc.

11
miditone

50

100

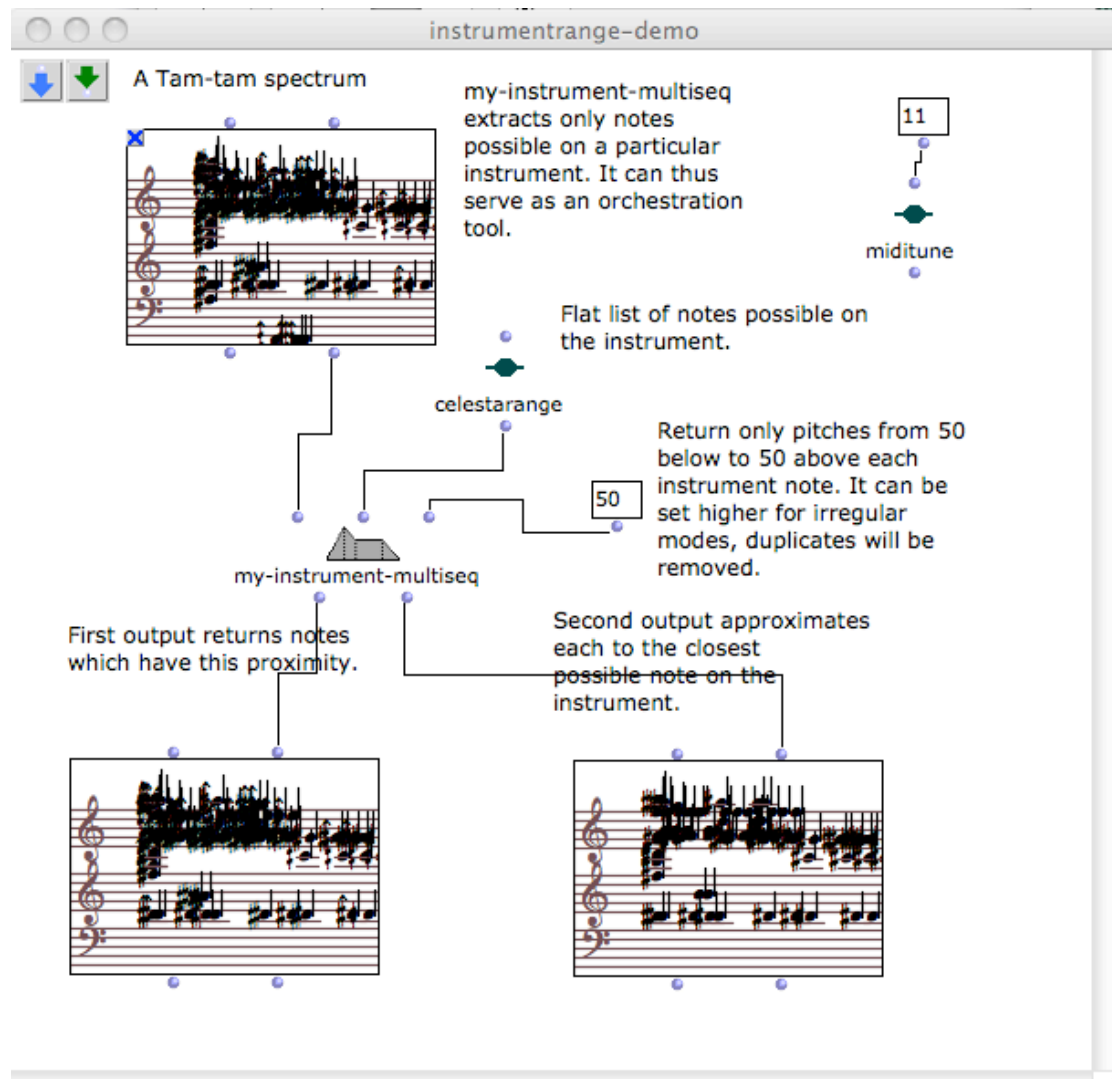
200

cent-approx-multiseq

cent-approx-multiseq

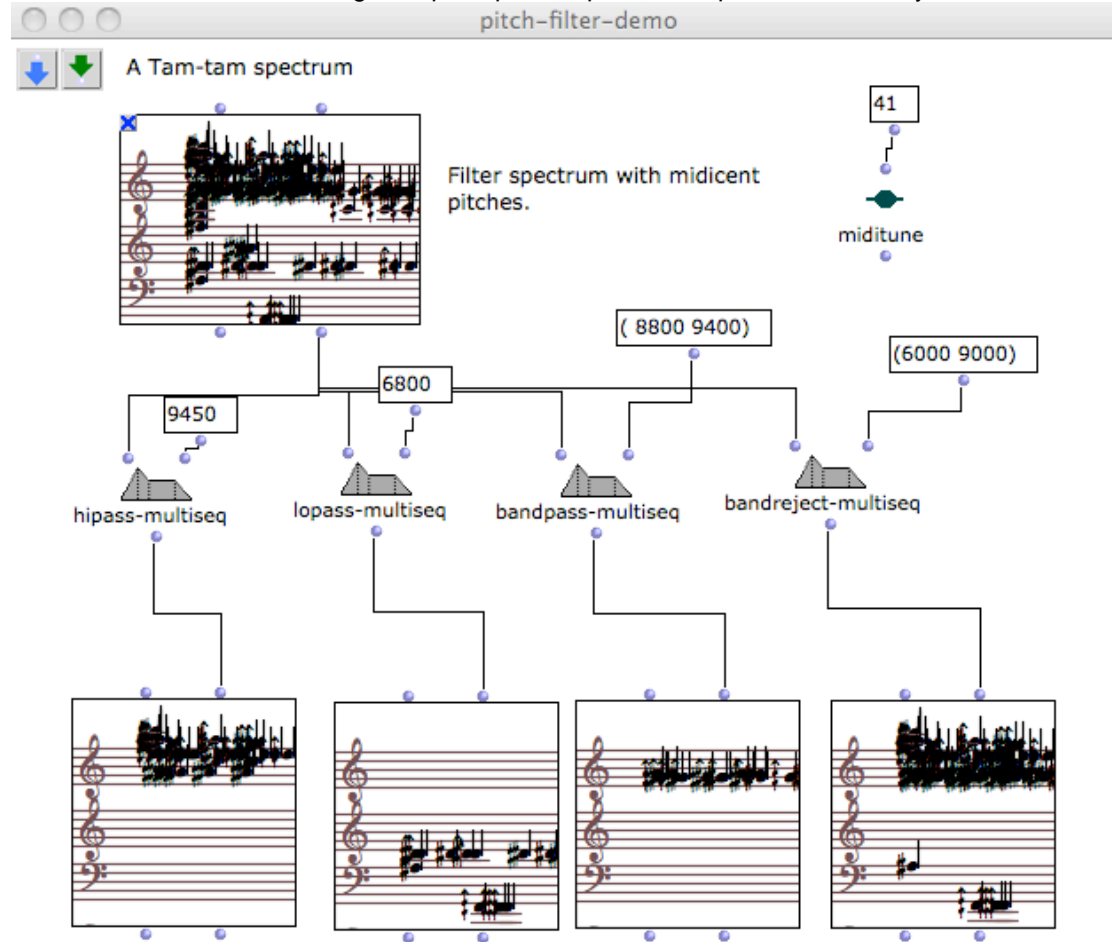
cent-approx-multiseq

Another possibility is extracting only notes within a certain proximity to notes of an instrument. Lists of possible midicent notes on the instruments can make this a useful orchestration tool.

