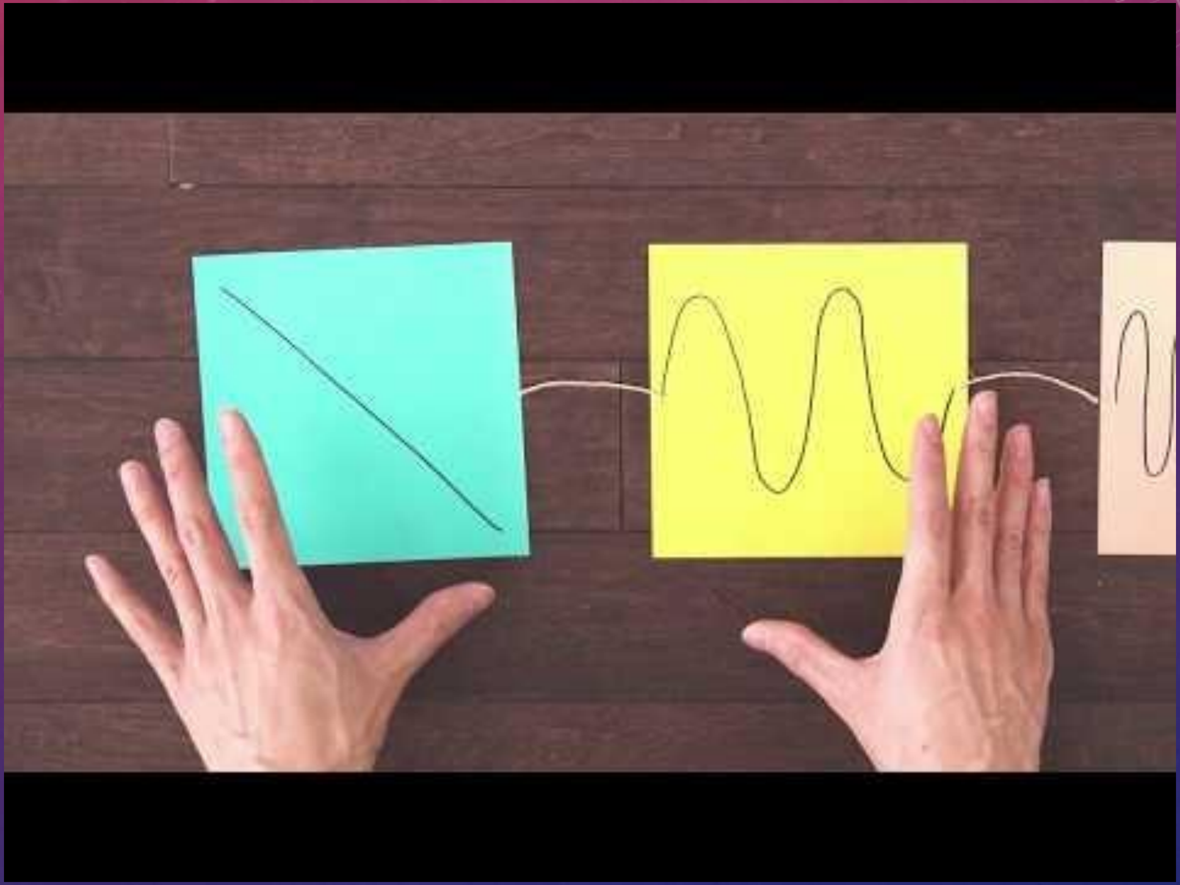
fmOP

A 6-OPERATOR PURR DATA FM SYNTHESIZER

By Frash Pikass

FM synthesis in a nutshell



Video: https://www.youtube.com/watch?v=vvBl3YUBUyY

FM synthesis, basic concepts (1/2)

Main frequency (pitch)

- The main frequency that the synthesizer must play (*e.g.: A440 on a keyboard*)

Operator

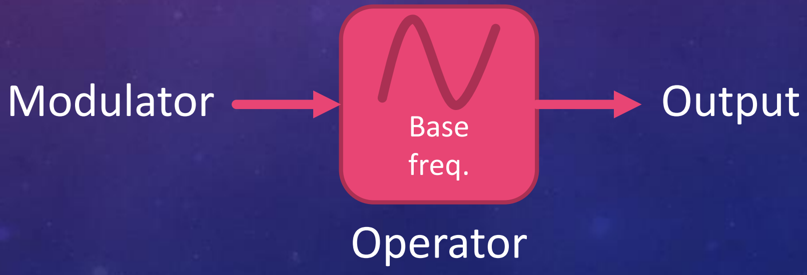
- An oscillator producing a sine wave that can be modulated in frequency by an input signal:

$$operator(t) = \sin(f_{base} + modulator(t))$$

- **Base (carrier) frequency**: a multiple of the main frequency

$$f_{base} = (f_{main} + pitchBend) \cdot ratio$$

- **Modulator**: an operator whose output modulates another operator
- **Carrier**: an operator whose output is being modulated (usually it’s what you can hear in a synth)



Modulator

Base freq.

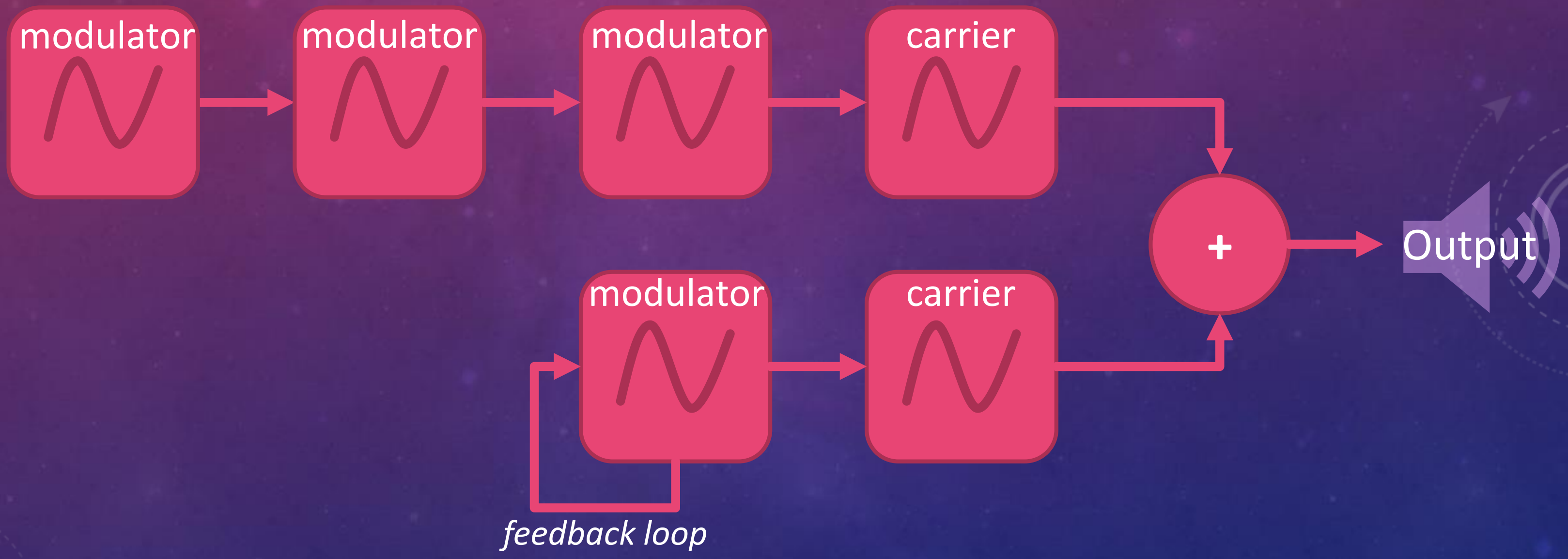
Output

Operator

FM synthesis, basic concepts (2/2)

Algorithm

- A specific configuration of connected operators
- Example (*Yamaha DX7 algorithm 2*):



The Yamaha DX7 (1983)

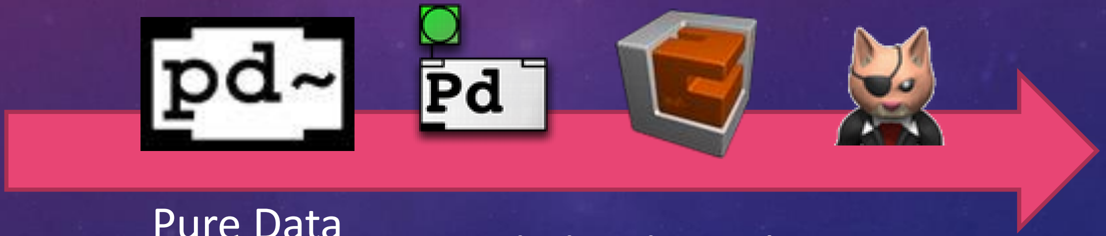
- **The most famous exponent of FM synthesis**
- Produced some of the **most iconic sounds** from the ‘80s (like the legendary E.PIANO2)
- **6 FM operators** with
  - **FM modulation:** every operator can modulate other operators or itself (**feedback**)
  - The innovative **LR4 Envelope Generator**, more flexible than ADSR
  - **Key velocity** sensor, **pitch bending, mod wheel, MIDI**
  - Plenty of customizable parameters
- **32 algorithms** → operator configurations chosen by Yamaha
- **16 voices** → up to 16 notes can play at the same time

> “
>
> let’s make this in a very modular way,
> it won’t take long!
>
> ”

Myself, before a 3 months Pure Data code-a-thon

Why Purr Data?

