DII, University of Brescia
Signals and Communication LAB

PIPE ORGAN EMULATION
with Pure Data

A.Y. 2013-2014

Nicola Simoni

University of Brescia
Communication Technologies and Multimedia
Digital Audio Processing

TABLE OF CONTENTS

MOTIVATION

THE IMPORTANCE OF THE PIPE ORGAN

Pipe organ is one of the oldest instruments. Its invention is accredited to Greeks, in the 3rd century BC.

It was used by many different cultures. In the ancient world it was played in stadiums during competitions and also during wars, to scare the enemies thanks to its impressive sound.

Then it became the favorite instrument of emperors and also the official instrument of the Church.

Nowadays it is still used in churches and during some classical concerts.

In modern music is used an electronic version of the pipe organ, known as Hammond organ.

OUR GOAL

Pipe organs, unluckily, have some disadvantages.
In fact they:

- are complex to be built
- need in general a quite big space for the installation
- need a periodic maintenance
- are very expensive!

The aim of our work is to create a model of the pipe organ and implement it in software.

In this way the sound of the pipe organ can be easily (and cheaply) generated with the use of a computer, an external MIDI keyboard and an amplification system.

WE PROCEED IN THE FOLLOWING WAY

1. First of all we will study the working principle of a real pipe organ.
2. Then we will find a mathematical (and efficient) way that allows to reproduce its sound.
3. Finally we will implement the model in software.

We decided to use the programming language **Pure Data** because it has been designed specifically for multimedia applications, and it can be easily embedded in almost any machine (e.g.: Raspberry Pi).

THE PIPE ORGAN

LIVE IS BETTER

The best way to understand how a thing works...is to see it live!

So we looked for a real pipe organ. We chose the one that is located in the church of Carzago della Riviera (it is represented in the picture below).

HOW DOES A PIPE ORGAN WORK?

We were given a book that tells the story of the restoration.
It contains also some good pictures that we reported here to help us with the explanation.

Let's start to analyze how a pipe organ work.

The instrument is composed of several parts. The main ones are:

1. Consolle
2. Pipes
3. Connections
4. Bellows
5. Tremolo generator
6. Windchest

CONSOLLE

The consolle is the interface between the organ and the organ player.

It is composed by one (or more) **keyboard(s)** and a set of **pedals**. When a key or a pedal is pressed, air is driven into the pipes that produce the corresponding sound.

There are also the **registers**, that allow to decide which rank of pipes is reached by the air.

PIPES

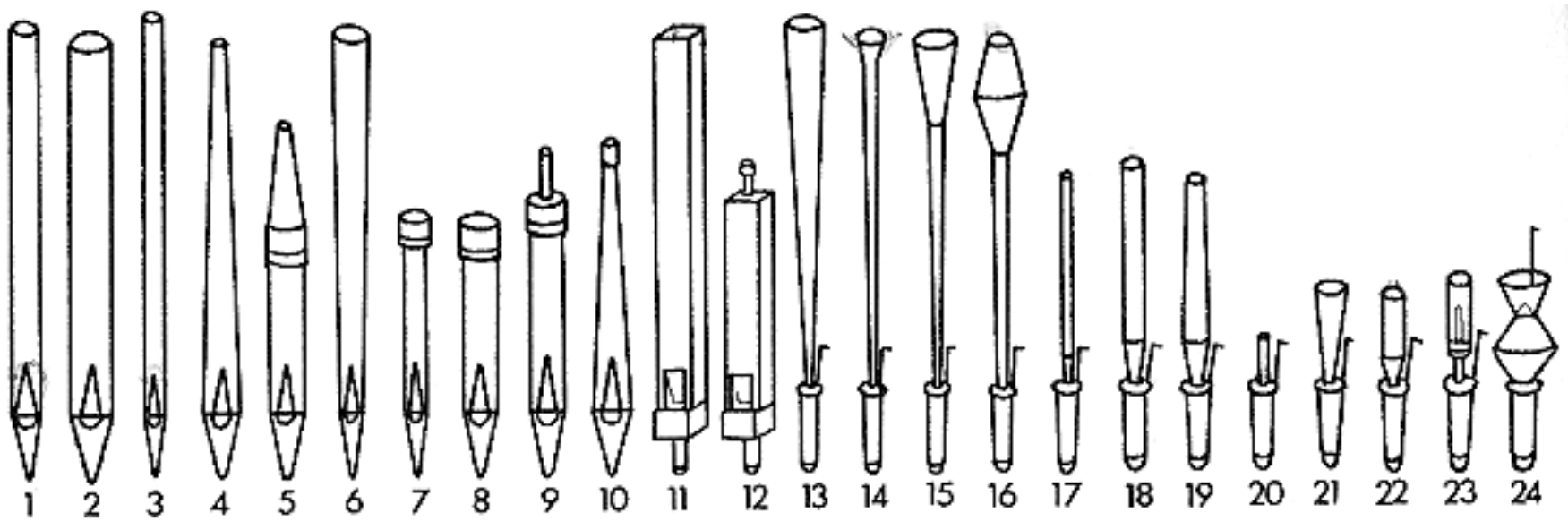
Yes, we said rank of pipes. In fact, the usual set of pipes that we are used to see is only a small portion of them. The others are hidden behind the organ.

But why so many pipes?
The reason
is that **each rank of pipes produces a different timbre**.
This is due to the different materials and shapes used.

As an example,
the organ at issue (that
is considered a small one)
contains in total 1230 pipes!

DIFFERENT PIPES

In the picture are shown some of the different pipes:



Flue pipes:

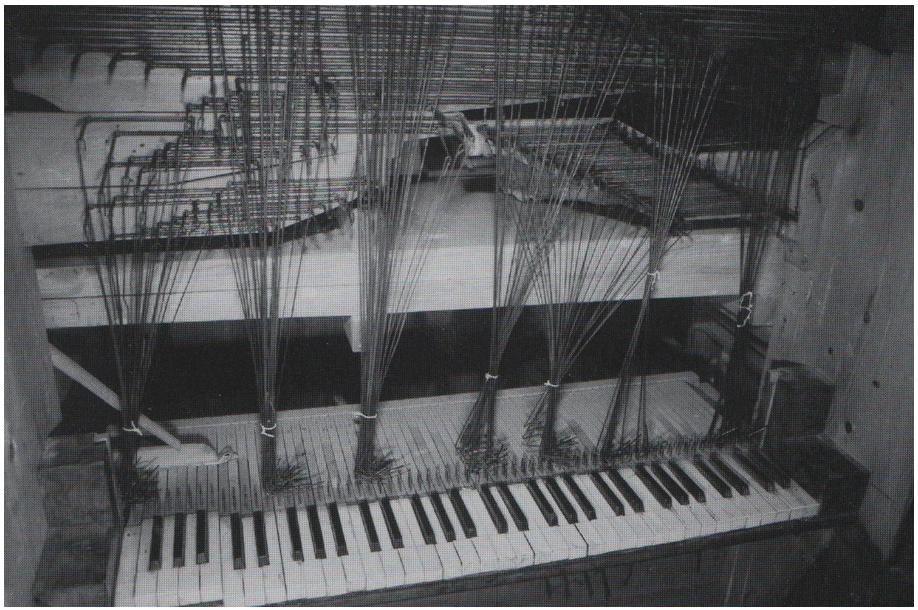
1 Principal
2 Flute
3 Viole
4 Spitzflute
5 Koppelflute
6 Trichterflute
7 Quintaton
8 Gedeckt / Bourdon
9 Rohrflute
10 Spitzgedeckt
11 Open Wood
12 Stopped Wood

Reed pipes:

13 Trumpet
14 Schalmei
15 Oboe
16 English Horn
17 Krummhorn
18 Dulcian
19 Musette
20 Regale
21 Tricher Regale
22 Vox humana
23 Rankett
24 Baerpfeife

CONNECTIONS

A complex system of **trackers** (tiranti) and **rollerboards** (catenacciature) connects the keyboad, the pedals, the registers and the pipes together.



In this way, according to the selected registers, some ranks of pipes are opened (they will produce sound), while others are closed on the bottom (the so called organ stop).

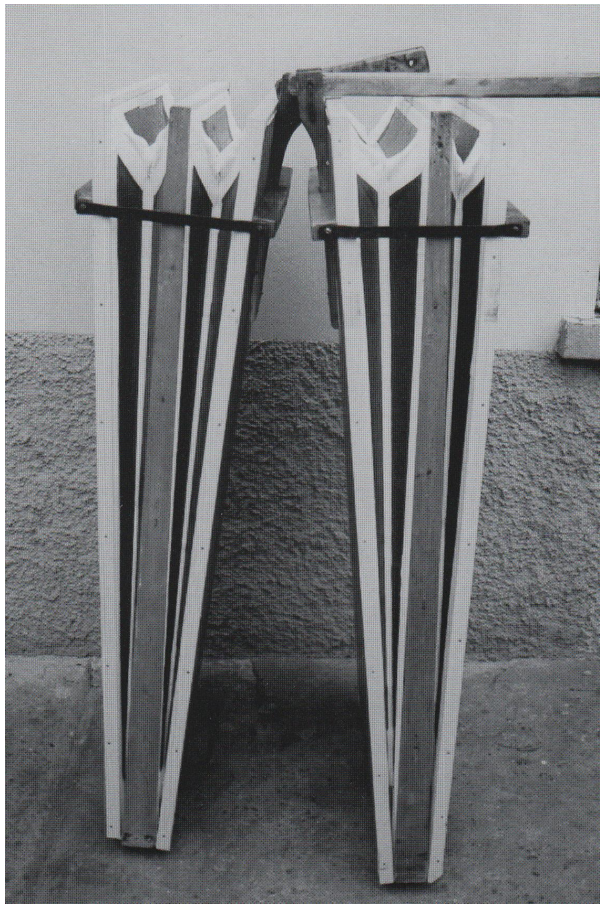
As an example, the organ at issue is equipped with 24 registers.