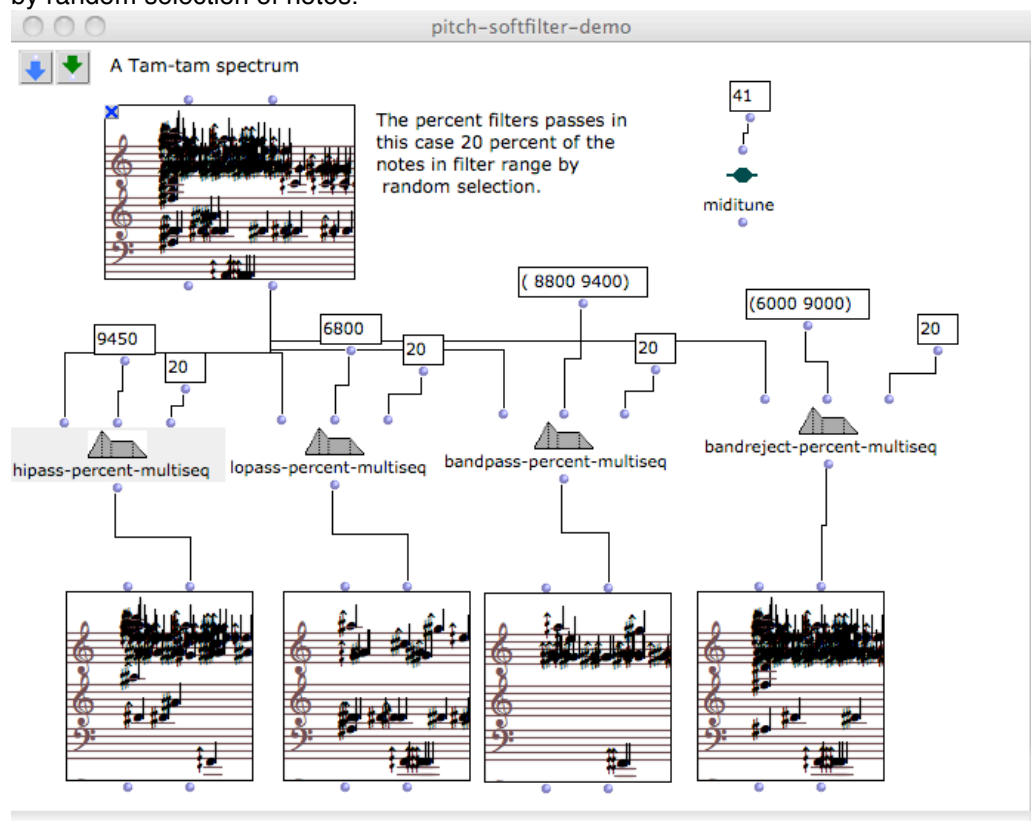


Pitch filters

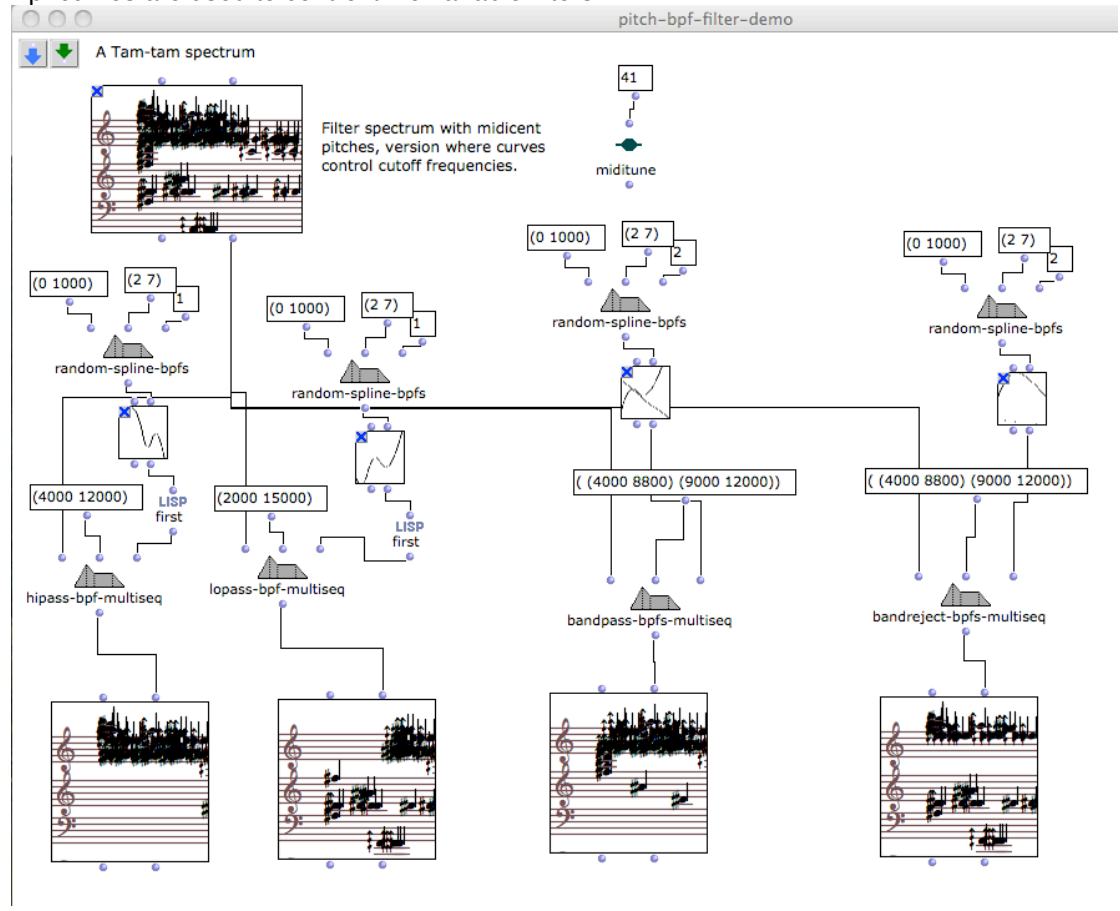
Notes can be extracted through simple hipass, lopass, bandpass and bandreject filters.



The soft filters do not remove everything (unless filter pass percent is 0), but thins out regions by random selection of notes.



Bpf curves are used to control time variable filters.