- Pure Data : a visual programming language for **interactive computer music**
- In the years, it has spawned a few **forks (flavours):**
  - Pd (vanilla): the original, created in the ‘90s by **Miller Puckette**,
  - Pd-extended offered powerful libraries → the codebase is now **abandoned**
  - Pd-L2Ork: forked from Pd-Extended by the *Linux Laptop Orchestra* research group of **Ivica Ico Bukvic** at Virginia Tech → well maintained, has many useful libs
  - Purr Data the latest version of Pd-L2Ork → great *WebKit* GUI, best usability, resource-hungry



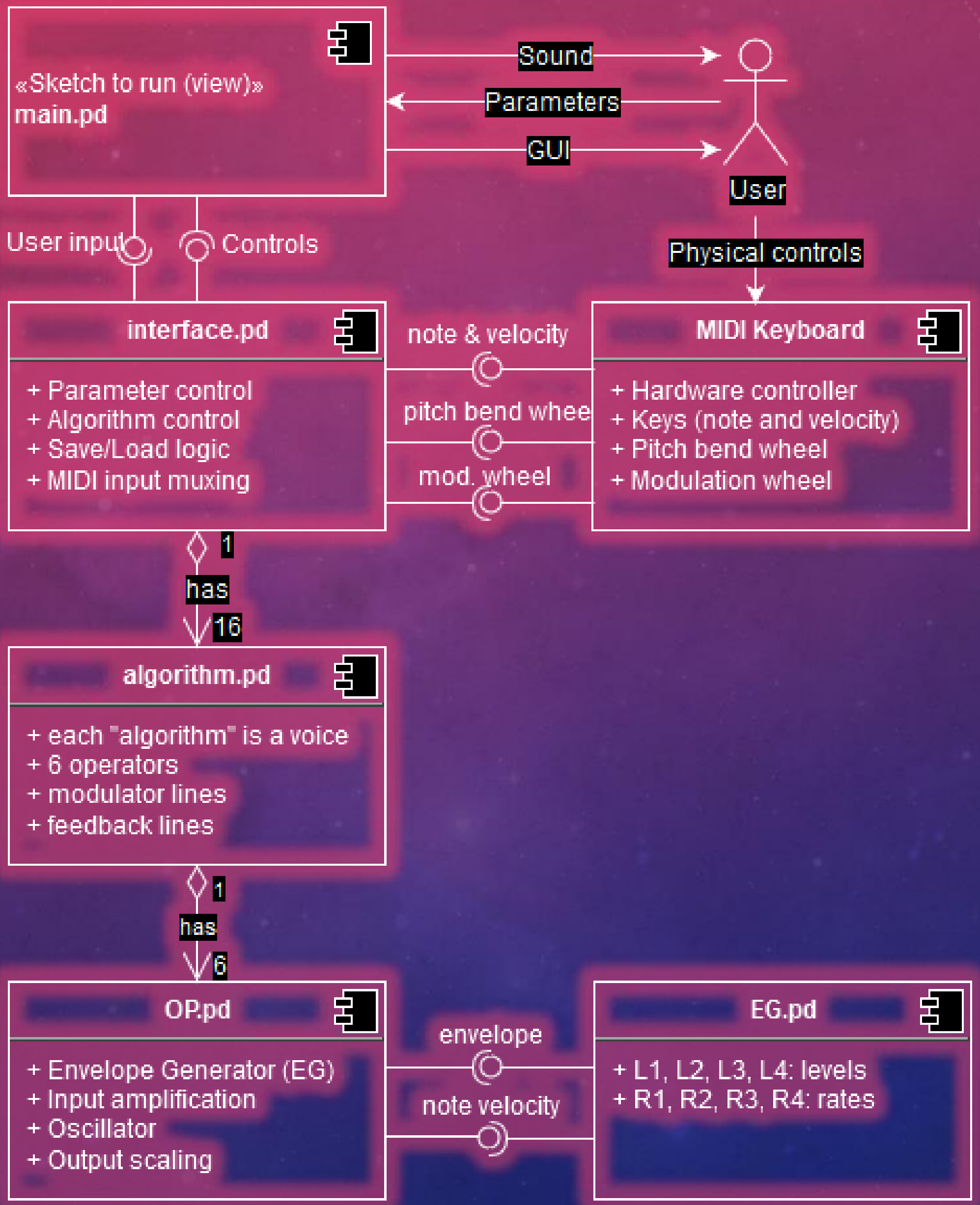
Pure Data (*vanilla*)

Extended

Pd-L2Ork

Purr Data

fmOP: main components and architecture



«Sketch to run (view)»
main.pd

Sound

Parameters

GUI

User

Physical controls

User input

Controls

interface.pd

- + Parameter control
- + Algorithm control
- + Save/Load logic
- + MIDI input muxing

note & velocity

pitch bend whee

mod. wheel

MIDI Keyboard

- + Hardware controller
- + Keys (note and velocity)
- + Pitch bend wheel
- + Modulation wheel

1

has

16

algorithm.pd

- + each "algorithm" is a voice
- + 6 operators
- + modulator lines
- + feedback lines

1

has

6

OP.pd

- + Envelope Generator (EG)
- + Input amplification
- + Oscillator
- + Output scaling

envelope

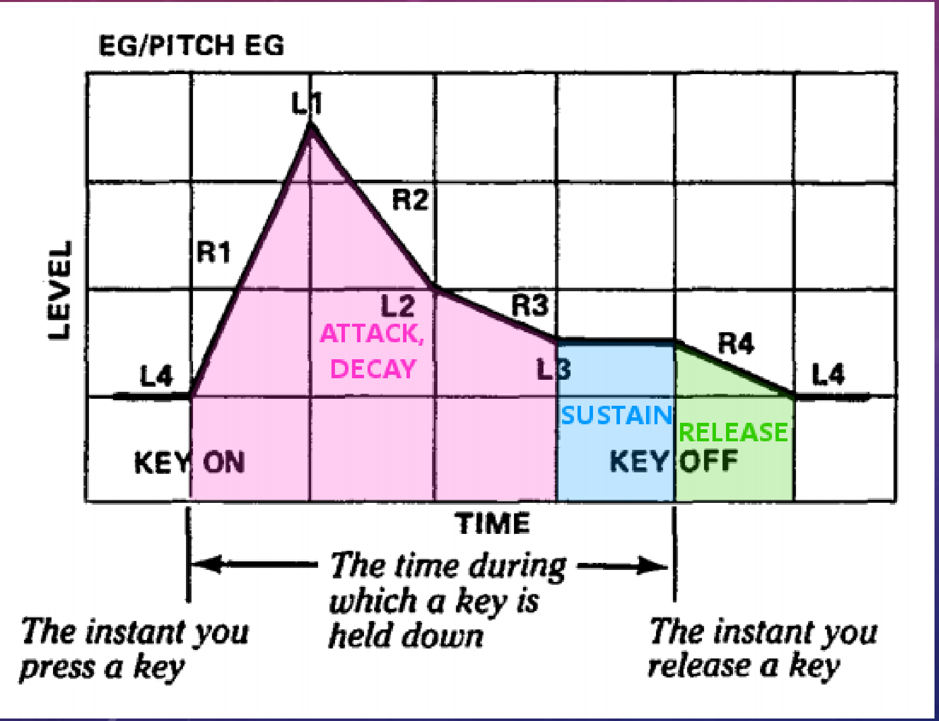
note velocity

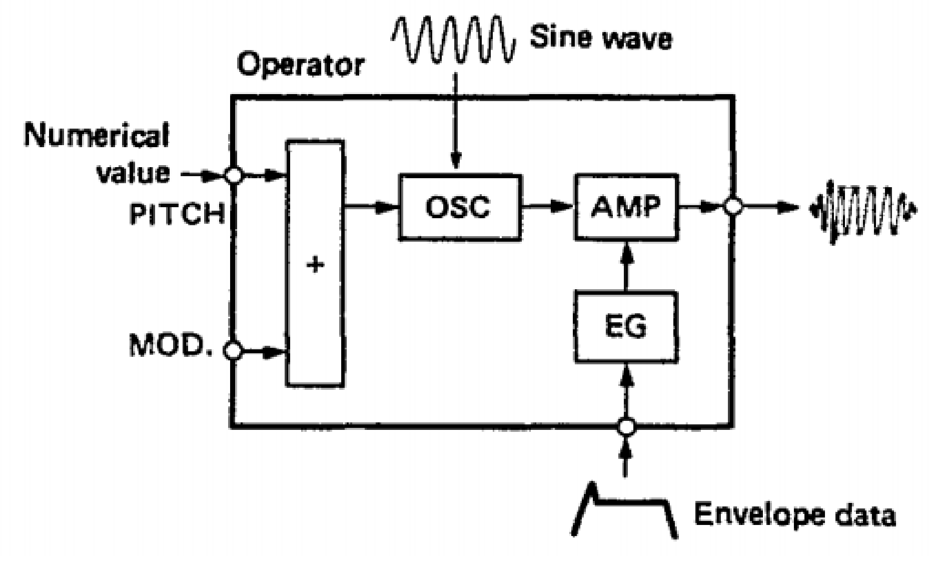
EG.pd

- + L1, L2, L3, L4: levels
- + R1, R2, R3, R4: rates

The LR4 Envelope Generator (*EG.pd, EGRate.pd*) (1/2)

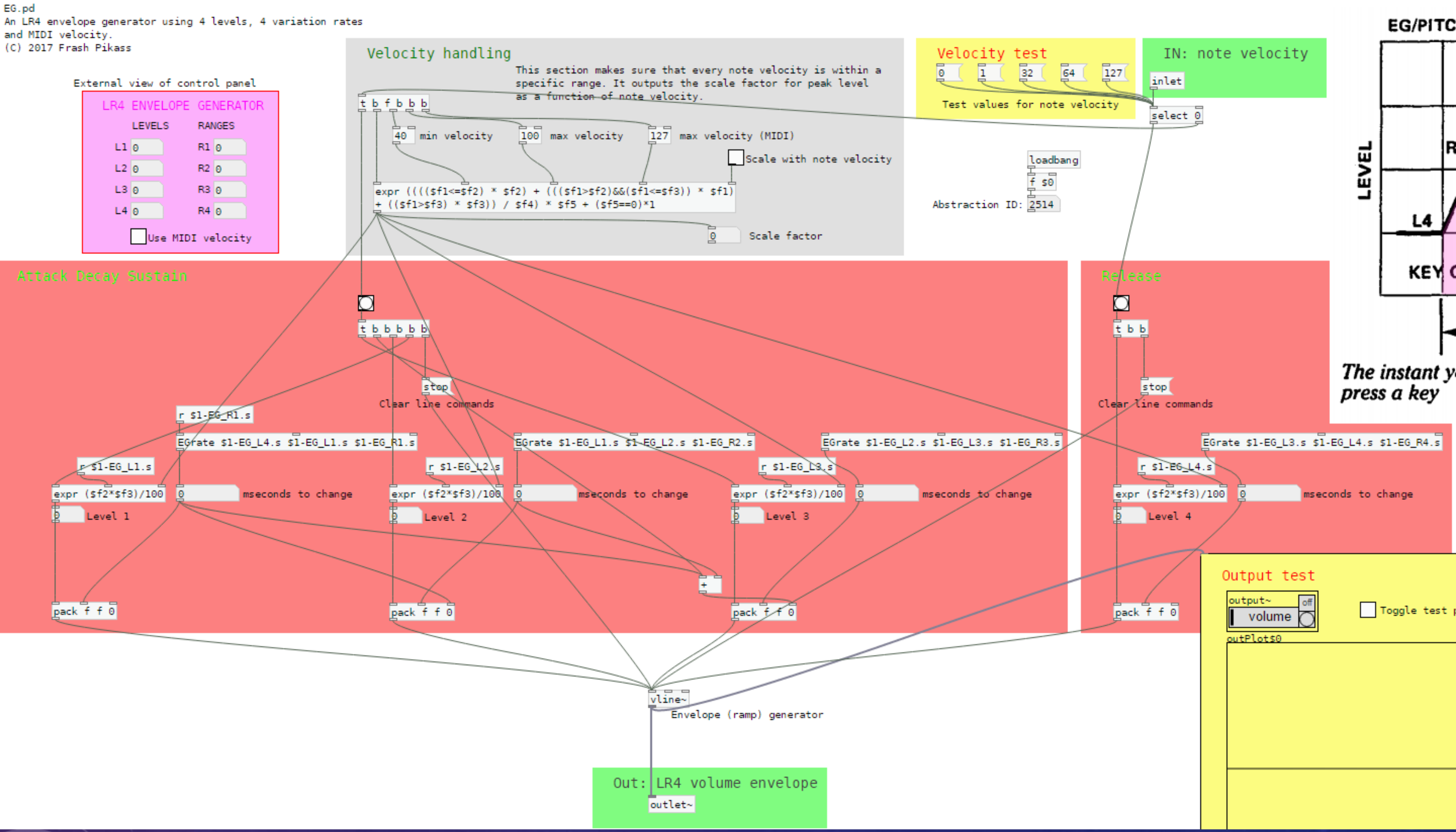
- Envelope is described in terms of 4 **Levels** and 4 **Rates**
- **Level** = amplitude point *[0, 99], linearly mapped to the range [0, 1)*
- **Rate** = transition speed *[0, 99], logarithmically mapped to the range [0, inf)*
  - At the same rate, for the same level difference, *attacks (level increments)* are faster than *decays (level drops)*
  - *Logarithmic mapping*: fast rates (close to 99) have finer differences between them; slow rates have coarser differences
- *Operator level* and *MIDI speed* (note velocity) determine the maximum note volume