BELLOWS

The **bellows** (mantice) is the engine of the pipe organ.
It is an air pump that provides a constant air flow to the pipes.

In the past they were driven by water or by human force.

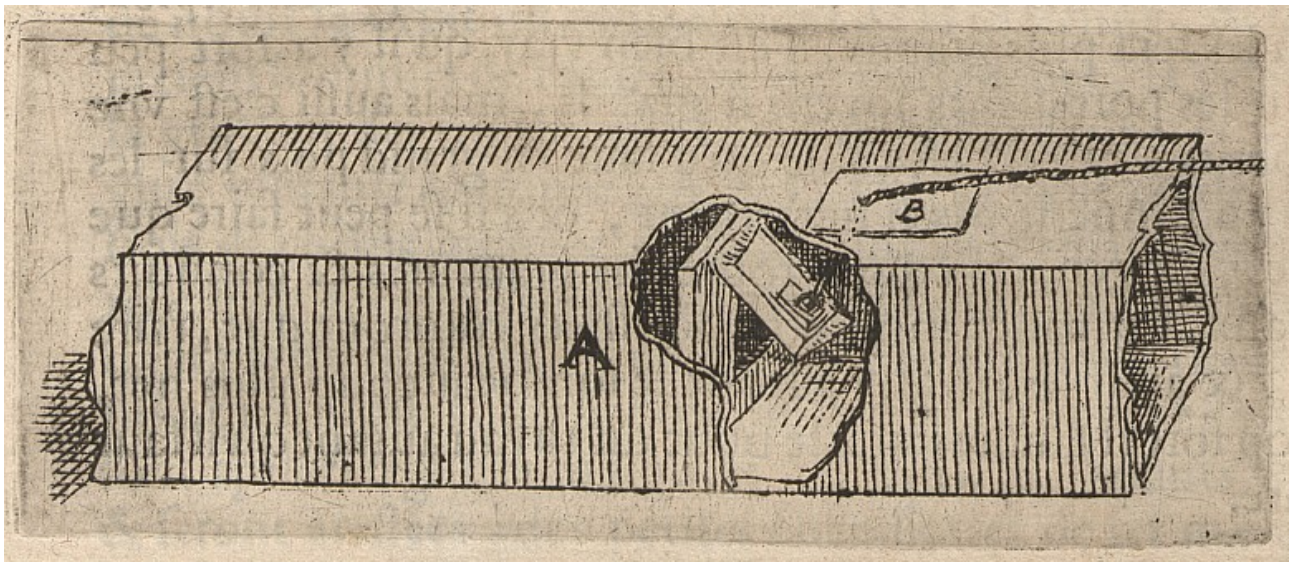
Since the invention of the electric motor, most of the pipe organs were modified and equipped with electronic driven bellows.



TREMOLO GENERATOR

The **tremolo generator** produces the tremolo effect.

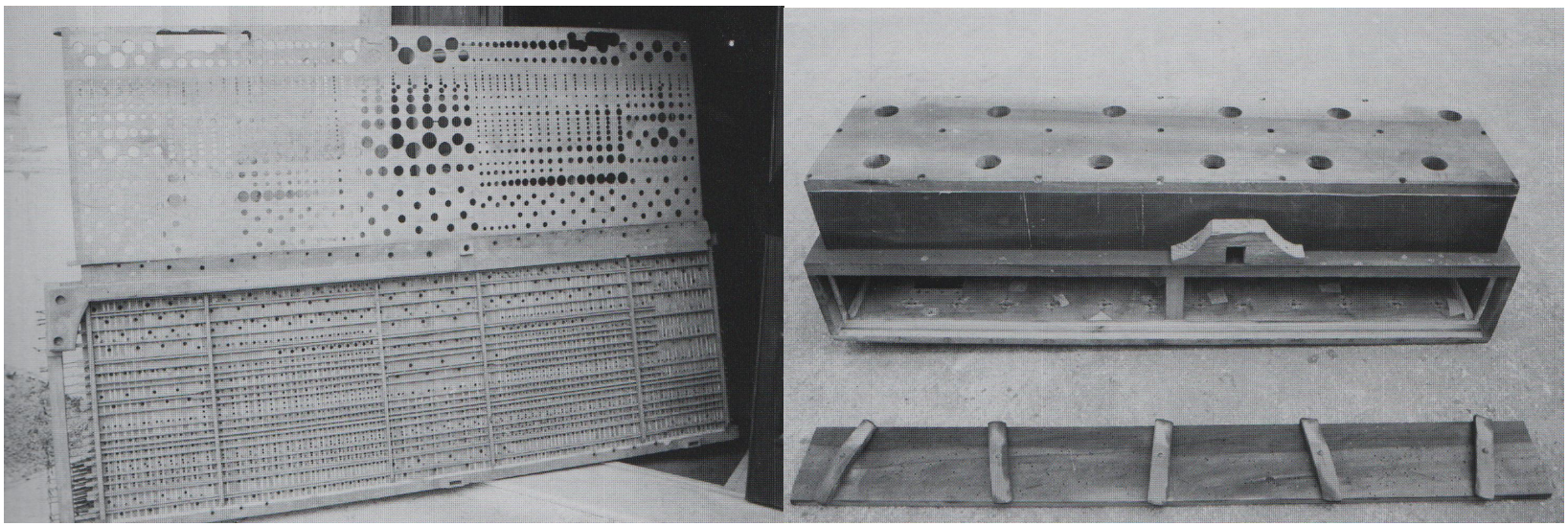
It is obtained by varying the intensity of the air blown by the bellows.



It is a mechanism that modulates the air flow by the means of an oscillating movement.

WINDCHEST

The **windchest** (somiere) is the frame of the organ, usually built with walnut.
It is air-proof and its function is to collect the air from the bellows and feed the pipes.



In the picture we can see the upper part of the big windchest and the small windchest (for the horn pipes).

THE SOUND GENERATION

THE PIPE

In order to derive a mathematical model of the pipe organ, we need to understand how the sound is physically generated.

As we already said, sound is produced by driving air into the pipes.

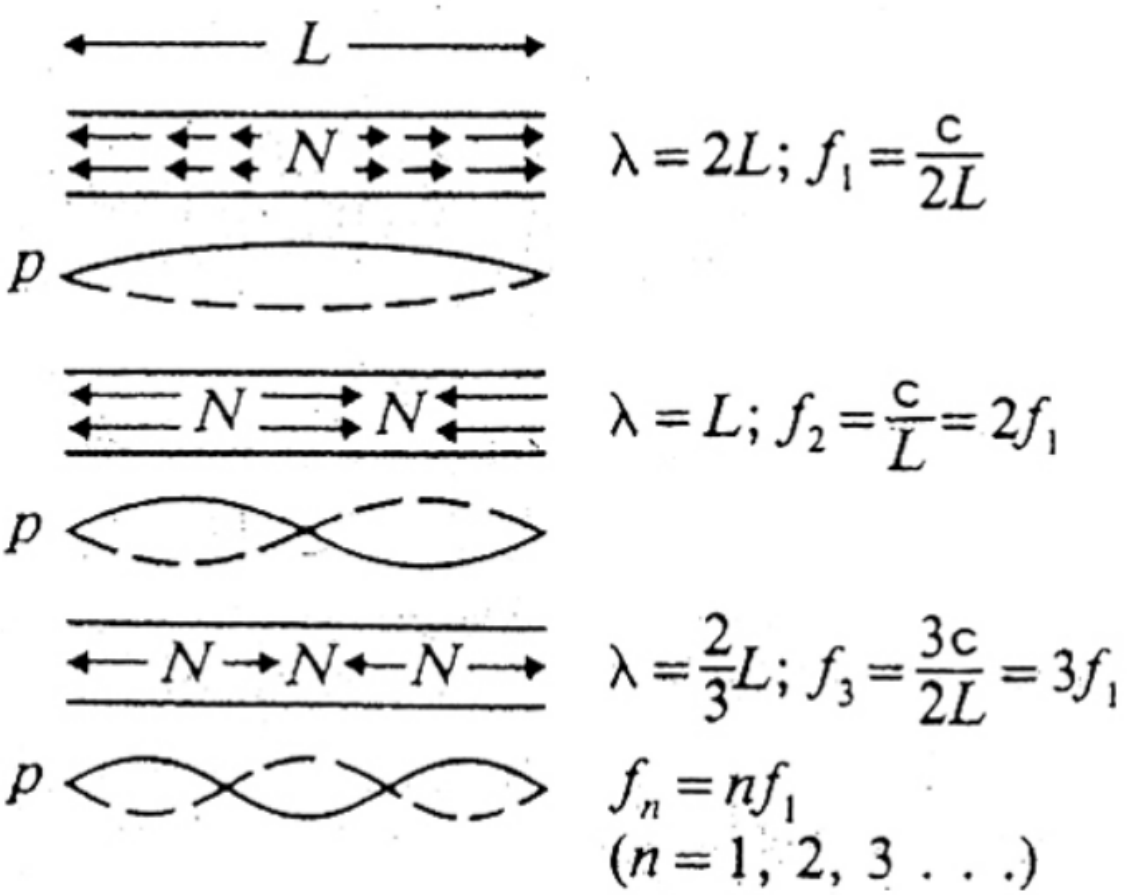
A pipe can be considered (approximatively) as a **cylinder opened at both extremities**. If we analyze its resonance frequencies, we find out the following relationship:

$$f = \frac{nv}{2L}$$

Where **f** is the resonant frequency, **n** is an integer positive number, **v** is the speed of sound and **L** is the length of the cylinder.