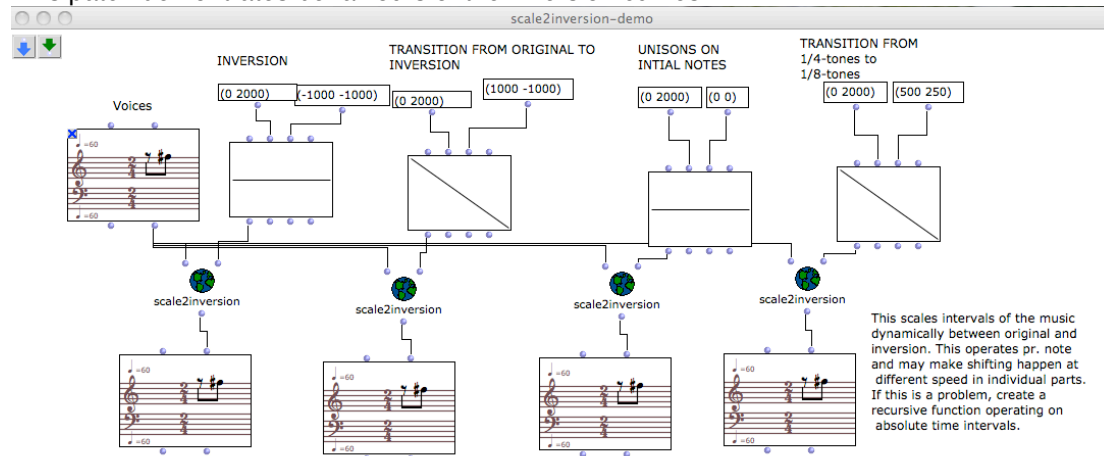
Interval inversion curves

Modus Quaternion offers 4 versions of a material:

- Original.
- Retrograde (time curve from end to beginning).
- Inversion (intervals are multiplied by -1).
- Retrograde inversion.

Through a combination of inversion functions and time pointers, all four versions are available with Ruben-OM. Controlling inversion degree and time pointers, over time by curve shapes, offers not just four, but endless variations of a material. At some point, the original identity of the material may be obscured.

This patch demonstrates behaviours of the inversion curves.



The original fragment from Ravel's Sonata.



A pure inversion of the same fragment.



A transition from original to inversion. This extract is short, the whole process can be studied from the patch. Intervals reach microtonal in-between states, and the fragment ends with inverted intervals.

A musical score in 2/4 time with a tempo of quarter note = 60. It consists of five systems of staves. The first system shows a melodic line in the upper voice and a rhythmic accompaniment in the lower voice. The second system continues the melodic line with more complex rhythmic patterns. The third system shows the melodic line becoming more fragmented and the accompaniment becoming more active. The fourth and fifth systems show the melodic line further developing and the accompaniment becoming more complex.

Original is interval multiplied by 1, inversion is multiplication by -1. A transition can go through 0. Intervals multiplied by 0 will give only note repetitions.

A musical score in 2/4 time with a tempo of quarter note = 60. It consists of five systems of staves. The first system shows a melodic line in the upper voice and a rhythmic accompaniment in the lower voice. The second system continues the melodic line with more complex rhythmic patterns. The third system shows the melodic line becoming more fragmented and the accompaniment becoming more active. The fourth and fifth systems show the melodic line further developing and the accompaniment becoming more complex.

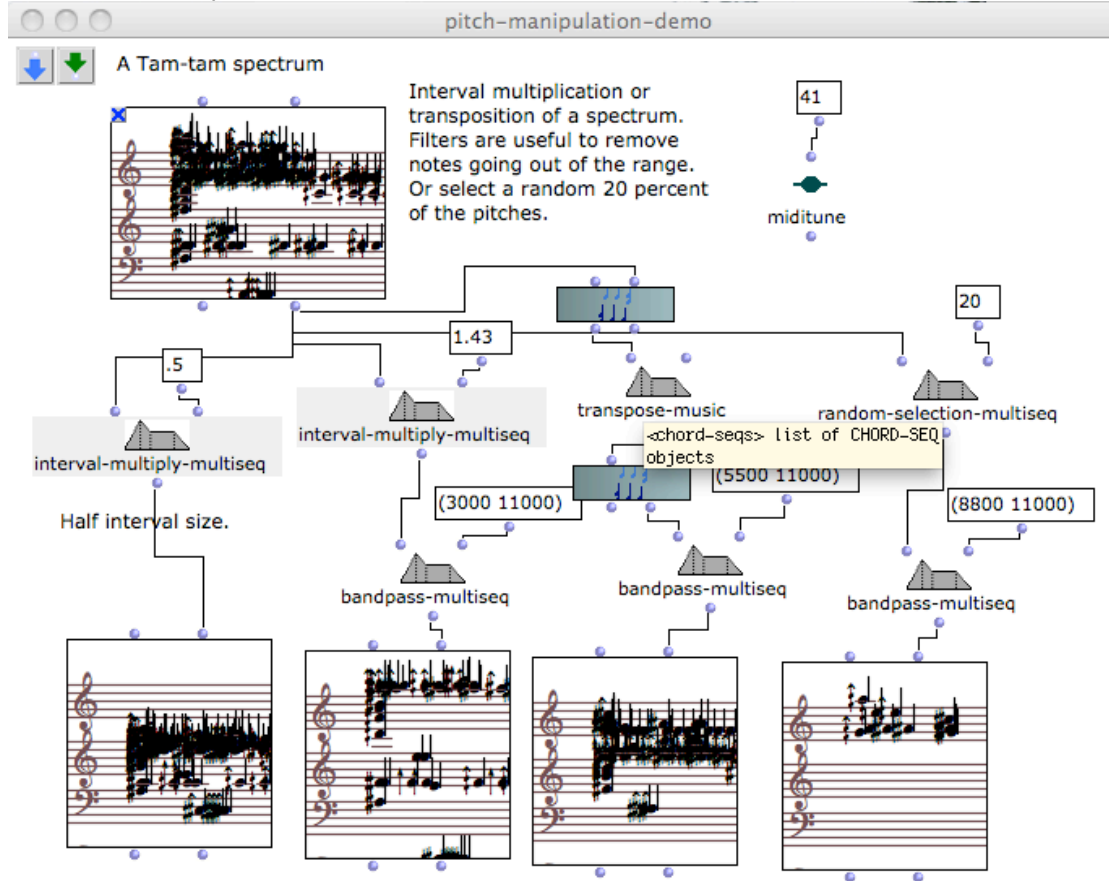
Half-tone materials can be transformed to quarter-tones by diminishing the interval sizes. This version gradually transforms from quarter-tones to eight-tones.

A musical score in 2/4 time with a tempo of quarter note = 60. It consists of five systems of staves. The first system shows a melodic line in the upper voice and a rhythmic accompaniment in the lower voice. The second system continues the melodic line with more complex rhythmic patterns. The third system shows the melodic line becoming more fragmented and the accompaniment becoming more active. The fourth and fifth systems show the melodic line further developing and the accompaniment becoming more complex.

Pitch multiplication

There are 2 possible 'pitch multiplications'.

First literally multiplying intervals by a single floating number, for new microtonal scalings. A tam-tam spectrum loses some of the original sonority when multiplied by a floating number, as the stretched spectrum becomes more artificial.



A second possibility is what Pierre Boulez defined as 'chord multiplication'. This is pure transposition, but number of notes gets multiplied.

harmonize-multiseq

INPUTS 1. List of chord-seqs. 2. List of numbers: Cent transpositions (positive or negative) from each note of the input music.

OUTPUTS 1. List of chord-seqs.