EG/PITCH EG
LEVEL
L1
R2
R1
L2
ATTACK,
DECAY
R3
L4
L3
R4
L4
SUSTAIN
RELEASE
KEY ON
KEY OFF
TIME
The time during which a key is held down
The instant you press a key
The instant you release a key



(from the Yamaha DX7 Product Manual)

The LR4 Envelope Generator (*EG.pd, EGRate.pd*) (2/2)



EG.pd
An LR4 envelope generator using 4 levels, 4 variation rates
and MIDI velocity.
(C) 2017 Frash Pikass

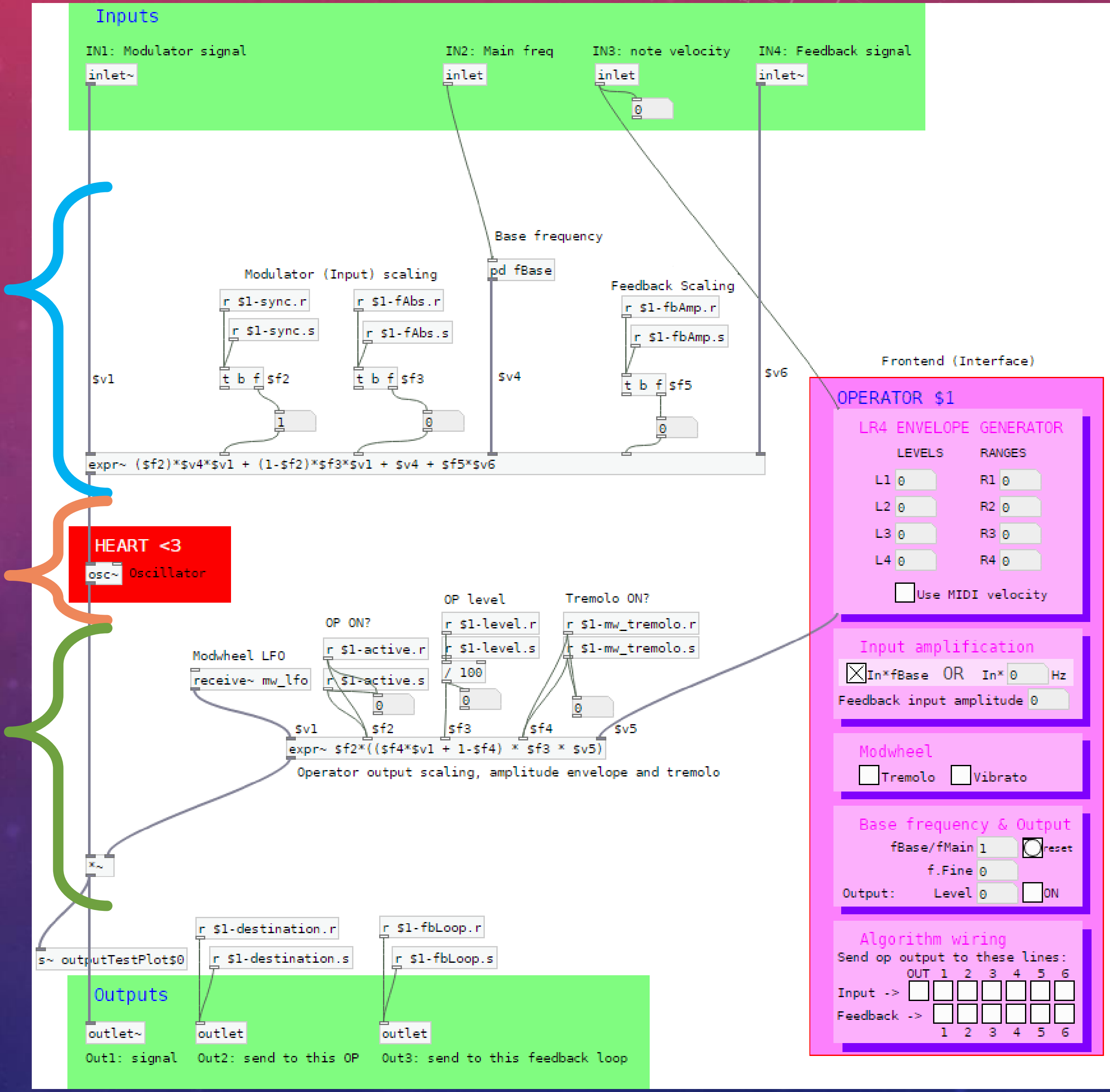
Velocity handling

This section makes sure every note velocity is within a specific range. It outputs the scale factor for peak level as a function of note velocity.

```
expr ((((($f1<=$f2) * $f2) + ((($f1>$f2)&&($f1<=$f3)) * $f1)
+ (($f1>$f3) * $f3)) / $f4) * $f5 + ($f5==0)*1
```

The FM operator (*OP.pd*) (1/5)

- It's the core of the whole synth
- It's made of three parts, in cascade:
  - The **frequency calculator**
  - The **oscillator**
  - The **envelope** (also adds **tremolo**)



```
expr~ ($f2)*$v4*$v1 + (1-$f2)*$f3*$v1 + $v4 + $f5*$v6
```

```
expr~ $f2*(($f4*$v1 + 1-$f4) * $f3 * $v5)
```

The FM operator (*OP.pd*) (2/5)

- The frequency calculator is the most important of the three parts; handles:
  - f_{main} := **main frequency** generated by the keyboard note
  - $mod(t)$:= **modulator** signal (external input)
  - $feedback(t)$:= **feedback** signal
  - $LFO_{vibrato}(t)$:= vibrato, controlled with the mod wheel on the keyboard (if active for the OP)
  - $pitch\ bend$:= pitch bend wheel on the keyboard
  - $ratio$:= the ratio between the base frequency of this OP and f_{main}
- Base frequency is computed as follows:

$$f_{base}(t) = (f_{main} + pitch\ bend) * ratio + f_{fine} + LFO_{vibrato}(t)$$

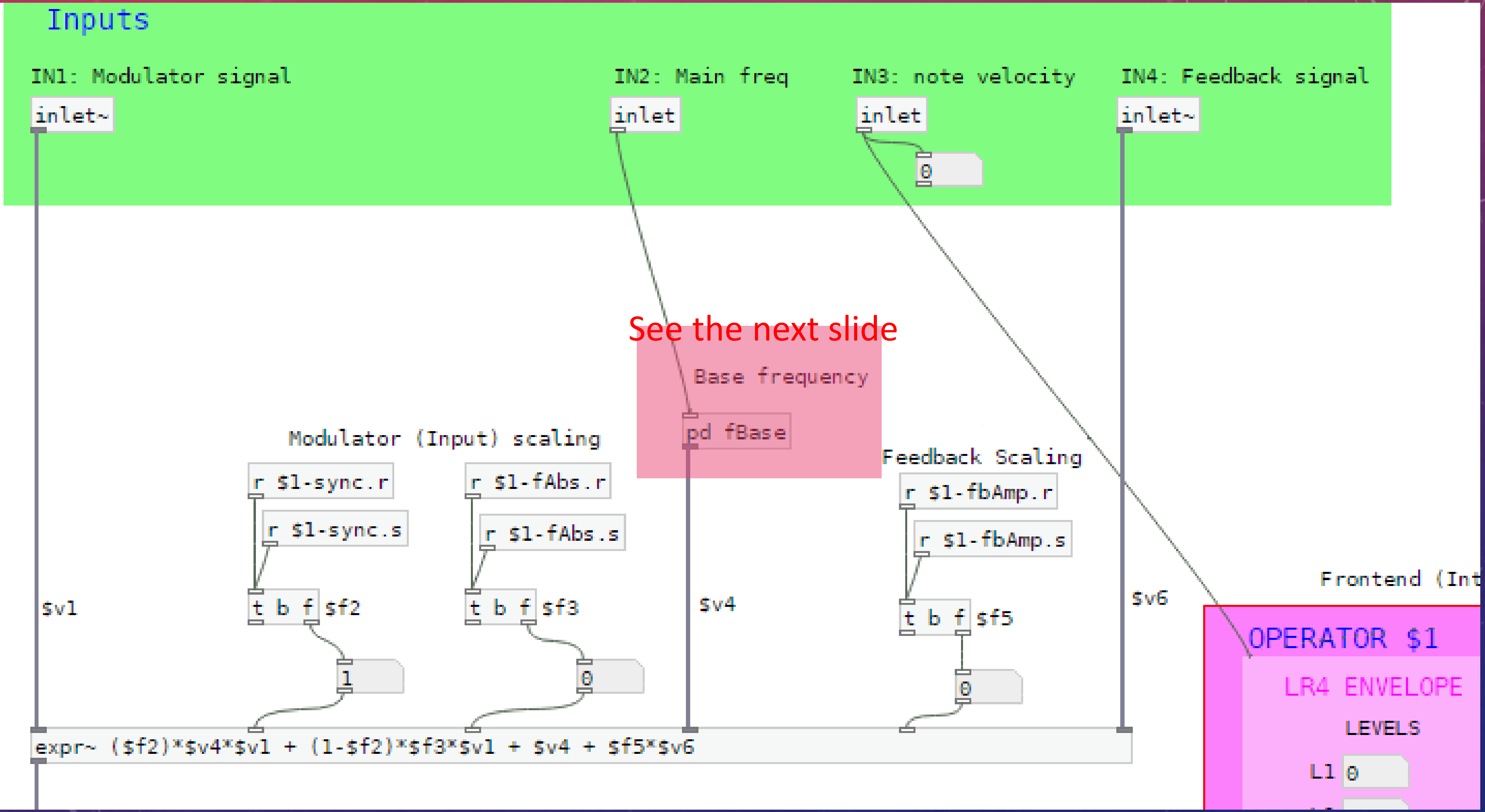
- The input signal $mod(t)$ from another OP is handled differently according with user selection on how it should be scaled/amplified:

$$a(t) = \begin{cases} f_{base}(t) * mod(t), & if\ input\ must\ be\ synchronized\ with\ f_{base}(t) \\ static\ width * mod(t), & otherwise \end{cases}$$