RESONANCE MODES

In the picture are represented the first three modes:



THE TIMBRE

So **a pipe produces a sound that is equal to the sum of the sinusoids at the various resonant frequencies**.

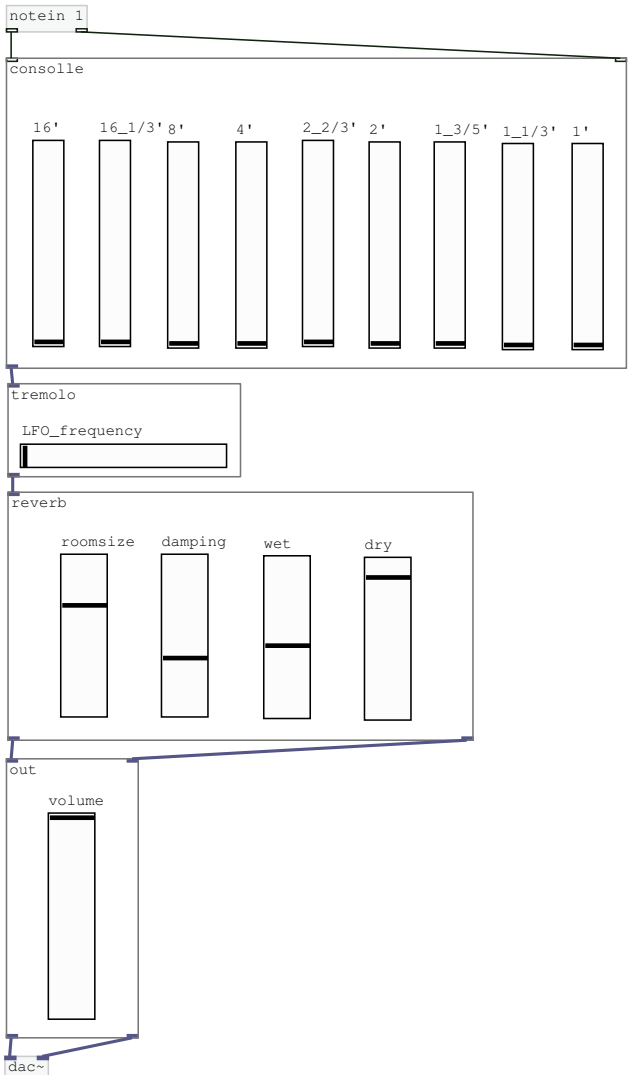
What makes the difference in sound between the various types of pipes is the **shape** and the **material** used.

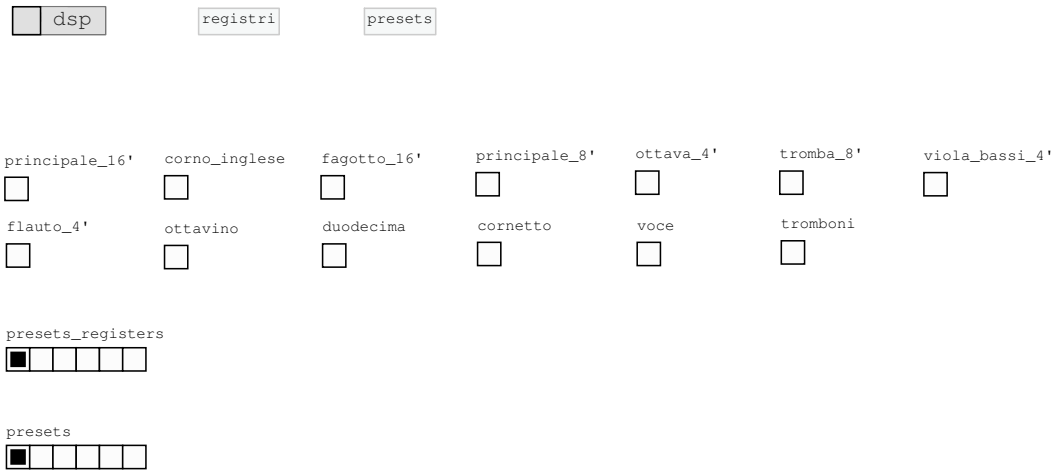
That is, **what makes two pipes sound different is their harmonic content** (the different amplitudes of the resonant frequencies).

This information is FUNDAMENTAL for the mathematical model: in fact this allows us to use the **additive synthesis technique**.

THE CODE

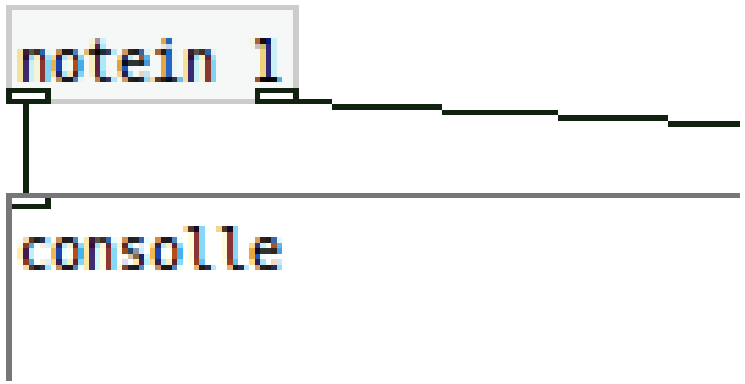
THE INTERFACE





THE INPUT

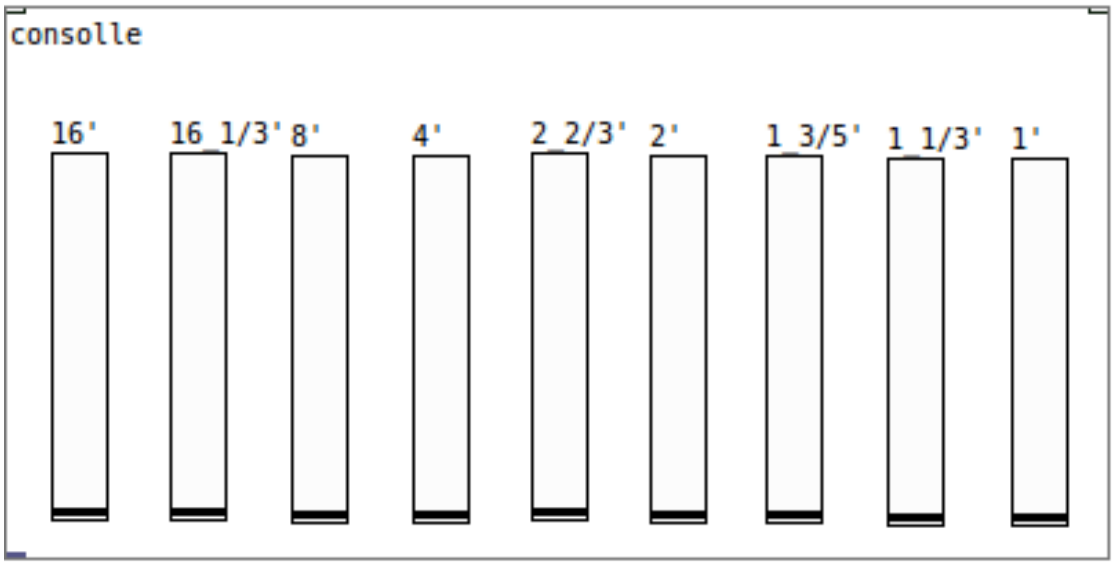
The **notein** object captures the signal of a MIDI keyboad and sends on its output the corresponding **note number** (left outlet), also called pitch, and **velocity** (right outlet).



The note number corresponds to the note played, while the velocity is its intensity. Velocity goes from 0 (note off) to 127. The number 1 indicates that we read just the first channel of the MIDI device.

THE CONSOLLE

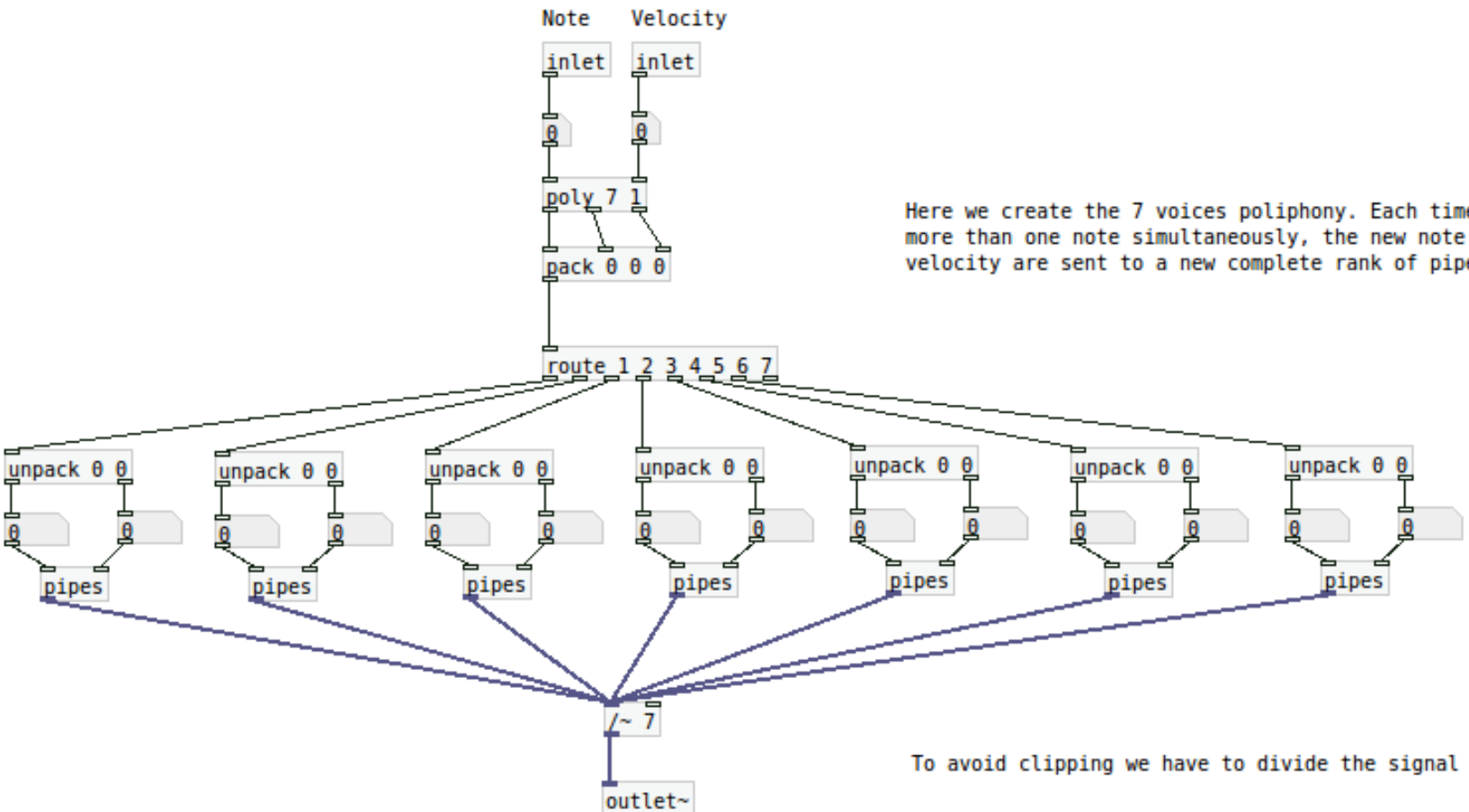
The note and velocity messages are sent to the **consolle** object.