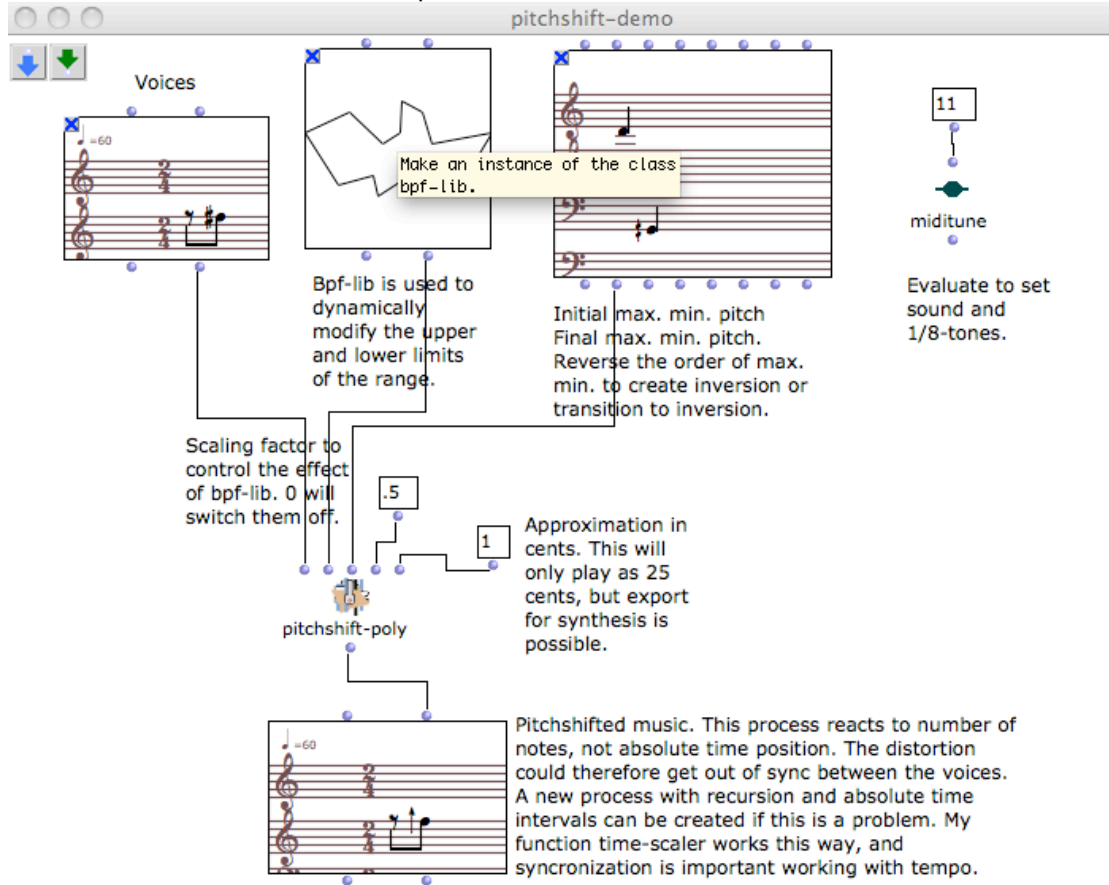
DESCRIPTION
The interval lists will be added as a chord to every note
This works like an organ mixture, or a static version of the Boulez 'chord multiplication'. If the second input was a list of lists (or another list of chords-seqs), with some sort of indexing matrix for chords to 'multiply4' with each other, the patch could be expanded to work on the techniques used by Boulez in parts of *Le Marteau sans maitre*. The patch inside could be expanded for this purpose.

interval-multiply-multiseq

If we were to follow up the use of chord multiplication in Boulez's *Le Marteau sans maitre*, this patch would need to be developed further. The piece involves not just one chord to 'chord multiply' with the others, but intricate matrixes of combinations.

Pitch shifting

Through pitch-shifting, the overall range of the music can be dynamically shaped over time. If the overall pitch shifting range is similar to the input, a transposition will happen. Otherwise the music will be twisted out of shape.



Ravels Sonatina is used as an example.



This is a microtonally distorted Ravel. The melodic gestures can still be recognized.



A pitch shifted Ravel fragment also scrambled with a time pointer would be harder to identify, while it would contain imprints of the general activity of the music.

Glissando

We can create a glissando from one chord to another, to hear harmony of the transitions, or for csound synthesis.

gliss-chords

gliss-chords

INPUTS 1. Chord self: Initial pitches. 2. Chord self: Finale pitches. Order matters for glissando directions. 3. List of bpf(1000*1000): Curve modulation of the pitch interpolation. 4. List of numbers: Cents range of the bpf modulation. 5. List of numbers: Number of notes in each part. 6. Number: Seconds duration.

OUTPUTS 1. List of chord-seqs.

DESCRIPTION gliss-chords create a glissando from one chord to another. Bpf-shapes allow the transitions to happen in other shapes than straight lines. The function makes it possible to listen to harmonic situations during this transition, even though a glissando notation for a group of strings would only contain the turning points.

A trill glissando will require four chords, when additional trill notes at beginning and end are included.

tr-gliss-chords

sort-chord-notes

(0 1000) (-1000 1000)

repeat-n

15 8

50 -600 -750 INPUT

transpose-music transpose-music transpose-music

(345 467 120 457 50 120 25 75 345 278 45 76 83 250 123 54 345 76)

(39 41 29 35 47 25 19 51 50 38 28 25 24 36 21 18 17 27)

trill-chords 19.45

miditune 11

trill-chords

INPUTS

1. Chord 2. Chord 3. Chord 4. Chord
5. List of few, last 6. List of few, last 7. List of positions are for trill notes. 8. List of numbers: Number of notes for each part. 9. Number: Seconds duration.
- self: Initial main chord. self: Initial trill chord. self: Final main chord. self: Final trill chord. bpf (1000*1000): Curve added to glissando shape of main chords. If there are too many, the list will be repeated. If there are too many, the list will be cropped. bpf (1000*1000): Curve added to glissando shape of trill chords. If there are too many, the list will be repeated. If there are too many, the list will be cropped.
- numbers: Cents range of the bpf curves. Even positions are for main notes, odd

OUTPUTS

1. List of chord-seqs.

DESCRIPTION

trill-chords creates a texture of trills

The trill glissando score.

The trill glissando is combined with addition of rests. Generally ascending bpf's increase probability and duration of rests over time. Finally the rhythms are quantized.

add-rests-multiseq

INPUTS

- List of chord-seqs.
- List of bpf's(1000*1000): The bpf's show trajectories for percentage of rests through the input music. If there are too few bpf's for the number of chord-seqs, last will be repeated. If there are too many, the list will be cropped.
- List of 2 numbers: Maximum and minimum percentage of rests.
- List of bpf's(1000*1000): The bpf's show trajectories for length of rests through the input music. If there are too few bpf's for the number of chord-seqs, last will be repeated. If there are too many, the list will be cropped.
- List of 2 numbers: Maximum and minimum milliseconds length of rests.

OUTPUTS

- List of chord-seqs.

DESCRIPTION Pierre Boulez has called inserted rests "pockets of silence". `add-rests-multiseq` inserts random silences based of probability curves for how often rests are inserted, and how long they are. As the function has random operations, the result will be different at each evaluation.

A trill glissando with progressively more rests. The whole score can be seen and heard in the patch.