- The oscillator is thus represented by this function:

$$\boxed{osc(t) = \cos\left(a(t)\ + f_{base}(t)\ + amplitude_{feedback} * feedback(t)\right)}$$

The FM operator (*OP.pd*) (3/5)

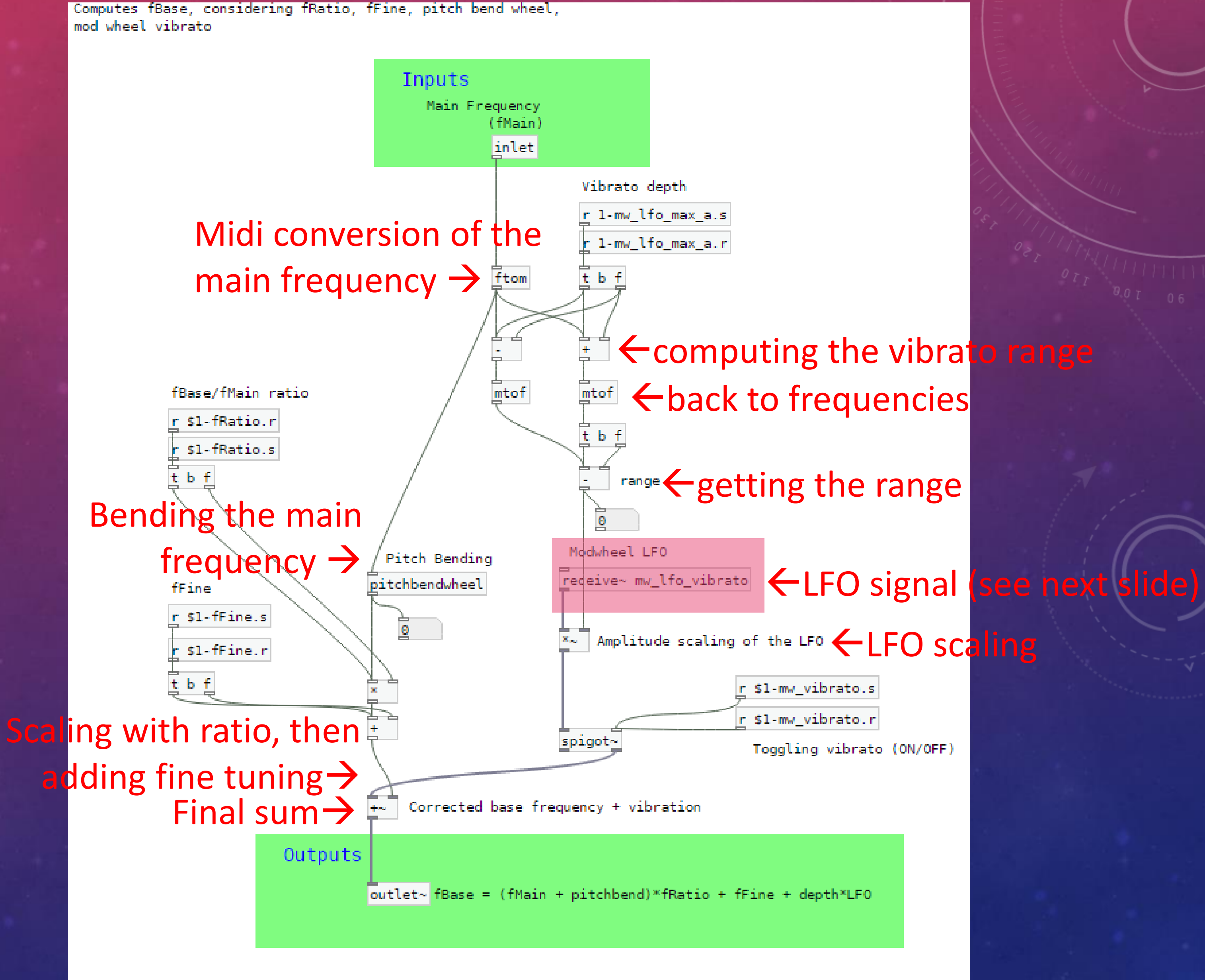


```
inlet~
inlet
inlet
inlet~
0
pd fBase
r $1-sync.r
r $1-sync.s
t b f
1
r $1-fAbs.r
r $1-fAbs.s
t b f
0
r $1-fbAmp.r
r $1-fbAmp.s
t b f
0
expr~ ($f2)*$v4*$v1 + (1-$f2)*$f3*$v1 + $v4 + $f5*$v6
0
```

See the next slide

The FM operator (*OP.pd*) (4/5)

This is how the base frequency is computed inside of the operator



Computes fBase, considering fRatio, fFine, pitch bend wheel, mod wheel vibrato

Inputs

Main Frequency (fMain)

```
inlet
ftom
```

Vibrato depth

```
r 1-mw_lfo_max_a.s
r 1-mw_lfo_max_a.r
t b f
-
+
mtof
mtof
t b f
-
```

range

```
0
```

fBase/fMain ratio

```
r $1-fRatio.r
r $1-fRatio.s
t b f
```

Pitch Bending

```
pitchbendwheel
0
```

fFine

```
r $1-fFine.s
r $1-fFine.r
t b f
*
+
```

Modwheel LFO

```
receive~ mw_lfo_vibrato
*~
```

Amplitude scaling of the LFO

```
r $1-mw_vibrato.s
r $1-mw_vibrato.r
spigot~
```

Toggling vibrato (ON/OFF)

```
+~
```

Corrected base frequency + vibration

Outputs

```
outlet~
```

fBase = (fMain + pitchbend)*fRatio + fFine + depth*LFO

Midi conversion of the main frequency →

←computing the vibrato range

←back to frequencies

←getting the range

Bending the main frequency →

←LFO signal (see next slide)

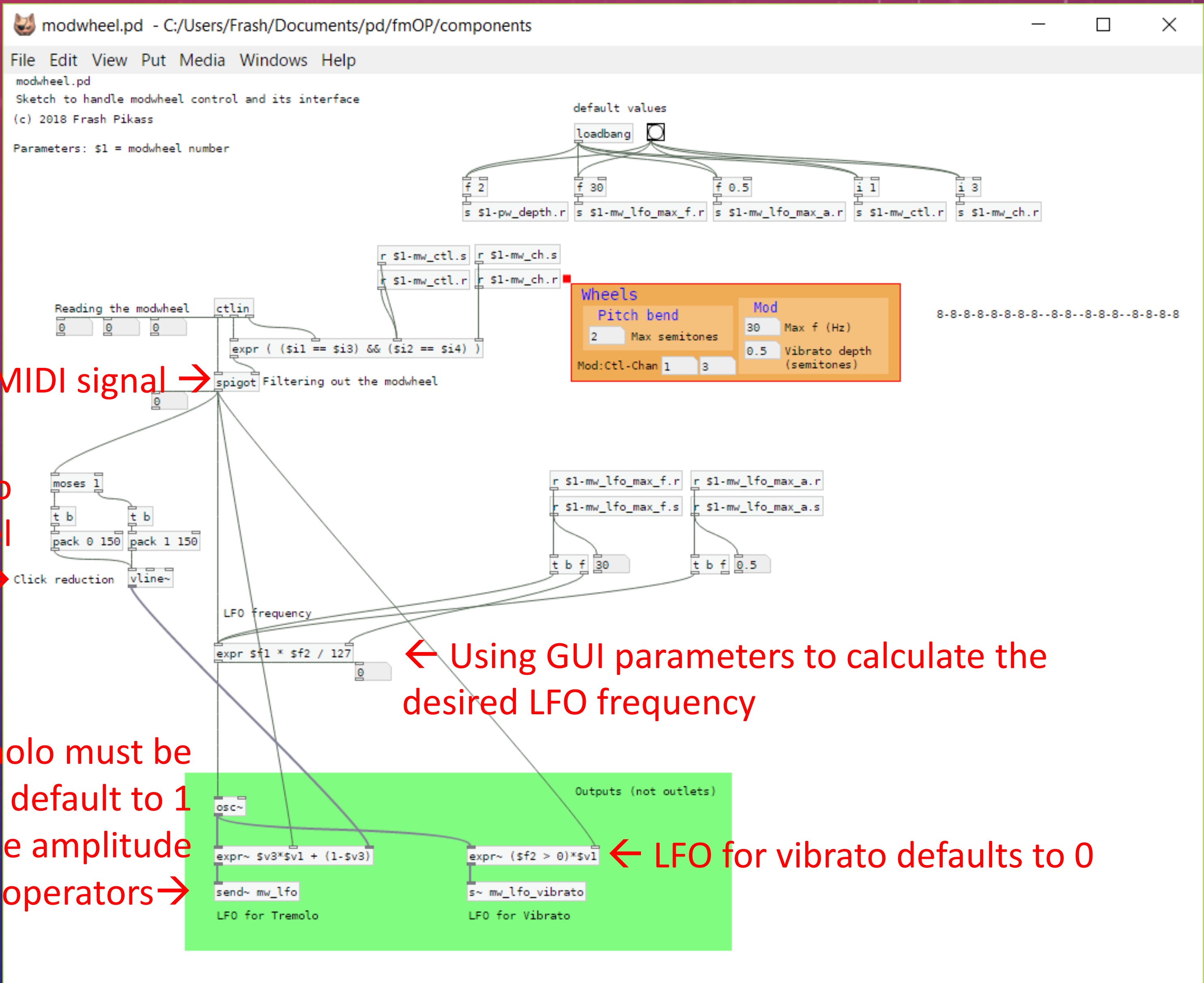
←LFO scaling

Scaling with ratio, then adding fine tuning→

Final sum→

The FM operator (*OP.pd*) (5/5)