The faders represent the amount of volume of the corresponding harmonics (we will see later in detail). On the outlet we have the pipe organ signal.

The consolle object is a canvas: let's have a look inside it.

THE CONSOLLE (INSIDE)

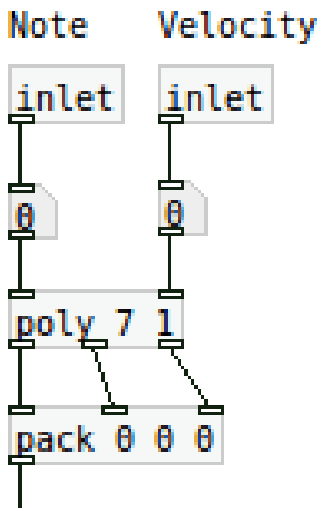


POLYPHONY

Polyphony allows to play more than one note simultaneously.
In this case we opted for a 7 voices polyphony.

The object **poly** receives
the pitch and the velocity and
returns the **voice number**,
the pitch and the velocity.

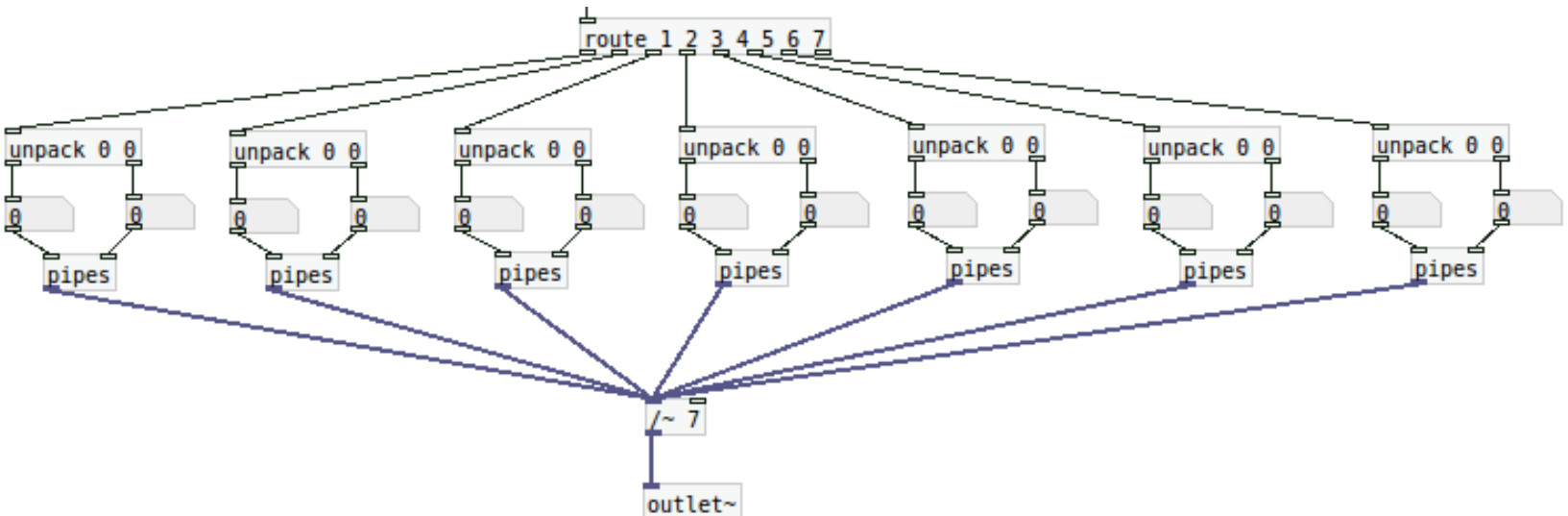
This stream is collected
by the object pack that
outputs a concatenated list.



POLYPHONY

The list composed by voice number, pitch and velocity is sent to the object **route** that routes the information according to the relative voice number.

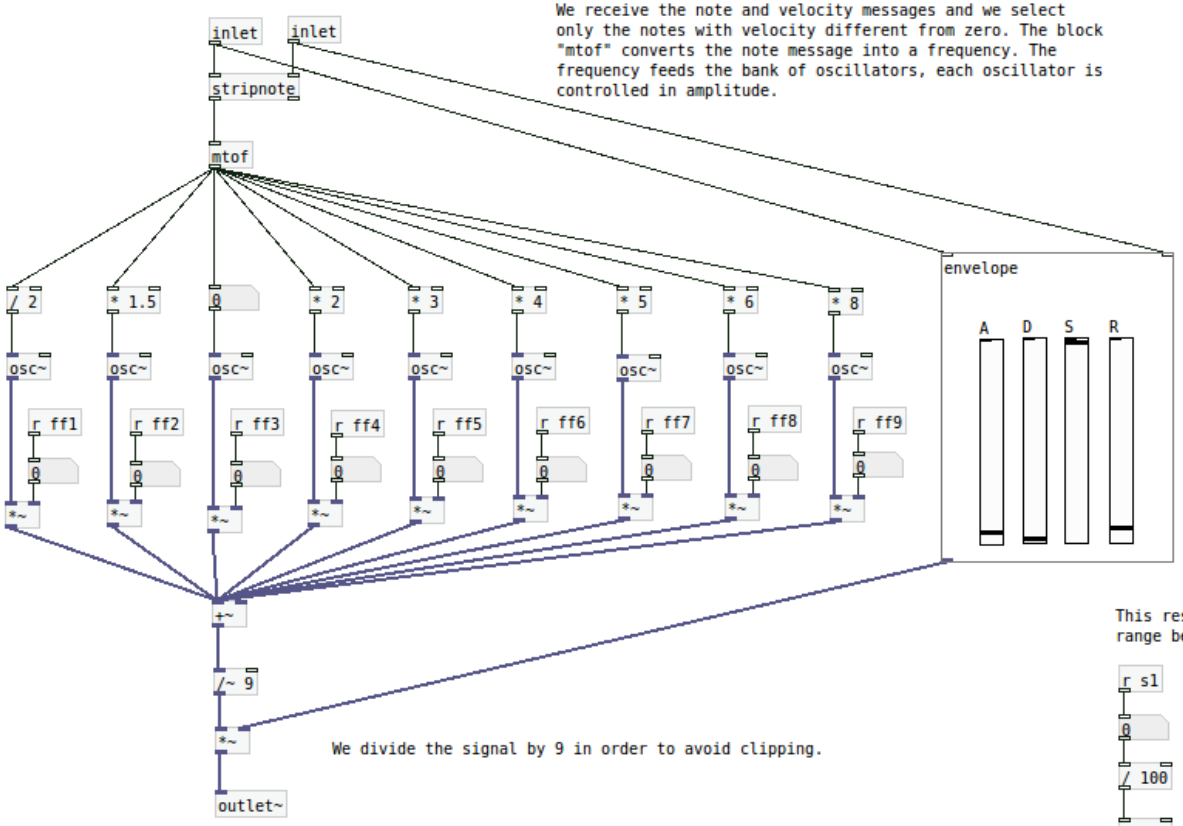
The streams are unpacked and sent to the object **pipes**.



The outputs of the pipes objects are summed together (the division by 7 avoids saturation of the DAC).

PIPES

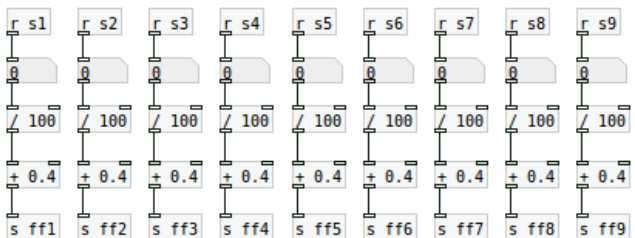
Here is where sound generation happens.