POLY

Selection: CHORD

Duration: 39000 ms

Different processes are combined to create a larger orchestral texture. Notes of a tam-tam spectrum serve as beginning and end of a glissando. This could be a static sound, but the bpf's create a constant glissando in all parts. The texture is thinned out by rests, and finally scaled to a single orchestral attack/decay envelope.

hollowed-iterated-texture

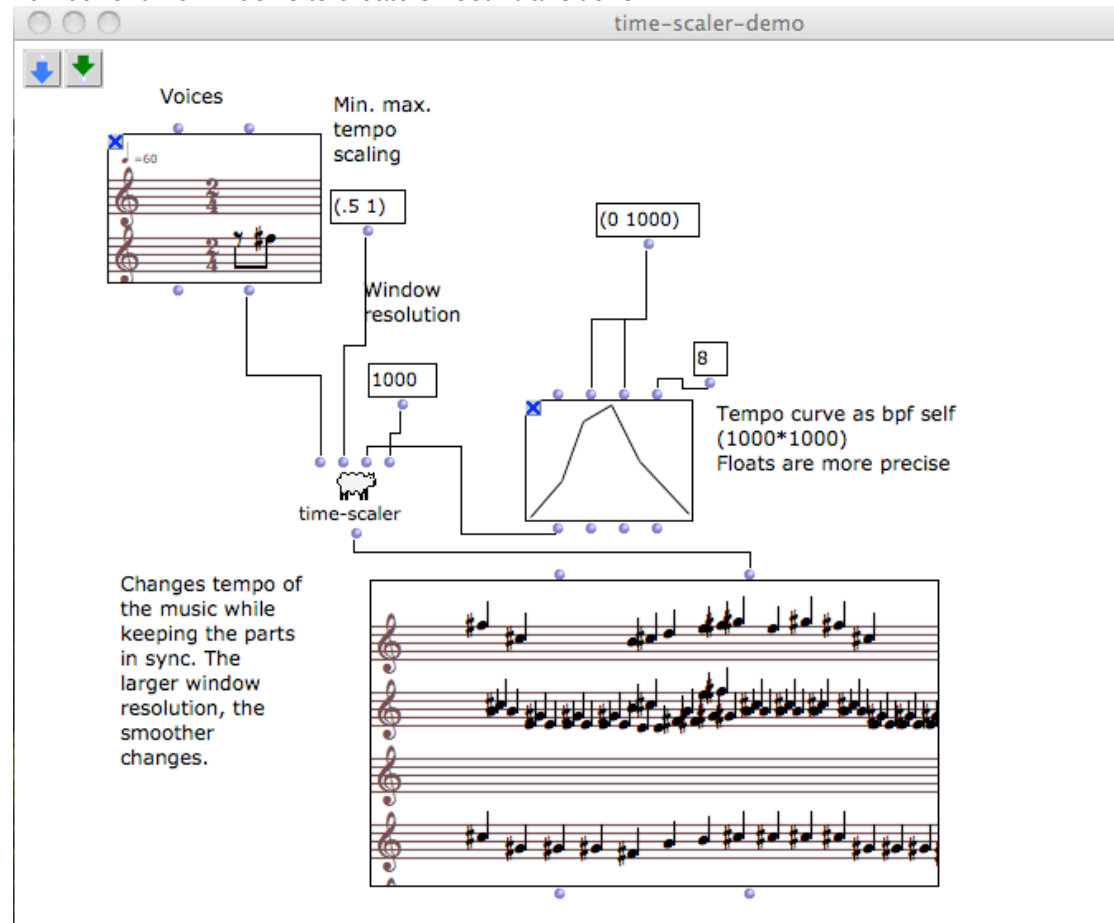
This patch combines a number of functions to create a broken iterated texture. The steps are:

1. Extract portions of a tam-tam lasting more than 50 ms
2. Make a chord and sort it.
3. Make a gliss with equal start and end, but with graphical inflections of a quarter-tone.
4. Add percents of rests after randomized graphs.
5. Quantify to a poly object with pyramid shaped envelopes.

POLY

Time scaling

A human performance would have constant irregularities and tempo changes. A MIDI performance can get more flexible through curves of tempo. The music is split into a large number of time windows to create smooth transitions.



Micro polyphonies in works by Ligeti are dense textures made through numerous simultaneous rhythmic versions of the same melodic line, like a cluster of delays. We can attempt to thicken a line through time scaling with a number of randomized curve shapes. This does not show Ligeti techniques, but a related textural approach.

timescale-ligeti-canon

This is test of one principle use in several orchestra works by Ligeti:
A single melody is time stretched into a dense texture. I'm not using exact techniques from Ligeti, but individual bpfs to create a different phasing of the rhythm for every string player.

Multiple melodies can start at the same time, every string player in five sections has a separate time scaling curve. This creates a large score, which can be heard in the patch, and exported to Finale for editing.

t5part-ligeti-canon

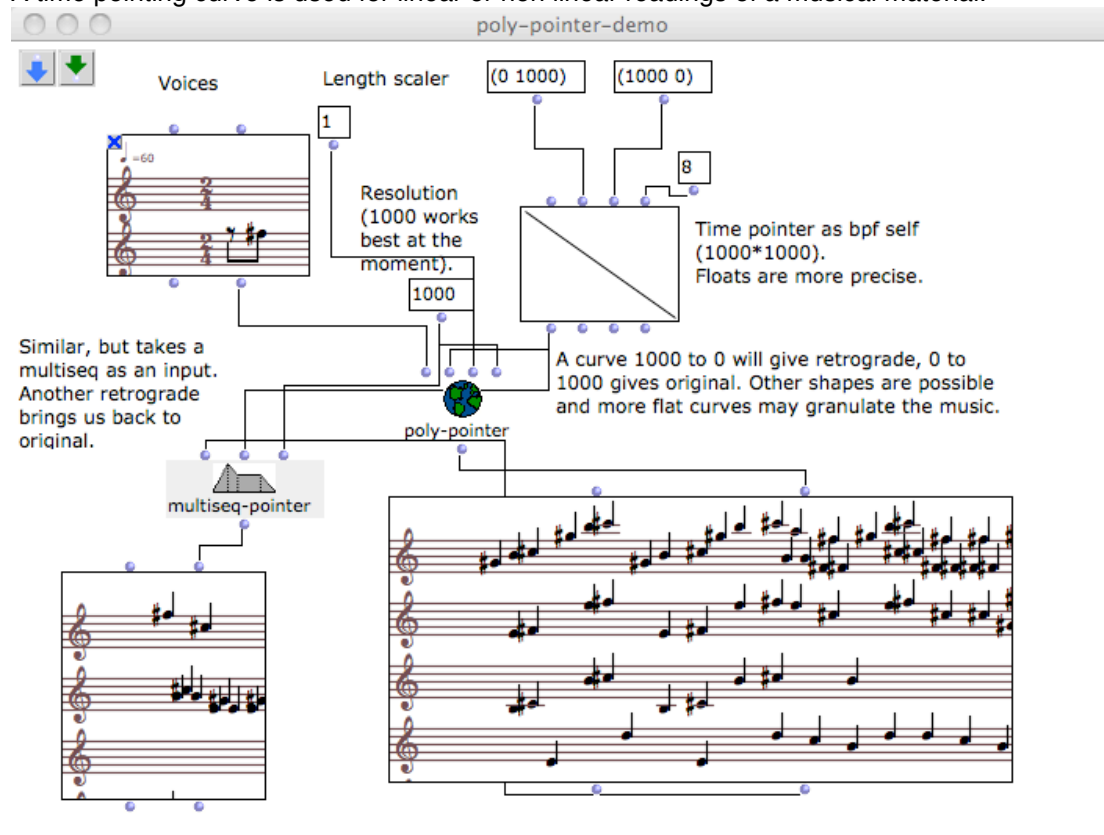
For string quintet

For large orchestra

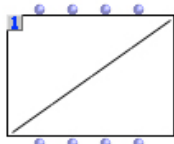
save-as-ef

Time pointer

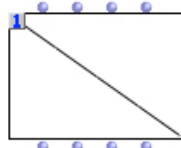
A time pointing curve is used for linear or non linear readings of a musical material.