This is how the modwheel is used to generate LFOs for tremolo and vibrato


modwheel.pd - C:/Users/Frash/Documents/pd/fmOP/components
File Edit View Put Media Windows Help

```
modwheel.pd
Sketch to handle modwheel control and its interface
(c) 2018 Frash Pikass

Parameters: $1 = modwheel number

default values
loadbang
f 2                 f 30                  f 0.5                 i 1              i 3
s $1-pw_depth.r     s $1-mw_lfo_max_f.r   s $1-mw_lfo_max_a.r   s $1-mw_ctl.r    s $1-mw_ch.r

r $1-mw_ctl.s   r $1-mw_ch.s
r $1-mw_ctl.r   r $1-mw_ch.r

Reading the modwheel   ctlin
0   0   0
expr ( ($i1 == $i3) && ($i2 == $i4) )
spigot Filtering out the modwheel
0

Wheels
  Pitch bend              Mod
  2   Max semitones       30   Max f (Hz)
                          0.5  Vibrato depth
Mod:Ctl-Chan 1   3             (semitones)

8-8-8-8-8-8-8-8--8-8--8-8-8--8-8-8-8

moses 1
t b            t b
pack 0 150     pack 1 150
Click reduction  vline~

r $1-mw_lfo_max_f.r   r $1-mw_lfo_max_a.r
r $1-mw_lfo_max_f.s   r $1-mw_lfo_max_a.s
t b f  30             t b f  0.5

LFO frequency
expr $f1 * $f2 / 127
0

Outputs (not outlets)
osc~
expr~ $v3*$v1 + (1-$v3)     expr~ ($f2 > 0)*$v1
send~ mw_lfo                s~ mw_lfo_vibrato
LFO for Tremolo             LFO for Vibrato
```

Catching the correct MIDI signal →

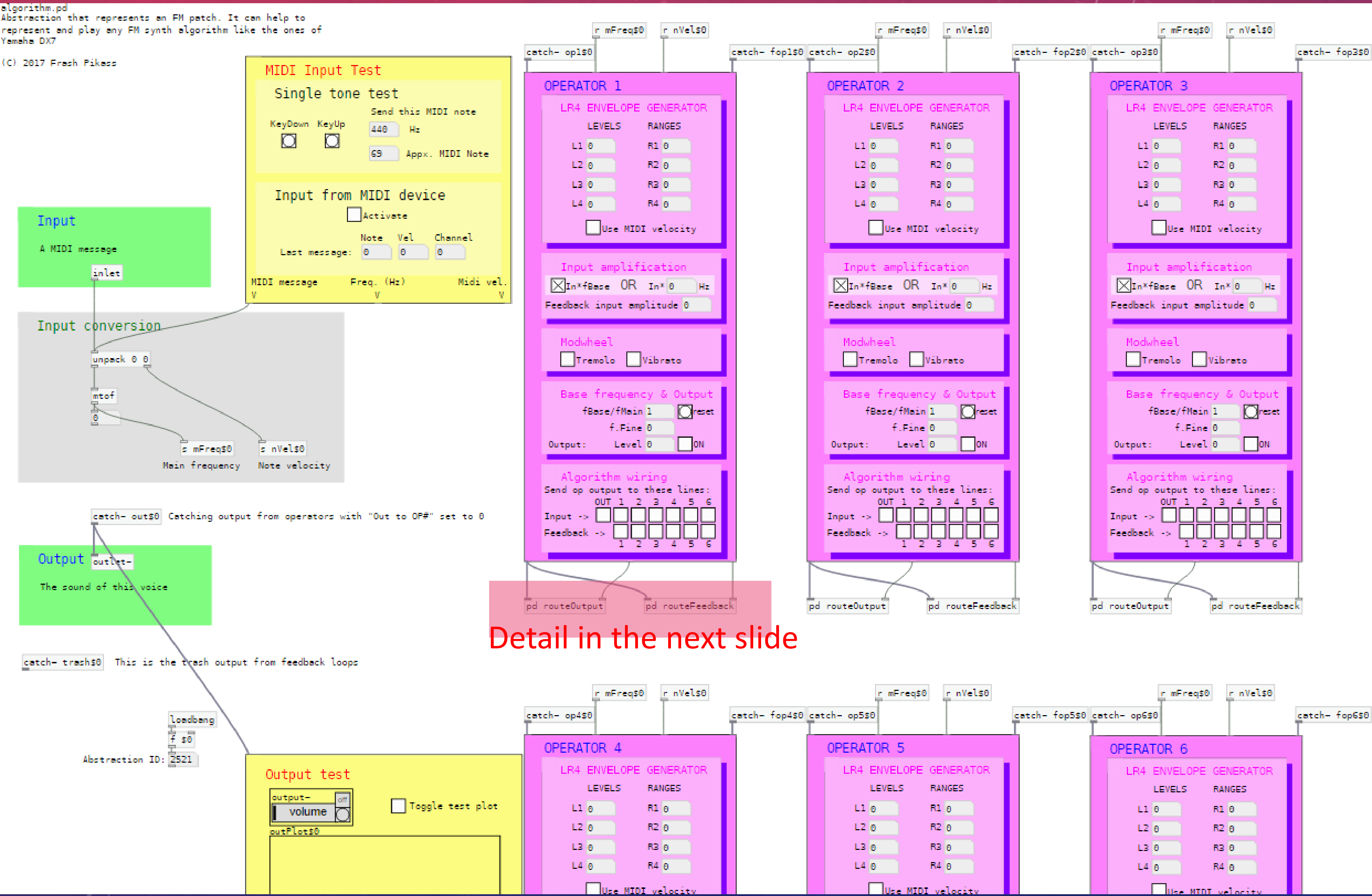
Envelope generator to smooth out wheel quantization→

← Using GUI parameters to calculate the desired LFO frequency

The LFO for tremolo must be very smooth and default to 1 since it multiplies the amplitude envelope of operators→

← LFO for vibrato defaults to 0

Wiring it all together: *algorithm.pd* (1/2)



algorithm.pd
Abstraction that represents an FM patch. It can help to represent and play any FM synth algorithm like the ones of Yamaha DX7

(C) 2017 Fresh Pikass

catch~ out$0 Catching output from operators with "Out to OP#" set to 0

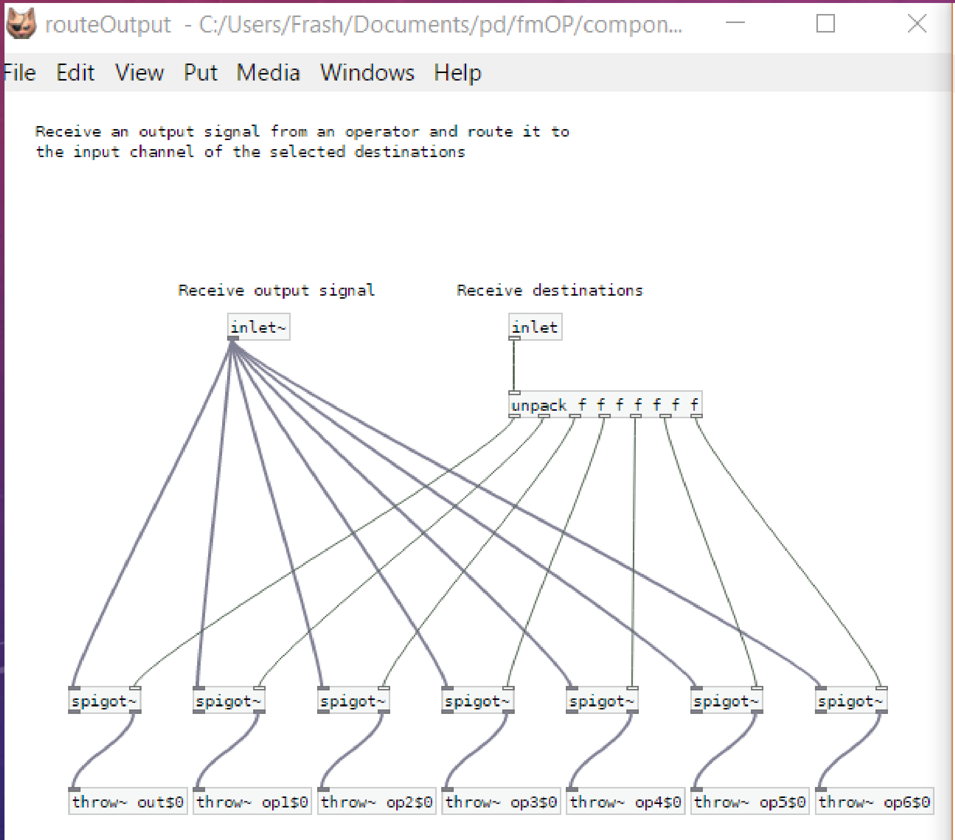
catch~ trash$0 This is the trash output from feedback loops

Abstraction ID: 2521

Detail in the next slide

Wiring it all together: *algorithm.pd* (2/2)

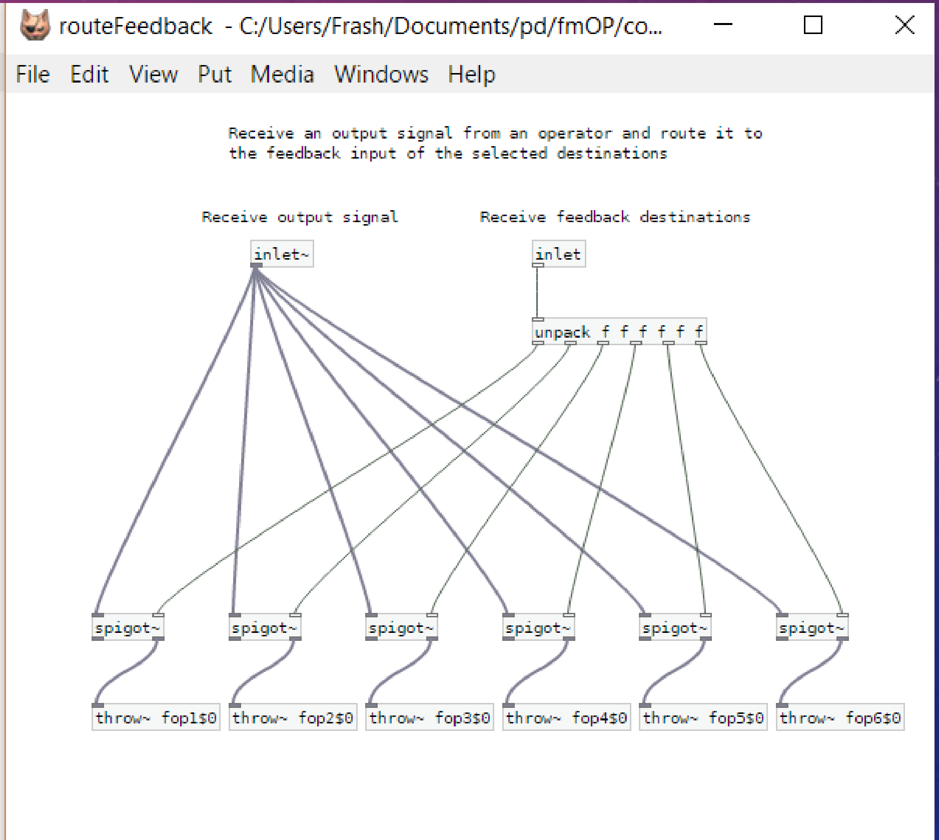
- An instance of *algorithm.pd* is both a **signal router** and an **FM voice**
- Every operator receives the same frequency and the note velocity as an input
- The user can choose from the OP panel if the output of that OP should go to the input of one or more OPs
- Output lines and feedback loops makes FM modulation possible
- This is a detail of how modulator and feedback signals are routed internally:



Receive an output signal from an operator and route it to the input channel of the selected destinations

Receive output signal

Receive destinations



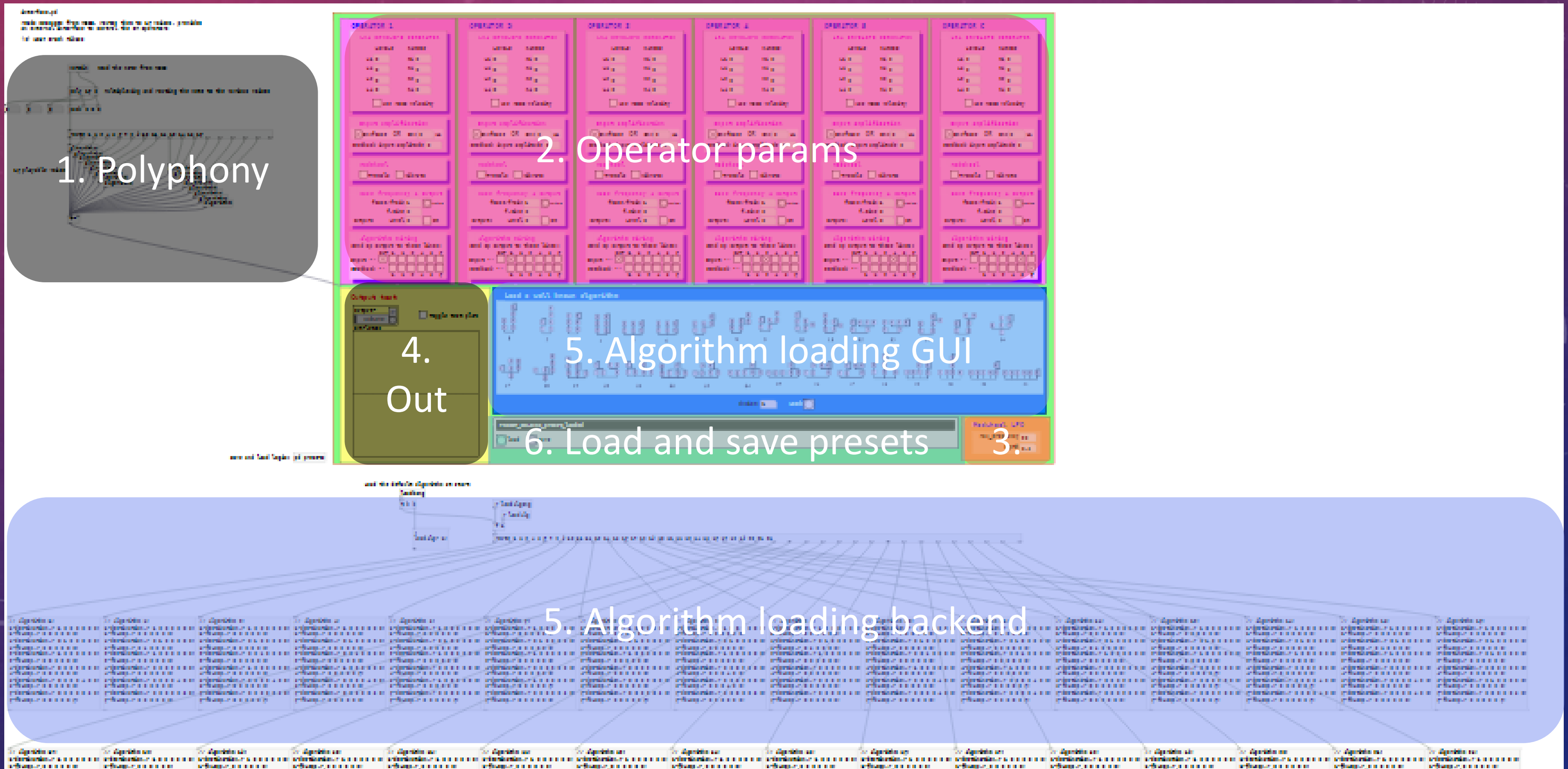
Receive an output signal from an operator and route it to the feedback input of the selected destinations

Receive output signal

Receive feedback destinations

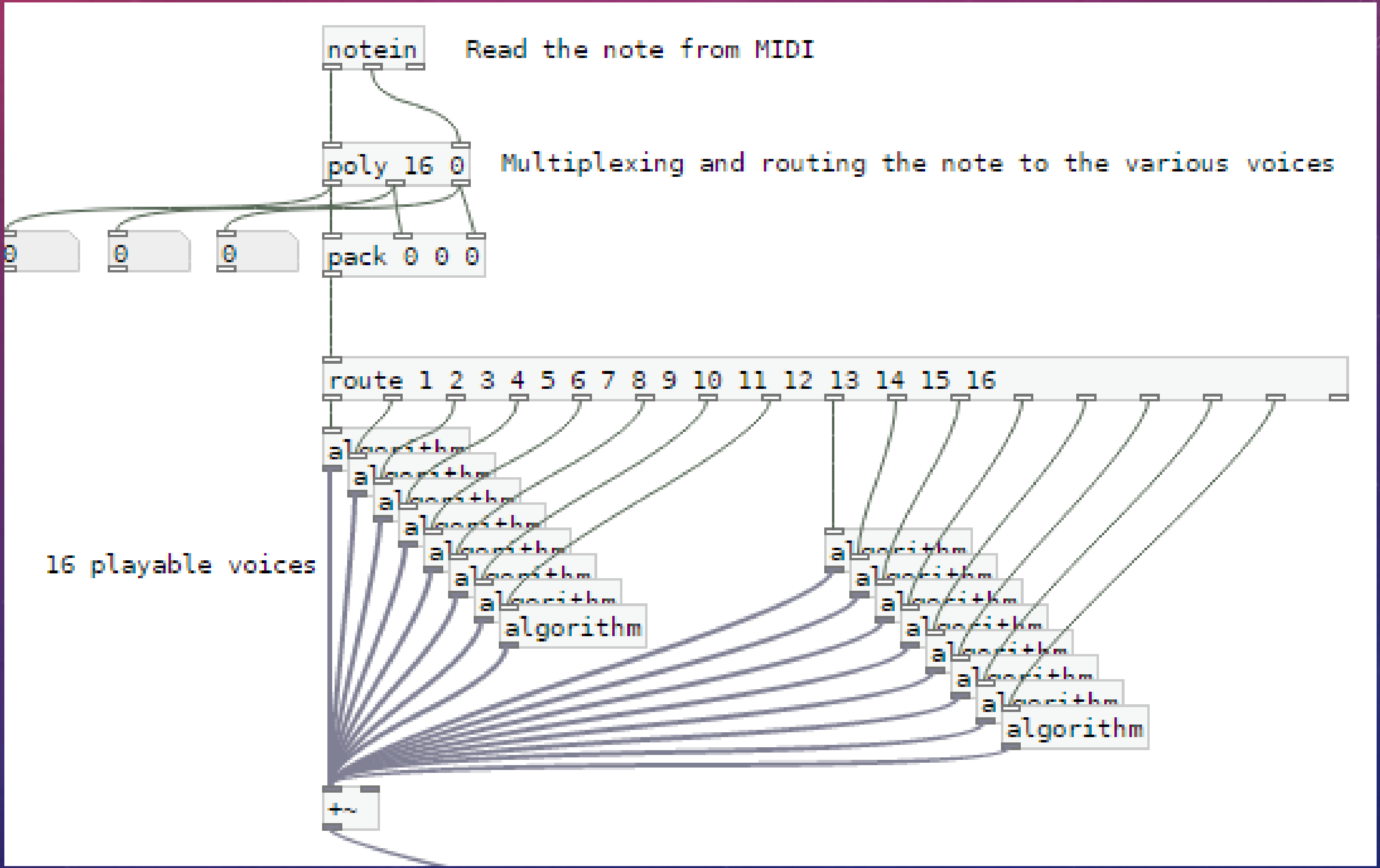
User interface: *interface.pd*

- The complete puzzle
  1. 16 voice polyphony with MIDI muxing
  2. Operator parameters
  3. Modwheel parameters
  4. A small output controller with an oscilloscope
  5. A way to load a preset algorithm (one of the original Yamaha ones)
  6. A way to load and save preset synths



User interface: Polyphony (1/6)

- Polyphony is achieved through 16 instances of *algorithm.pd* and the midi multiplexer *poly*
- Each *algorithm* is a voice of the synth which can play in parallel with each other
- The output of all algorithms is summed up at the end



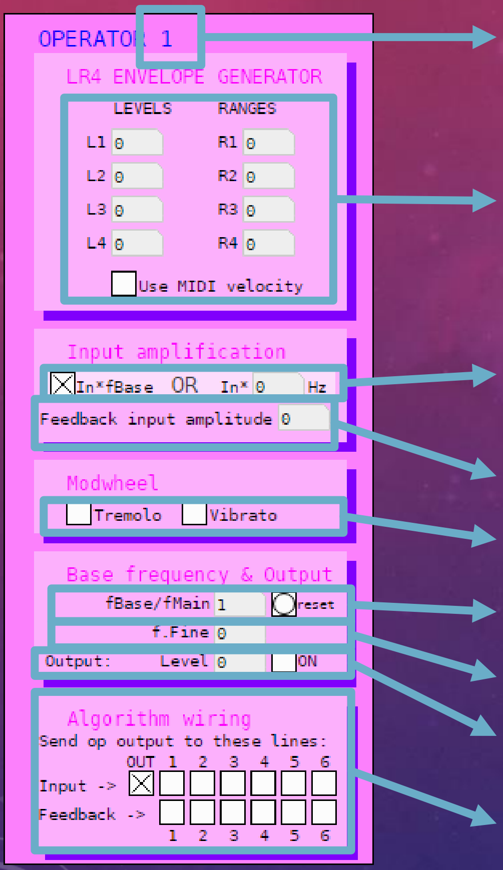
Read the note from MIDI

Multiplexing and routing the note to the various voices

16 playable voices

User interface: Operator Parameters (2/6)