OSCILLATORS VALUES (from left to right): 16' one octave below the fundamental 5 1/3' one fifth above the fundamental 8' fundamental note 4' one octave above the fundamental 2 2/3' one octave and one fifth above the fundamental 2' two octaves above the fundamental 1 3/5' two octaves and a major third above the fundamental 1 1/3' two octaves and a fifth above the fundamental 1' three octaves above the fundamental N.B.: with fundamental we mean the input note

Note and velocity messages are sent to the envelope that returns the appropriate waveform, that has to be multiplied with the bank of oscillators.

This rescales the incoming values of the faders into a range between -1 and +1.



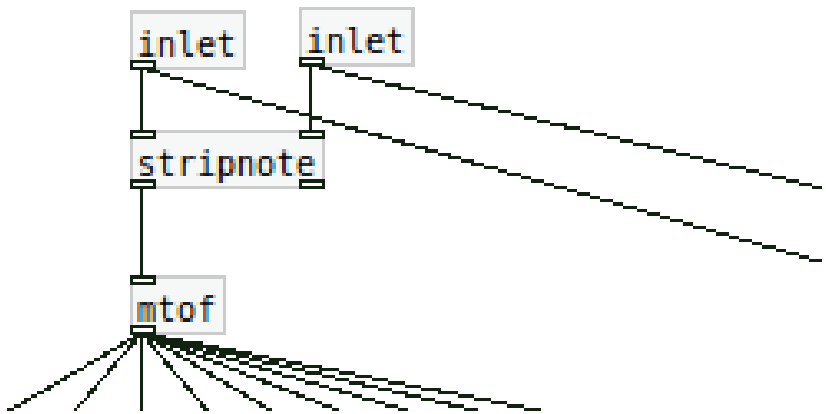
PIPES

Pitch and velocity messages enter the block stripnote, that outputs only the notes with velocity different from zero (that is the notes that are played!).

The block mtof **converts the pitch information into a frequency**.

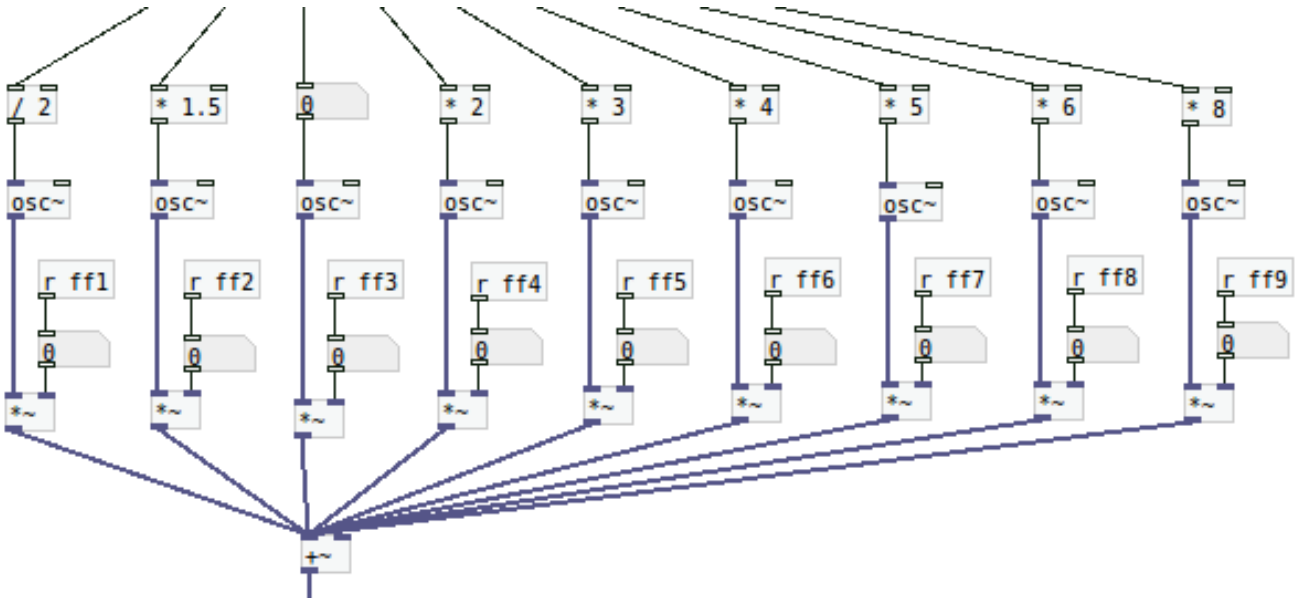
The note frequency
is sent to a bank of oscillators.

Note that the pitch and
velocity are sent also to the
envelope block.



OSCILLATOR BANK

The frequency signal reaches the **bank of oscillators**. The block **osc** outputs a cosine wave with the frequency corresponding to the input signal.



The sinusoids are multiplied by the value coming from the faders (in the consolle object) in order to set their amplitudes.
The outputs of the oscillators bank are summed together.

OSCILLATOR FREQUENCIES

As we have seen before, each organ pipe generates a sound that is composed by a series of harmonics. In order to reproduce a realistic sound, we use 9 components.

As the harmonic generation is due to physical vibration of bodies ($f = nv/2L$), the frequencies of the harmonics correspond to the **natural intervals**.

Don't make confusion: the instrument is tuned according to the **equal temperament**!

OSCILLATOR FREQUENCIES

Virtual pipe length	Ratio	Interval vs fundamental
16'	1/2	one octave below
5 & 1/3'	3/2	one fifth above
8'	1	FUNDAMENTAL NOTE
4'	4/2	one oct. above
2 & 2/3'	6/2	one oct. and one fifth above
2'	8/2	two oct. above
1 & 3/5'	10/2	two oct. and a major third above
1 & 1/3'	12/2	two oct. and one fifth above
1'	16/2	three oct. above

AN IMPORTANT OBSERVATION

By **combining the right quantity (volume) of the harmonics we are able to reproduce the sound (timbre) of any pipe**.

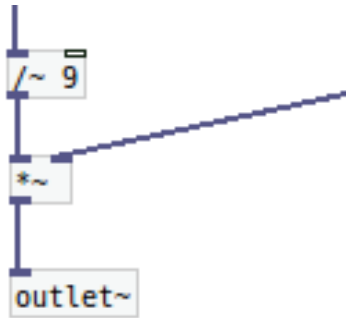
As the registers are linear combinations of pipes, we can achieve their sound still by properly combining the same harmonics!

In this way, **we can reproduce the sound of any register by properly setting just one bank of oscillators**!

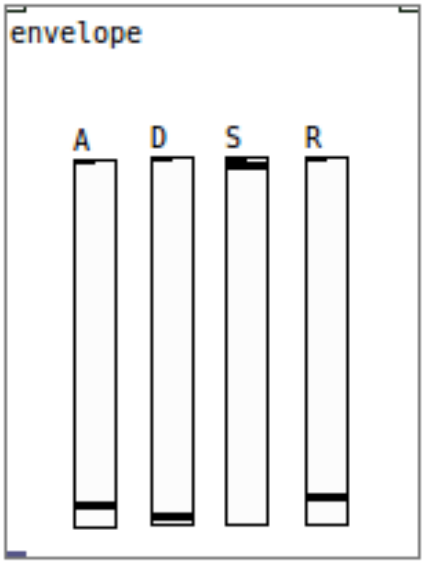
So there is no need to build one oscillator for each register, and this translates into efficiency.

OSCILLATOR OUTPUT

The sum signal is divided by the number of oscillators (9) in order to avoid clipping.

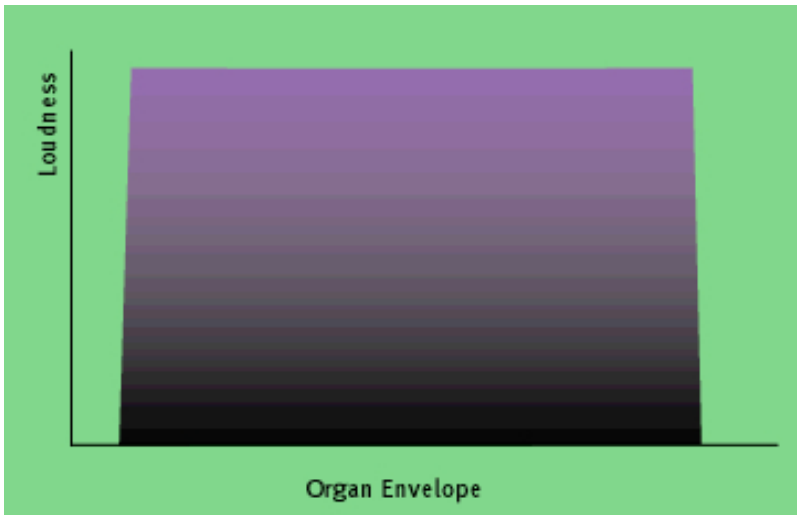


And then it is multiplied by the **envelope signal**.



THE ORGAN ENVELOPE

In the picture is shown the envelope of a pipe organ.