A curve where y moves 0-1000 will return the original order:



A curve with y moving 1000-0 returns a retrograde curve:

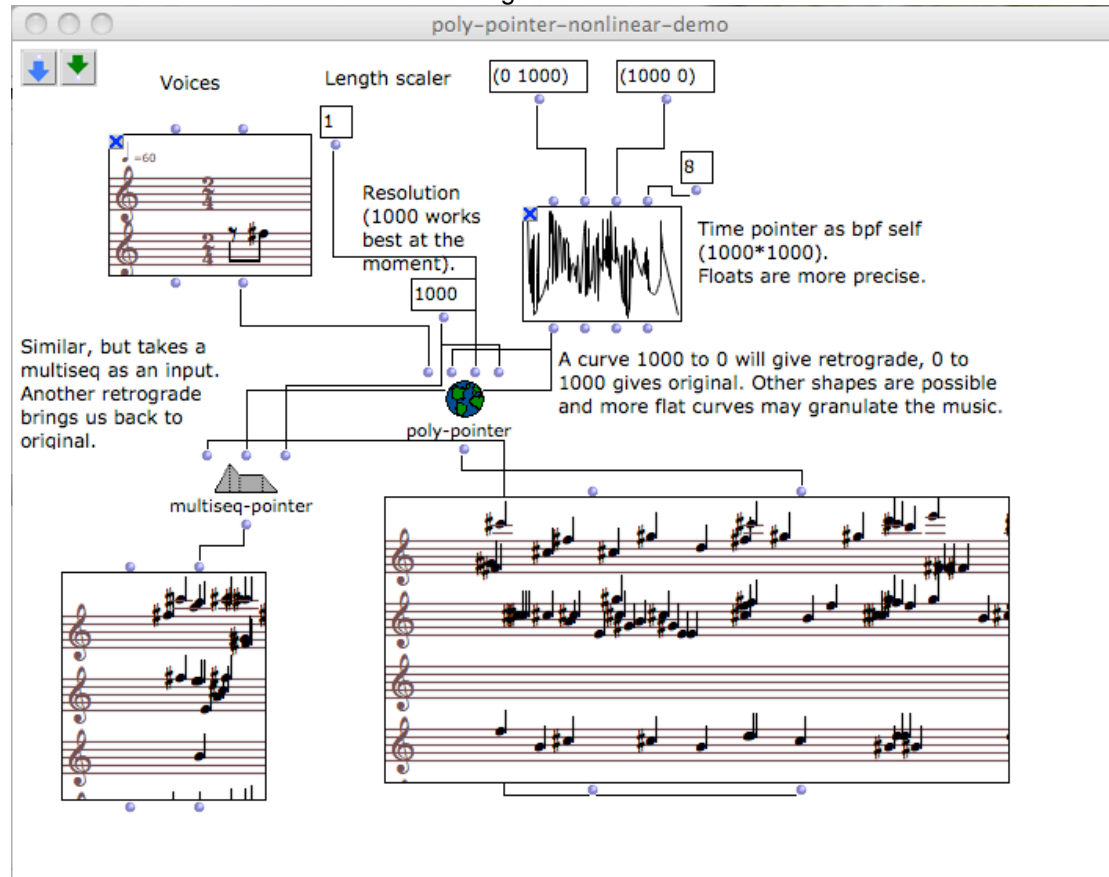


This is the resulting retrograde of the Ravel Sonata:

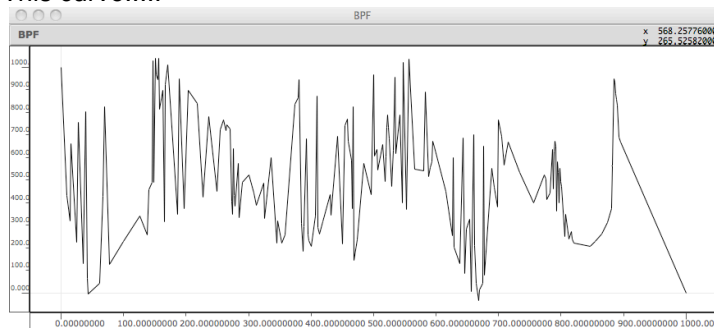
t: 7646 ms

Duration: 24227 ms

Other curves can create non linear readings of the music.



This curve.....



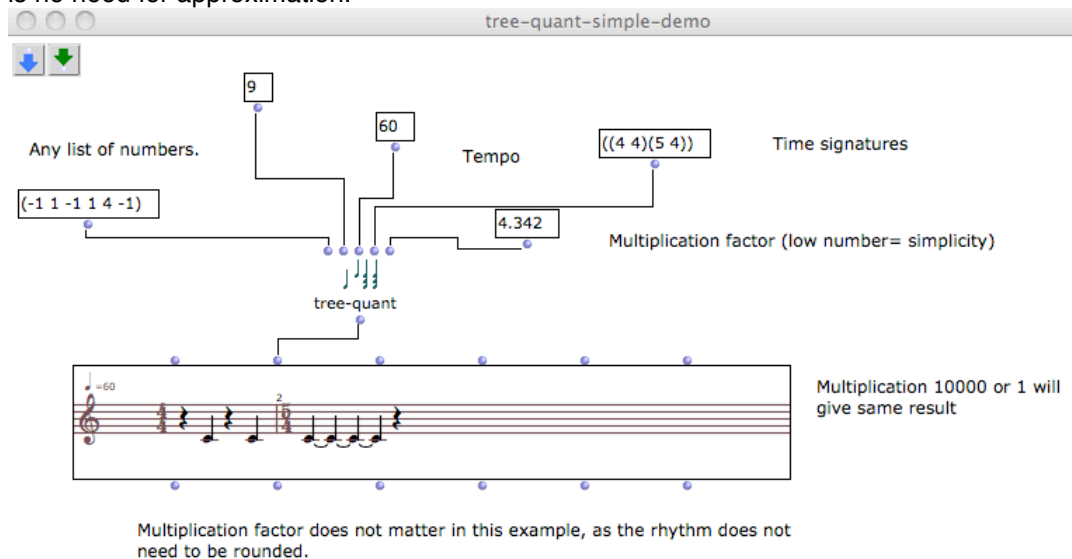
... will create a scrambled version of Ravel's Sonata. The pitches are all from the original, while irregular scannings through the music are visible. At points Ravel is rapidly read from the beginning to the end, multiple times, while calmer curves can make the music stop, or dwell on shorter fragments.

Rhythm quantification

Open Music uses rhythm trees as a format for time signatures, measures and proportions. To quantify absolute durations, or irrational lists of numbers, to a tempo and rhythm is complicated. The standard function *omquantify* tends to lose a lot information. *mktree* is more accurate, but it's hard to control the the complexity of the quantifications. I needed a method handling any list of irrational proportions well.

The *tree-quant* functions are new alternatives included with Ruben-OM. A list of number proportions are converted to a rhythm tree without loss of notes. Changing the overall duration makes it easy to augment or diminish rhythms by any ratio.

The first example is simple. Proportions add up to 9, duration is 9 seconds at tempo 60. There is no need for approximation.



But if we for instance changed tempo to 59, or duration to 9.238724, we would see how different multiplications affect the complexity and precision of the approximation.