- The six operators in the interface are defined in such a way that their parameters are automatically sent to their counterparts in each algorithm



OPERATOR 1

LR4 ENVELOPE GENERATOR

LEVELS		RANGES	
L1	0	R1	0
L2	0	R2	0
L3	0	R3	0
L4	0	R4	0

□ Use MIDI velocity

Input amplification

☒ In*fBase OR In* 0 Hz

Feedback input amplitude 0

Modwheel

□ Tremolo □ Vibrato

Base frequency & Output

fBase/fMain 1 ○ reset

f.Fine 0

Output: Level 0 □ ON

Algorithm wiring

Send op output to these lines:

	OUT	1	2	3	4	5	6
Input ->	☒	□	□	□	□	□	□
Feedback ->		□	□	□	□	□	□
		1	2	3	4	5	6

Operator number

Envelope generator parameters

Scale modulator (input) signal frequency by fBase OR scale it by a fixed frequency

Scale the feedback loop input by this factor

Apply modwheel effects to this operator

Frequency ratio (this OP’s base frequency to main frequency)

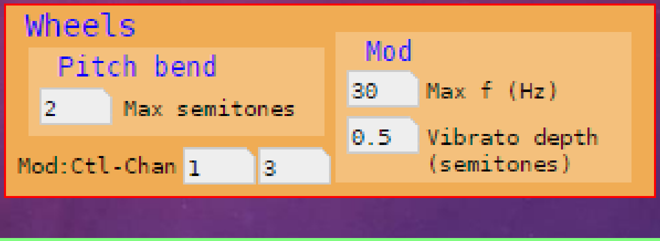
Frequency fine tuning

Operator main output level and ON/OFF switch

FM algorithm wiring (output→input and output→feedback loop)

User interface: Mod and Pitch wheels (3/6)

- The mod wheel can add **tremolo** and/or **vibrato** to selected operators
- Modulation happens thanks to an **LFO** (low frequency oscillator)
- Moving the mod wheel up increases the LFO frequency up to the **max value**
- The user can specify the **depth of the vibrato** in semitones
- The user can also specify the **pitch bend strength** in semitones
- The mod wheel MIDI channel and control number can be specified



Wheels

Pitch bend

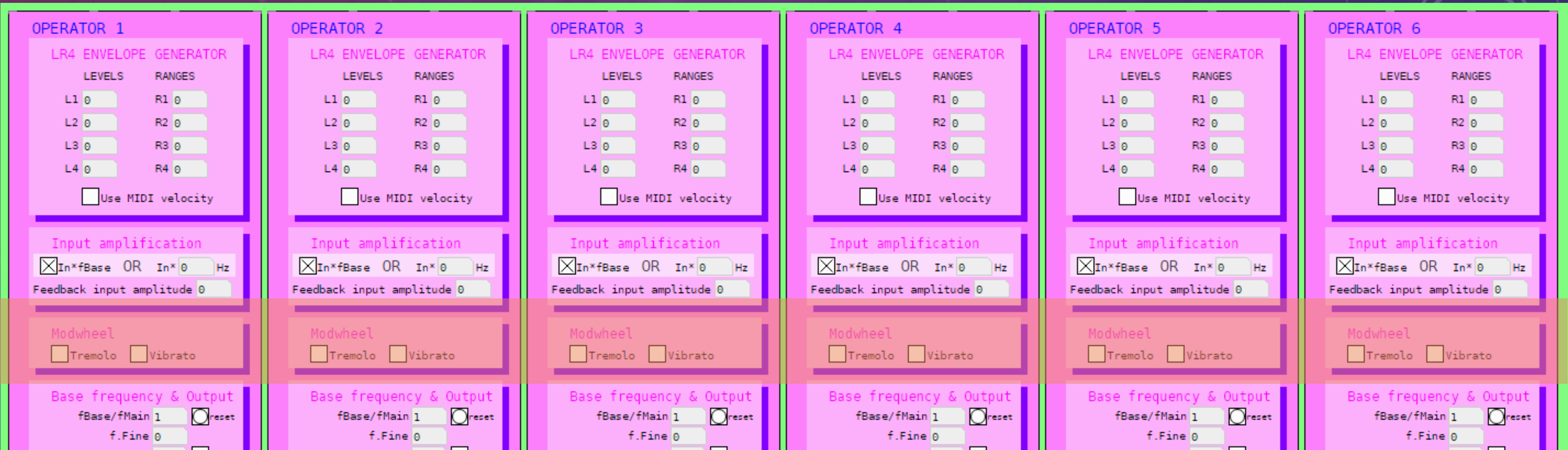
2 Max semitones

Mod:Ctl-Chan 1 3

Mod

30 Max f (Hz)

0.5 Vibrato depth (semitones)



OPERATOR 1

LR4 ENVELOPE GENERATOR

LEVELS RANGES

L1 0 R1 0

L2 0 R2 0

L3 0 R3 0

L4 0 R4 0

□ Use MIDI velocity

Input amplification

☒ In*fBase OR In* 0 Hz

Feedback input amplitude 0

Modwheel

□ Tremolo □ Vibrato

Base frequency & Output

fBase/fMain 1 ○ reset

f.Fine 0

OPERATOR 2

LR4 ENVELOPE GENERATOR

LEVELS RANGES

L1 0 R1 0

L2 0 R2 0

L3 0 R3 0

L4 0 R4 0

□ Use MIDI velocity

Input amplification

☒ In*fBase OR In* 0 Hz

Feedback input amplitude 0

Modwheel

□ Tremolo □ Vibrato

Base frequency & Output

fBase/fMain 1 ○ reset

f.Fine 0

OPERATOR 3

LR4 ENVELOPE GENERATOR

LEVELS RANGES

L1 0 R1 0

L2 0 R2 0

L3 0 R3 0

L4 0 R4 0

□ Use MIDI velocity

Input amplification

☒ In*fBase OR In* 0 Hz

Feedback input amplitude 0

Modwheel

□ Tremolo □ Vibrato

Base frequency & Output

fBase/fMain 1 ○ reset

f.Fine 0

OPERATOR 4

LR4 ENVELOPE GENERATOR

LEVELS RANGES

L1 0 R1 0

L2 0 R2 0

L3 0 R3 0

L4 0 R4 0

□ Use MIDI velocity

Input amplification

☒ In*fBase OR In* 0 Hz

Feedback input amplitude 0

Modwheel

□ Tremolo □ Vibrato

Base frequency & Output

fBase/fMain 1 ○ reset

f.Fine 0

OPERATOR 5

LR4 ENVELOPE GENERATOR

LEVELS RANGES

L1 0 R1 0

L2 0 R2 0

L3 0 R3 0

L4 0 R4 0

□ Use MIDI velocity

Input amplification

☒ In*fBase OR In* 0 Hz

Feedback input amplitude 0

Modwheel

□ Tremolo □ Vibrato

Base frequency & Output

fBase/fMain 1 ○ reset

f.Fine 0

OPERATOR 6

LR4 ENVELOPE GENERATOR

LEVELS RANGES

L1 0 R1 0

L2 0 R2 0

L3 0 R3 0

L4 0 R4 0

□ Use MIDI velocity

Input amplification

☒ In*fBase OR In* 0 Hz

Feedback input amplitude 0

Modwheel

□ Tremolo □ Vibrato

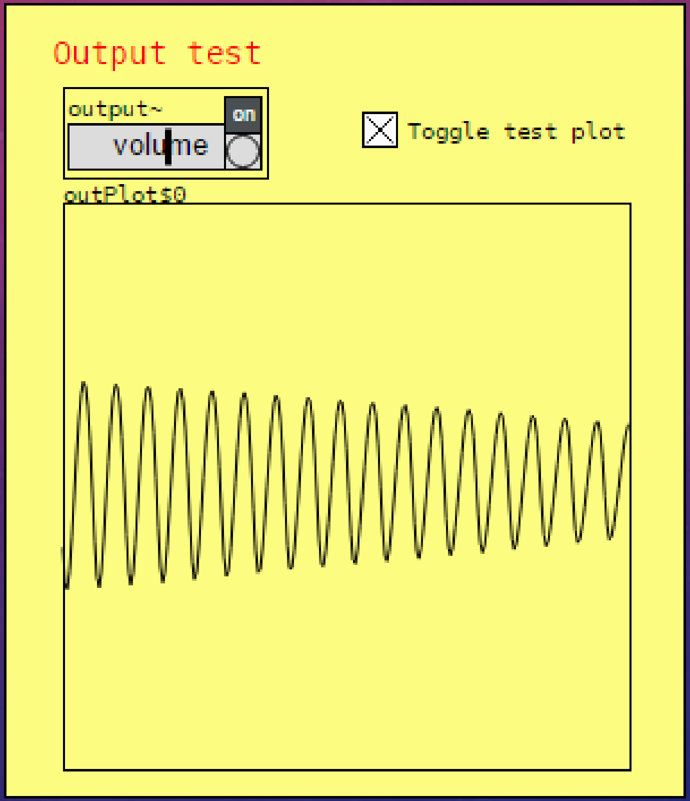
Base frequency & Output

fBase/fMain 1 ○ reset

f.Fine 0

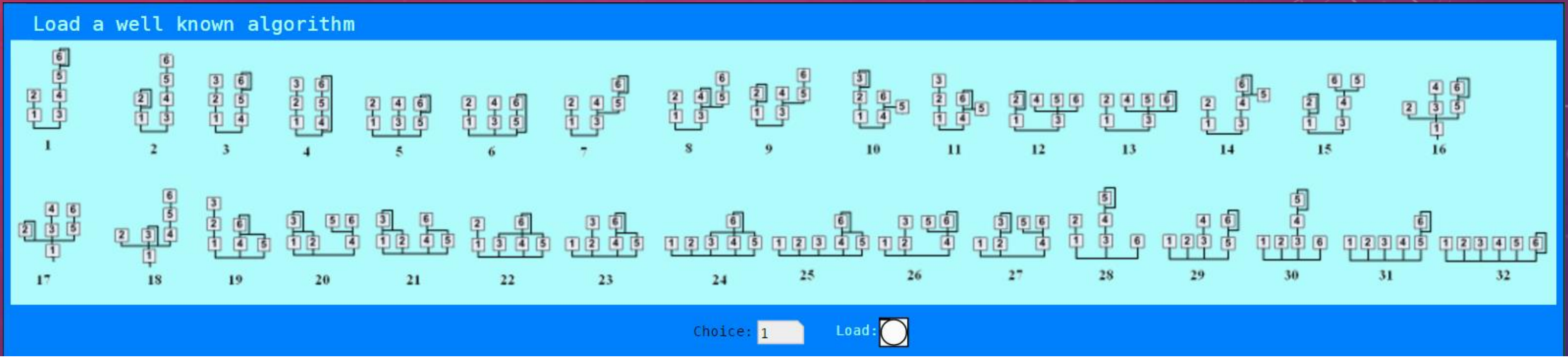
User interface: Output control (*testOutput.pd*) (4/6)