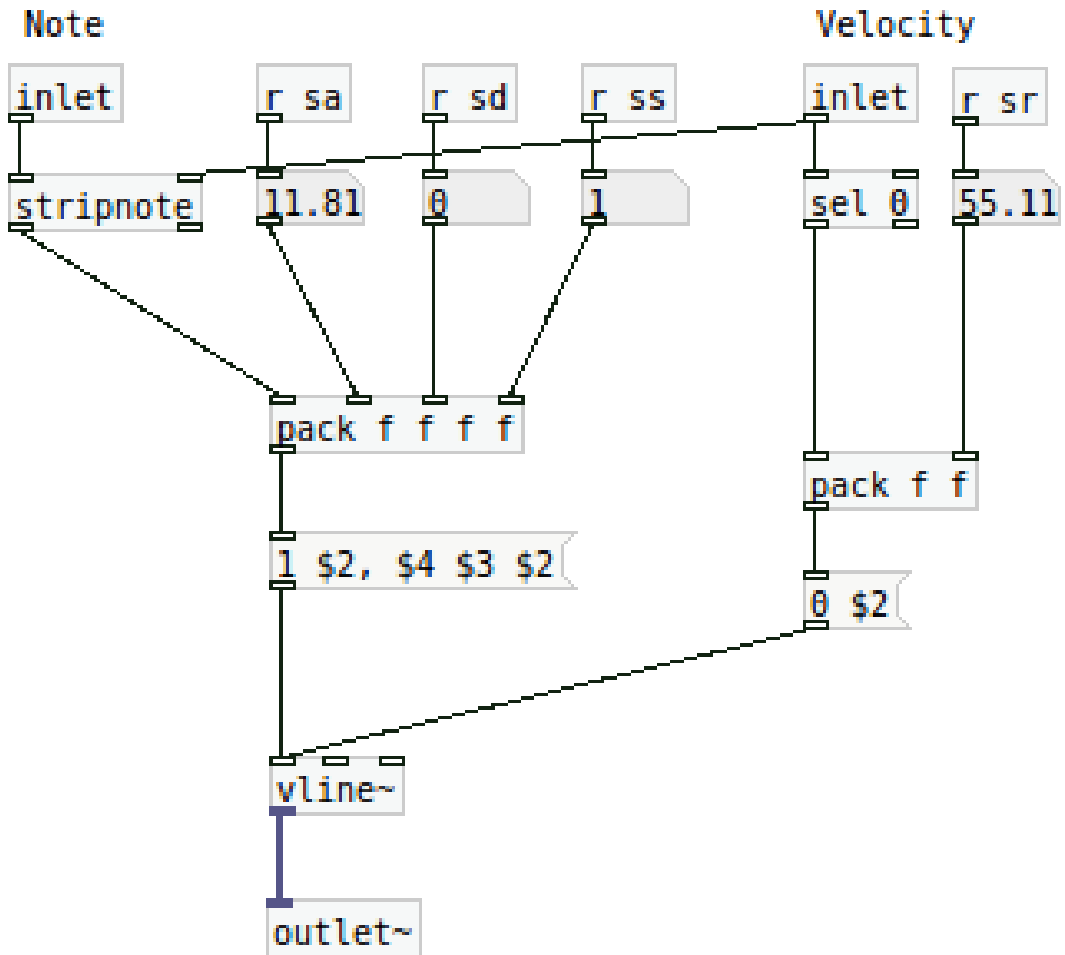


It has a **fast attack**, **no decay**, a **sustain** that depends on the duration of the note, and a **release** similar to the attack.

If we think at the sound process generation, we have an intuitive confirmation.

THE ENVELOPE GENERATOR

This is what we find inside the envelope generator:

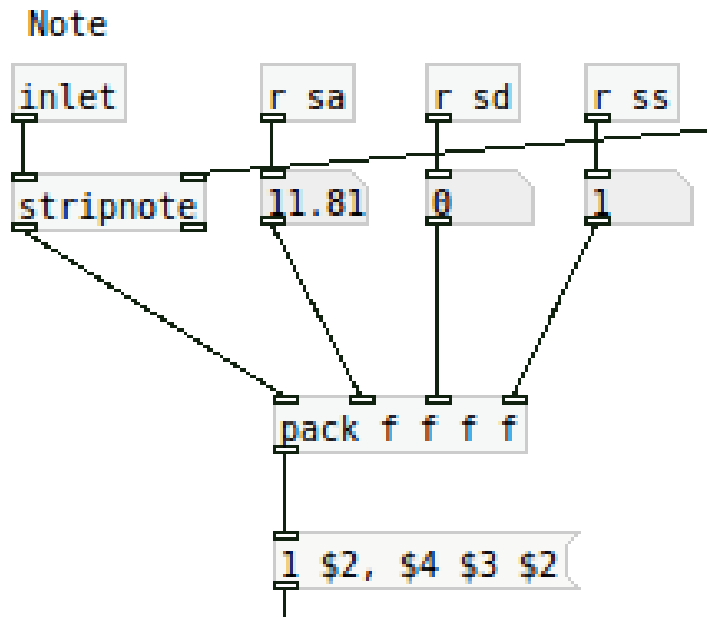


THE ENVELOPE GENERATOR (ATTACK-DECAY-SUSTAIN)

The attack-decay-sustain section checks if a note has been pressed and **draws a line** with the following rule:

go to **one** (max amplitude)
with an **attack time**
$2, then go to **sustain value**
$4 with a **decay time $3**,
after a **delay $2** (attack time).

In our case
we set the decay time to zero
and the sustain value to one.

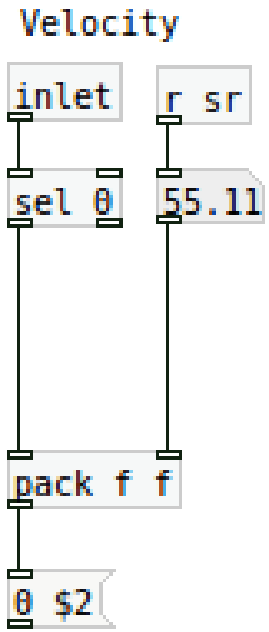


THE ENVELOPE GENERATOR (RELEASE)

The release section receives the note velocity (to check if the key is still pressed or not) and draws a line with the following rule:

when the velocity is zero go to **zero** within a **release time $2**.

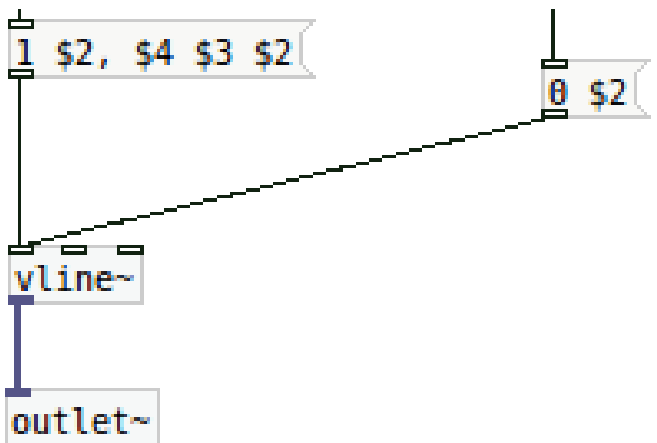
The
block sel 0 is used to trigger
the note release condition.



THE ENVELOPE OUTPUT

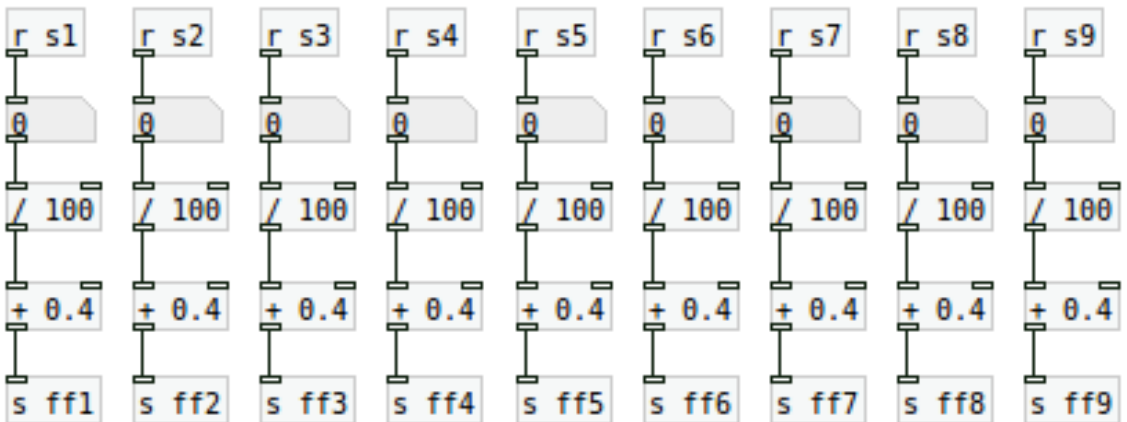
The attack-decay-sustain and release informations are used to draw the envelope waveform, that will multiply the signal coming from the oscillators bank.

Note that the parameters are controlled by the values received from the faders.



A NOTE ON THE OSCILLATORS VOLUME

We said that the faders control the oscillators volumes: actually, instead of driving the oscillators directly, they pass in this section that **rescales the combinations of the volumes between 0 and 1**.



Please take this as it is for now! We will see it in detail in a while...

EVERYTHING CLEAR?

Is everything clear for now?

I think it's time to make a quick summary...