A hierarchy of lists give measures with time signatures and proportions. Floats like 1.0 means the the note is tied from previous beat, negatives like -1 means that it is a rest. It is possible to add further levels of lists within lists to create subdivision within subdivisions. I have chosen not to go below the level of beat subdivision, but this could be a possible direction to further develop the quantifier.

A list of irregular proportions makes it possible to experiment with different multiplication factors. There are tiny values which could risk bringing calculations outside the numerical range and cause errors. These are increased and "saved" by the *tree-quant*.

tree-quant-complex-demo

Any list of numbers. Will be scaled to duration, very small numbers will be rounded up.

11.65 40 Tempo ((3 4)(2 8)(3 4)) Time signatures

(-2 1.43 -4 9.342 -2 4 .01 -.01 .002 -7 37)

4.323 Multiplication factor (low number= simplicity)
Int or float

tree-quant

Multiplication 4.323

VOICE

Multiplication 20

VOICE 4

Multiplication 5

VOICE 3

Multiplication 1

VOICE 2

Lets show these examples as rhythm trees.

With multiplication 1, proportions are simplified to 1 in most cases:

$(5/2 (((3\ 4) ((1\ (-1\ 1\ -1\ 1)) (1\ (3.0\ -1)) (1\ (-1\ 1\ 1\ -1\ 1\ -1)))) ((2\ 8) ((1\ (-2\ 1)) (1\ (1.0)))) ((3\ 4) ((1\ (1.0)) (1\ (1.0)) (1\ (1.0)))) ((3\ 4) ((1\ (3.0\ -1))\ -2))))$

With multiplication 4.323, proportions vary more. The structure of the list is the same, but contrasts between proportions is what makes it increasingly complex in notation:

$(5/2 (((3\ 4) ((1\ (-2\ 2\ -4\ 1)) (1\ (8.0\ -1)) (1\ (-2\ 4\ 1\ -1\ 1\ -1)))) ((2\ 8) ((1\ (-5\ 1)) (1\ (1.0)))) ((3\ 4) ((1\ (1.0)) (1\ (1.0)) (1\ (1.0)))) ((3\ 4) ((1\ (7.0\ -2))\ -2))))$

With multiplication 5:

$(5/2 (((3\ 4) ((1\ (-2\ 2\ -4\ 2)) (1\ (9.0\ -1)) (1\ (-2\ 4\ 1\ -1\ 1\ -1)))) ((2\ 8) ((1\ (-5\ 1)) (1\ (1.0)))) ((3\ 4) ((1\ (1.0)) (1\ (1.0)) (1\ (1.0)))) ((3\ 4) ((1\ (8.0\ -2))\ -2))))$

With multiplication 20:

$(5/2 (((3\ 4) ((1\ (-7\ 5\ -14\ 5)) (1\ (31.0\ -2)) (1\ (-6\ 14\ 1\ -1\ 1\ -1)))) ((2\ 8) ((1\ (-16\ 2)) (1\ (1.0)))) ((3\ 4) ((1\ (1.0)) (1\ (1.0)) (1\ (1.0)))) ((3\ 4) ((1\ (31.0\ -8))\ -2))))$

Since rhythm trees build notation from proportions to notation, Open Music is capable of notating results far beyond what is musically practical. This is a quantification with multiplication 3000:

These rhythms arise through the more precise approximation of the irrational values coming in:

$(5/2 (((3\ 4) ((1\ (-1047\ 749\ -2094\ 613)) (1\ (4501.0\ -225)) (1\ (-823\ 2094\ 3\ -6\ 6\ -3)))) ((2\ 8) ((1\ (-2251\ 161)) (1\ (1.0)))) ((3\ 4) ((1\ (1.0)) (1\ (1.0)) (1\ (1.0)))) ((3\ 4) ((1\ (4501.0\ -1051))\ -2))))$

The sum of numbers create the subdivision of each beat.

Through the experiences of tree-quant, I started to see the need to limit short rests. *tree-quant-legato* makes it possible to filter out short or unintended rests.

tree-quant-legato-demo

Proportions: (-2 1.43 -4 9.342 -2 4 .01 -.01 .002 -7 37)

Time signatures: ((3 4)(2 8)(3 4))

Tempo: 40

Multiplication factor (low number = simplicity): 10

Int or float: 1000

Rest threshold: 1000

tree-quant-legato

This is a version of my quantifier where you have the option to remove rests shorter than the threshold value. This should be close to a milliseconds limit, but you have to set it higher the higher the multiplication factor is.

This is useful when quantifying a chord-seq. Chord-seq takes onsets and duration only as ints. The result will often be a small unintended rest in between as a result of the approximation.

My quantifier is programmed to also keep very small values, which makes these approximations too complicated. It is common practise not to notate the rests between notes during rapid staccato passages. While material made for synthesis of midi performance should be as precise as possible.

VOICE

Rest threshold 1000

VOICE 2

Rest threshold 500

VOICE 3

Rest threshold 100

The preceding functions are strict; if a note doesn't start exactly on the beat, there will be a tie. *tree-quant-legato-tie* introduces a duration threshold to tie a note to next beat.

tree-quant-legato-tie-demo

This version of the quantifier adds control over tie approximation. If tie threshold is 0, it will be very strict, and always tie when proportions don't coincide with the beat. In some cases you would want smoother rhythms.

VOICE

VOICE 2

Numeric proportions, or macro rhythms, can easily be quantified.

macro-rhythm-demo

This is a method of creating poly-rhythms and floating augmentations from a few simple proportions. Each rhythm may start at a point within the series itself, thus creating subsets.

I have used this to make time structures for many pieces since 2003. However, these rhythms has not been used directly, but as triggering impulses I choose to use or ignore.

VOICE

Quantification can be a final step of transferring a spectrum to notation. *multiseq2poly-legato-tie* is a high level function using *tree-quant-legato-tie*. Through hockets a tam-tam spectrum is split by 1/16-tones. This will give each musician the chance to perform halftones at different tunings. Zero durations and overlaps are changed before quantification, and every part is time scaled down to a playable speed. Converting this spectrum to a score combines hockets, time scaling and quantification.

quantify-hockets-demo

A Tam-tam spectrum

Combination of processes demonstrated elsewhere to create a spectral polyphony for retuned orchestra groups. There are many other possible separation criteria, which can reduce the need to shorten all the long partials.

hocket-multiseq 12.5

hocket-and-quantify flat

miditune 41

Make an instance of the class poly.