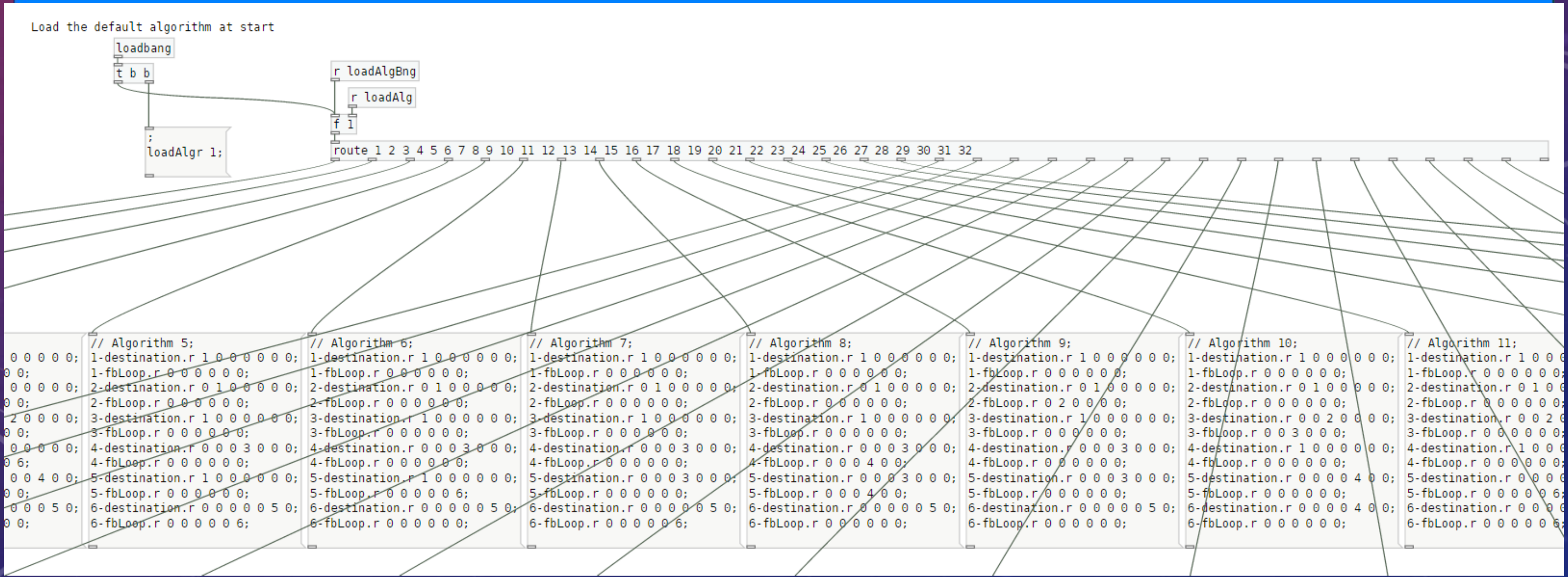
- The Output interface is an extension of the Purr Data standard one
- Other than offering volume control, allows to see the output
- Very convenient for sound design!



User interface: Algorithm Loading (5/6)

- From this portion of the GUI the user can select an algorithm
- Provided algorithms are the same which could be found on Yamaha DX7
- It takes a while to load the correct algorithm to all the operators (*wait until the load button blinks!*)
- The backend manages to load hardcoded algorithms by using pd messages





```
Load the default algorithm at start
loadbang
t b b
;
loadAlgr 1;

r loadAlgBng
r loadAlg
f 1
route 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
```

```
 0 0 0 0 0;
0 0;
 0 0 0 0 0;
0 0;
 2 0 0 0 0;
0 0;
 0 0 0 0 0;
0 6;
 0 0 4 0 0;
0 0;
 0 0 0 5 0;
0 0;
```

```
// Algorithm 5;
1-destination.r 1 0 0 0 0 0 0;
1-fbLoop.r 0 0 0 0 0 0;
2-destination.r 0 1 0 0 0 0 0;
2-fbLoop.r 0 0 0 0 0 0;
3-destination.r 1 0 0 0 0 0 0;
3-fbLoop.r 0 0 0 0 0 0;
4-destination.r 0 0 0 3 0 0 0;
4-fbLoop.r 0 0 0 0 0 0;
5-destination.r 1 0 0 0 0 0 0;
5-fbLoop.r 0 0 0 0 0 0;
6-destination.r 0 0 0 0 0 5 0;
6-fbLoop.r 0 0 0 0 0 6;
```

```
// Algorithm 6;
1-destination.r 1 0 0 0 0 0 0;
1-fbLoop.r 0 0 0 0 0 0;
2-destination.r 0 1 0 0 0 0 0;
2-fbLoop.r 0 0 0 0 0 0;
3-destination.r 1 0 0 0 0 0 0;
3-fbLoop.r 0 0 0 0 0 0;
4-destination.r 0 0 0 3 0 0 0;
4-fbLoop.r 0 0 0 0 0 0;
5-destination.r 1 0 0 0 0 0 0;
5-fbLoop.r 0 0 0 0 0 6;
6-destination.r 0 0 0 0 0 5 0;
6-fbLoop.r 0 0 0 0 0 0;
```

```
// Algorithm 7;
1-destination.r 1 0 0 0 0 0 0;
1-fbLoop.r 0 0 0 0 0 0;
2-destination.r 0 1 0 0 0 0 0;
2-fbLoop.r 0 0 0 0 0 0;
3-destination.r 1 0 0 0 0 0 0;
3-fbLoop.r 0 0 0 0 0 0;
4-destination.r 0 0 0 3 0 0 0;
4-fbLoop.r 0 0 0 0 0 0;
5-destination.r 0 0 0 3 0 0 0;
5-fbLoop.r 0 0 0 0 0 0;
6-destination.r 0 0 0 0 0 5 0;
6-fbLoop.r 0 0 0 0 0 6;
```

```
// Algorithm 8;
1-destination.r 1 0 0 0 0 0 0;
1-fbLoop.r 0 0 0 0 0 0;
2-destination.r 0 1 0 0 0 0 0;
2-fbLoop.r 0 0 0 0 0 0;
3-destination.r 1 0 0 0 0 0 0;
3-fbLoop.r 0 0 0 0 0 0;
4-destination.r 0 0 0 3 0 0 0;
4-fbLoop.r 0 0 0 4 0 0;
5-destination.r 0 0 0 3 0 0 0;
5-fbLoop.r 0 0 0 4 0 0;
6-destination.r 0 0 0 0 0 5 0;
6-fbLoop.r 0 0 0 0 0 0;
```

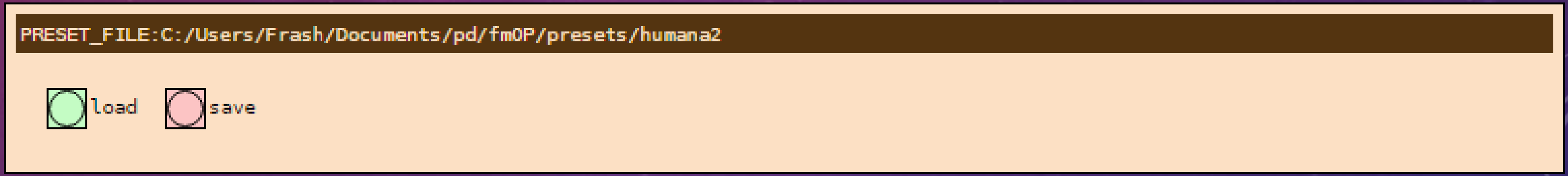
```
// Algorithm 9;
1-destination.r 1 0 0 0 0 0 0;
1-fbLoop.r 0 0 0 0 0 0;
2-destination.r 0 1 0 0 0 0 0;
2-fbLoop.r 0 2 0 0 0 0;
3-destination.r 1 0 0 0 0 0 0;
3-fbLoop.r 0 0 0 0 0 0;
4-destination.r 0 0 0 3 0 0 0;
4-fbLoop.r 0 0 0 0 0 0;
5-destination.r 0 0 0 3 0 0 0;
5-fbLoop.r 0 0 0 0 0 0;
6-destination.r 0 0 0 0 0 5 0;
6-fbLoop.r 0 0 0 0 0 0;
```

```
// Algorithm 10;
1-destination.r 1 0 0 0 0 0 0;
1-fbLoop.r 0 0 0 0 0 0;
2-destination.r 0 1 0 0 0 0 0;
2-fbLoop.r 0 0 0 0 0 0;
3-destination.r 0 0 2 0 0 0 0;
3-fbLoop.r 0 0 3 0 0 0;
4-destination.r 1 0 0 0 0 0 0;
4-fbLoop.r 0 0 0 0 0 0;
5-destination.r 0 0 0 0 4 0 0;
5-fbLoop.r 0 0 0 0 0 0;
6-destination.r 0 0 0 0 4 0 0;
6-fbLoop.r 0 0 0 0 0 0;
```

```
// Algorithm 11;
1-destination.r 1 0 0 0
1-fbLoop.r 0 0 0 0 0 0;
2-destination.r 0 1 0 0
2-fbLoop.r 0 0 0 0 0 0;
3-destination.r 0 0 2 0
3-fbLoop.r 0 0 0 0 0 0;
4-destination.r 1 0 0 0
4-fbLoop.r 0 0 0 0 0 0;
5-destination.r 0 0 0 0
5-fbLoop.r 0 0 0 0 0 6;
6-destination.r 0 0 0 0
6-fbLoop.r 0 0 0 0 0 6;
```

User interface: Preset save/load (6/6)

- Users can save and load presets to external files
- The process is fast, efficient and distributed over multiple pd files (so it's a bit tricky to understand)
- It relies on specific variable naming conventions and some special abstractions which deal with files and pd messages, but its Vanilla friendly
- See https://forum.pdpatchrepo.info/topic/9887/save-presets-to-textfile


PRESET_FILE:C:/Users/Frash/Documents/pd/fmOP/presets/humana2

Thank you for your attention!

Have fun with fmOP and follow me on Github for further updates!

https://github.com/frashpikass/fmOP/