QUICK SUMMARY

Until now we have seen that:

1. the MIDI signal is acquired and it enters the **consolle** block
2. inside the consolle block we manage a 7 voices **polyphony**
3. each voice goes to a **pipe** block
4. each pipe block contains a **bank of oscillators** and an **envelope generator**
5. all the oscillators and voices outputs are summed together

Easy, isn't it? :)

THE TREMOLO

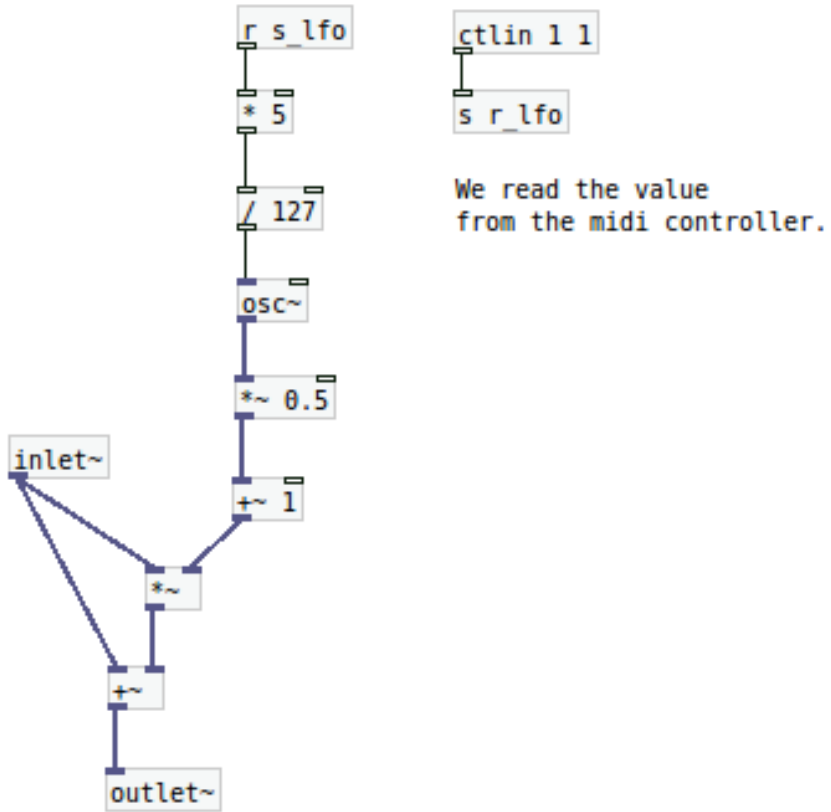
At the output of the block consolle, the signal goes into the block tremolo.

The tremolo receives a value from the keyboard wheel (that is between 0 and 127) that is normalized and multiplied by 5.

The maximum modulating frequency then is 5 Hz.

A sinusoid with amplitude 0.5 and shift of 1 is generated. This is called **low frequency oscillator** (LFO).

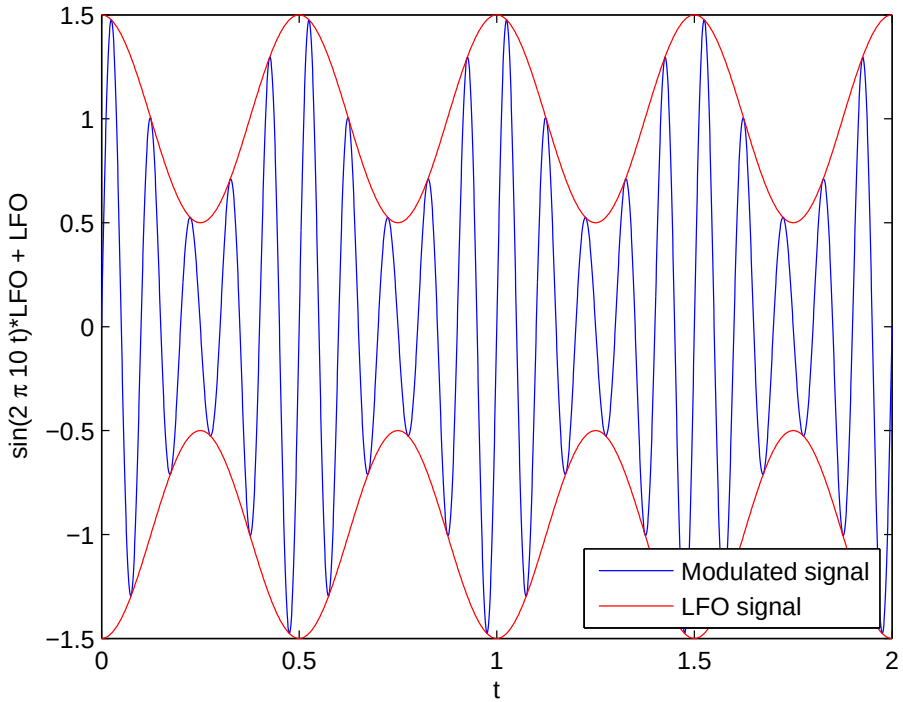
The modulating signal is multiplied with the organ signal.



THE TREMOLO (INSIDE)

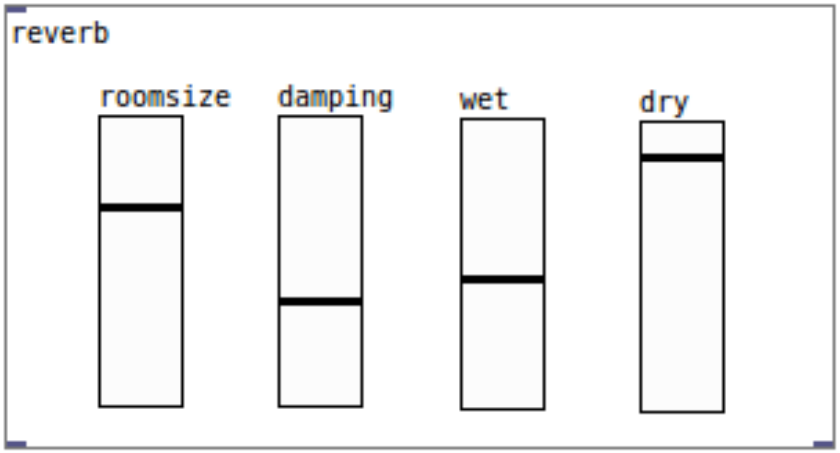
The organ signal is then summed to the modulated one.

In the following graph is shown an example of a sinusoid modulated by the LFO:



THE REVERB

The tremolo output reaches the reverb input.



The reverb is essential to give **realism** to the synthesized sound.

In fact churches **have a considerable reverberation**, usually with long decay times.

THE REVERB

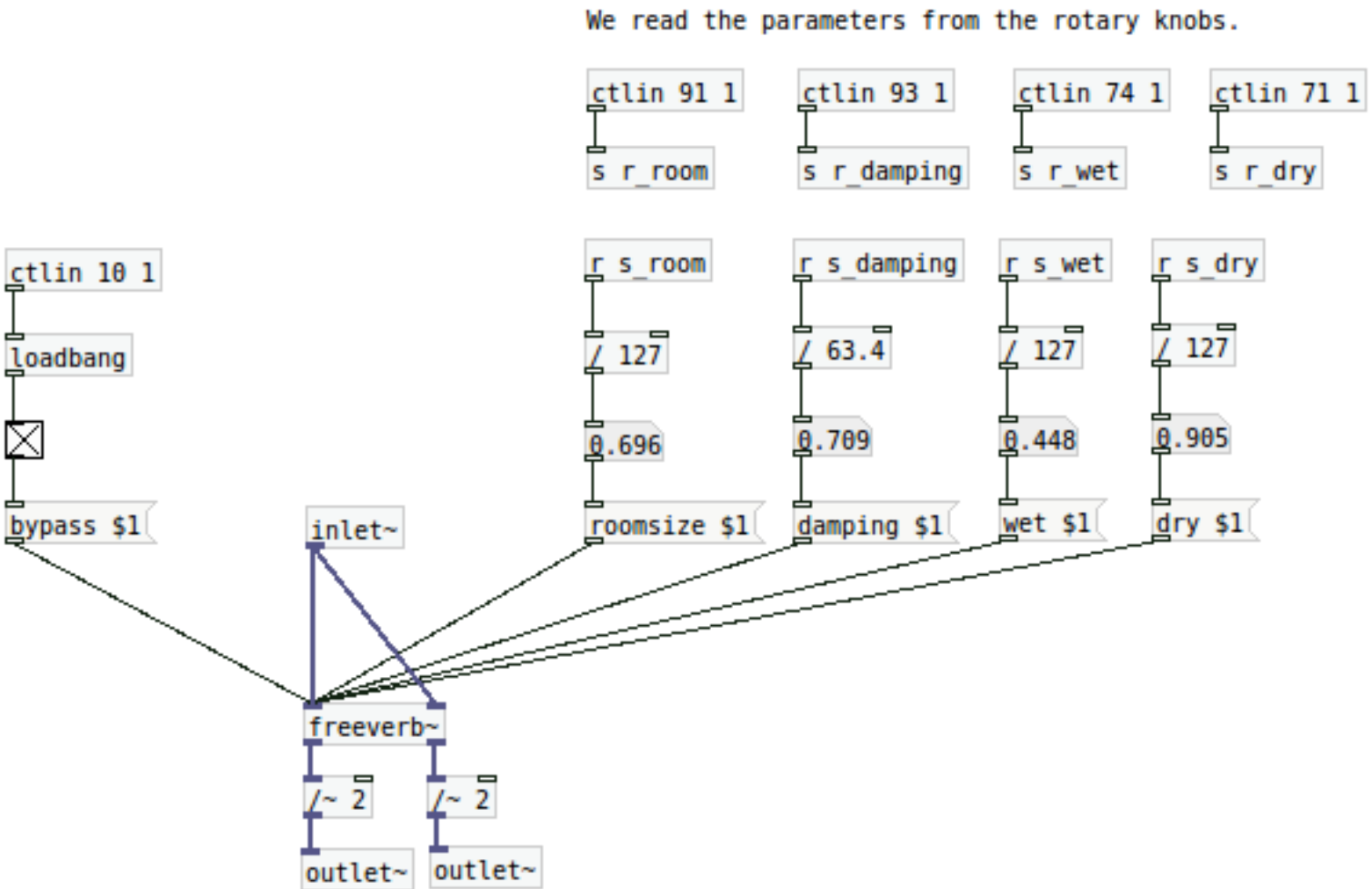
The object **freeverb** generates the reverb and comes with Pure Data. The reverb parameters are the following:

- ON/OFF
- Room
- Damping
- Wet
- Dry

We can control the reverb both by using the faders from the interface or by using the rotary knobs, and a button, on the MIDI keyboard.