OM Loop - hocket-and-quantify

chordseqs

listloop

LISP list

adjust-onset-dur 10

multiseq-extract-param

rebuild-multiseq

time-scaler 1000

multiseq2poly-legato-tie

collect

eachTime

finally

1000

1000

40

3.5452

0

20

(5 7)

((3 4))

(0 1000)

(999 999)

An approximation of only 1/8-tones will increase the amount of activity within each part, as notes are split between fewer parts.

quantify-hockets-multiple-demo

41
miditune

25
hocket-multiseq

hocket-and-quantify

flat

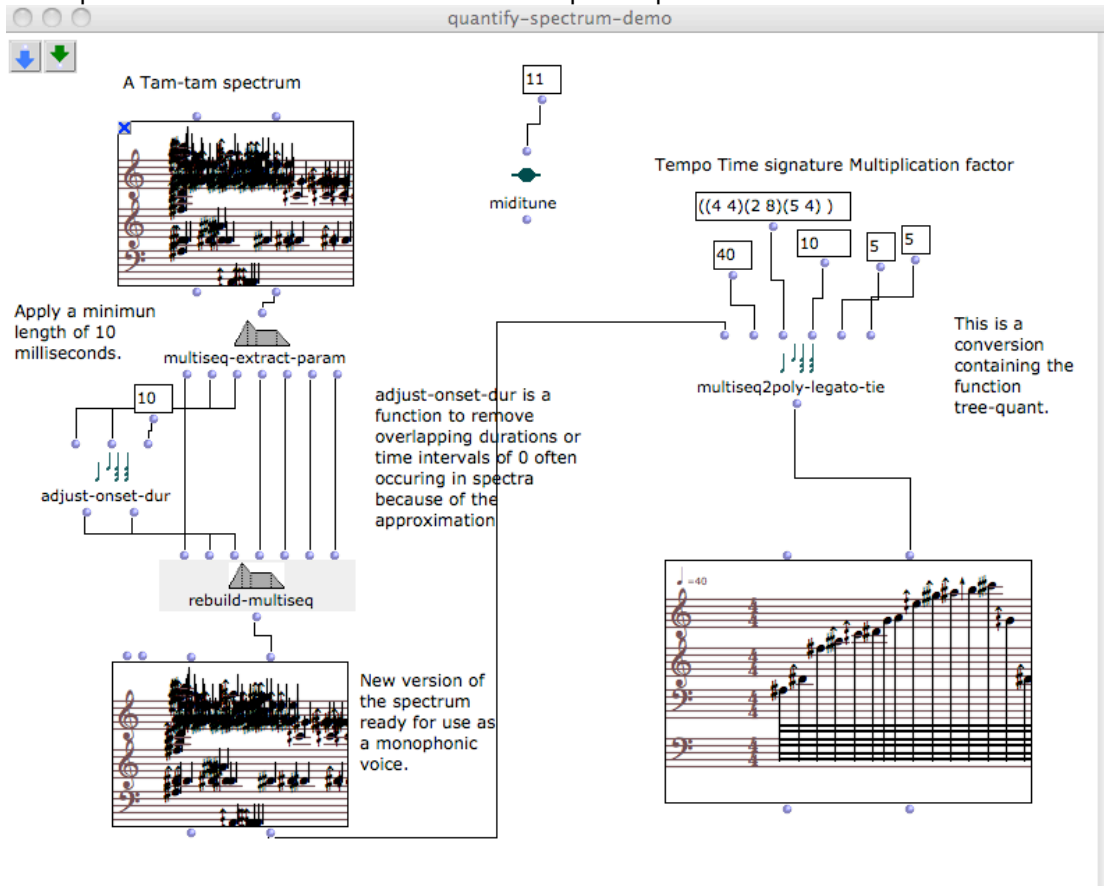
quarter = 40

quarter = 40

quarter = 40

quarter = 40

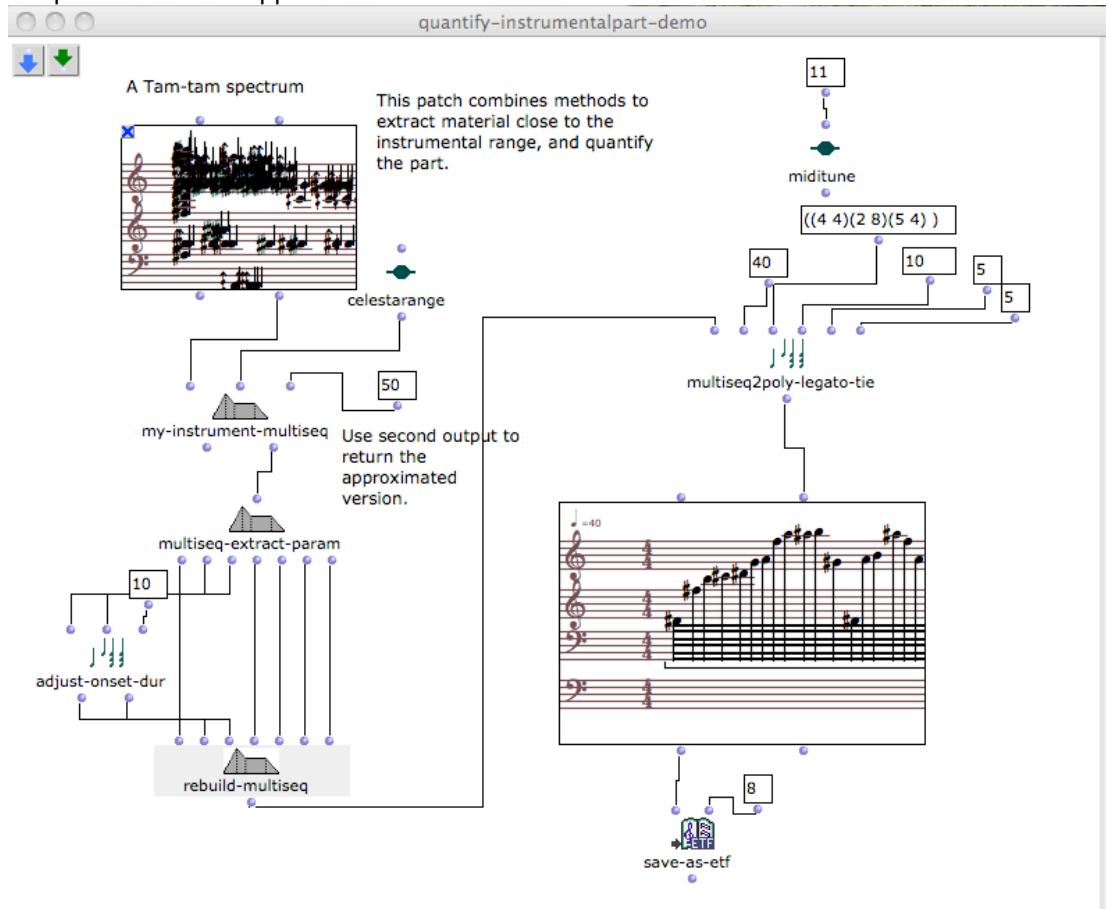
The spectrum can also be converted to a monophonic part.



The tam-tam spectrum melody.



A spectrum can be approximated to available notes on an instrument.



The notes exist on a celesta, but the part would need time scaling to be playable. That would not complete a piece by itself. Further technical challenges on the instruments should be considered, and musical choices can be made.



LITERATURE

[Multiple authors]. The Om Composer's Book.1, ed. Carlos Agon/ Jean Bresson/ Gérard Assayag, Paris: Editions Delatour France/ Ircam-Centre Pompidou, 2006.

— — — . The Om Composer's Book.2, ed. Jean Bresson/ Carlos Agon/ Gérard Assayag, Paris: Editions Delatour France/ Ircam-Centre Pompidou, 2008.

Harley, Maria Anna. "Space and Spatialization in Contemporary Music: History and Analysis, Ideas and Implementations." McGill University, School of Music, 1994.

Boulez, Pierre. Boulez on Music Today, trans. Susan Bradshaw and Richard Rodney Bennett. London: Faber and Faber, 1971.

Contemporary Compositional Techniques and Openmusic, ed. Bob Gilmore Rozalie Hirs, Paris: Editions Delatour/Ircam-Centre Pompidou, 2009.