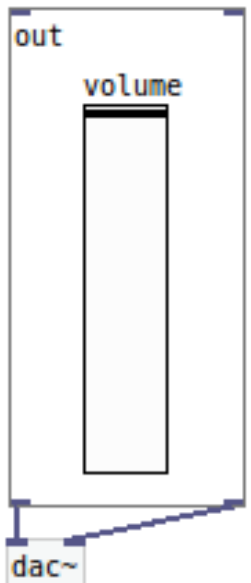
The reverb output is split into left and right output, although the input signal is monophonic, and both outputs are divided by two in order to avoid saturation.

THE REVERB (INSIDE)



THE OUTPUT

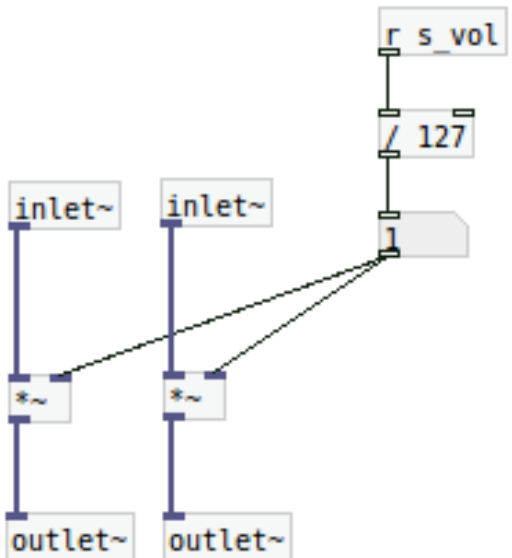
The reverb output goes into the volume control.



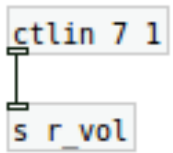
The output of this, ends into the digital-to-analog converter that sends the signal to the soundcard output.

THE OUTPUT (INSIDE)

The volume can be controlled both from the volume fader and from the MIDI keyboard.



Volume value

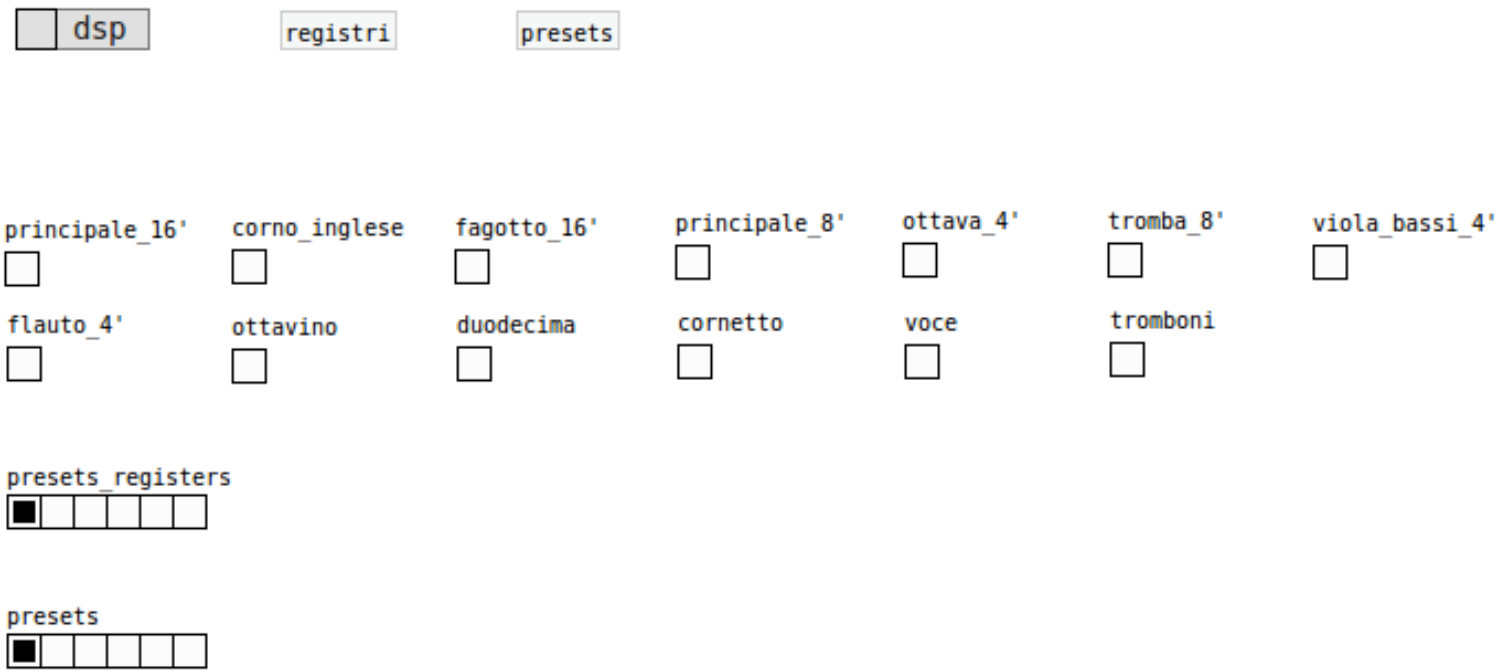


The two inlets are the left and right channels. These are multiplied by a value that varies from 0 to 1, controlled by the volume fader.

Both outputs are multiplied by the normalized volume value (attenuation).

REGISTERS AND PRESETS

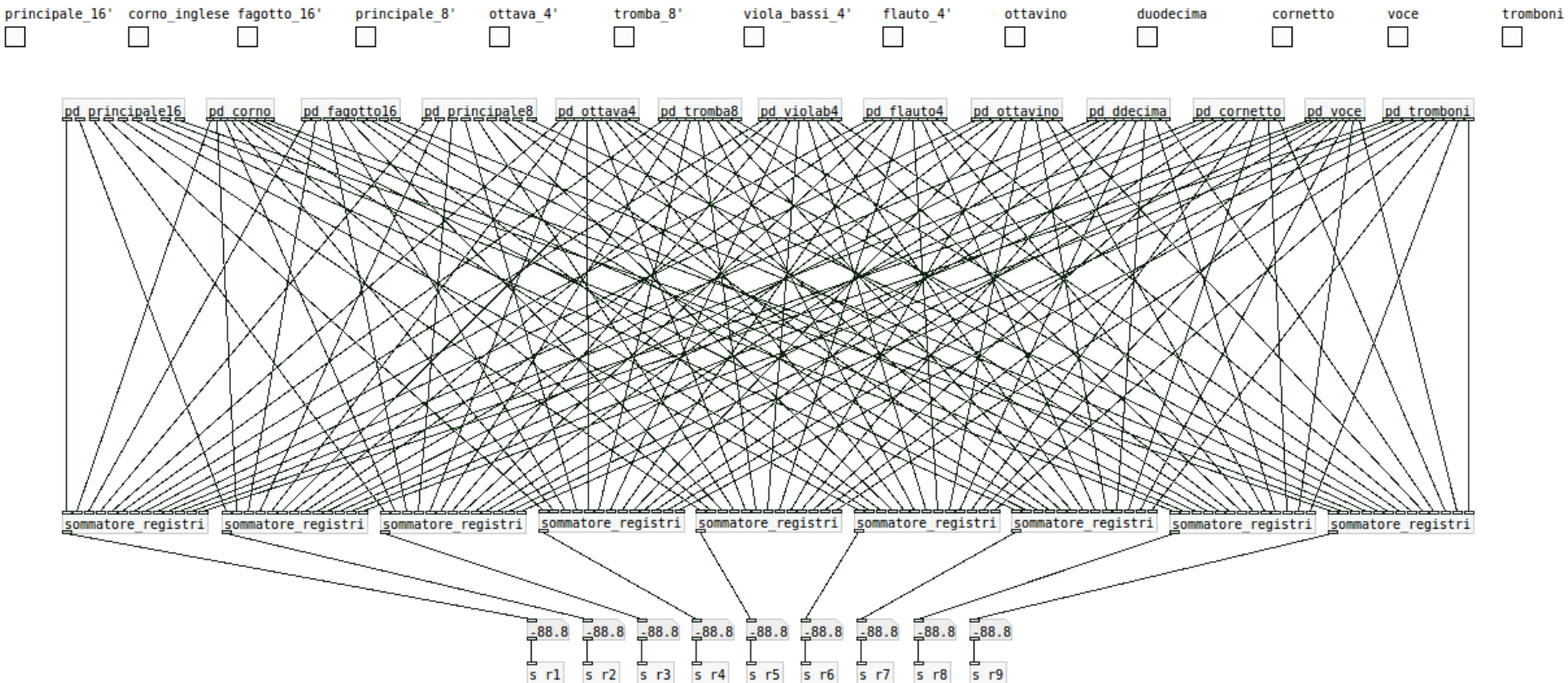
We need to check the last part of the project: how registers and presets are managed.



That is more tricky than what it seems...

REGISTERS

This is the content of the block **registri**.



REGISTERS (ONE STEP BEHIND)

Why so messy?

The register management needs a bit of attention.

Each register has a corresponding sequence of volume values (of the consolle faders).

Registers can be combined, that is, it is possible to activate **more than once at time**.

How this translate into volume variations?

In other words, activating two registers with settings **x** and **y**, how is the setting of the combination **x + y**?

REGISTERS (ONE STEP BEHIND)

If we were just going to sum the corresponding volumes for each harmonic, this wouldn't be correct.

Let's make an example.

Suppose that the registers are assigned the following settings:

x=[0 0 7 2 5 3 1 0 1];

y=[3 4 5 2 1 8 1 0 0];

And suppose also that their range goes from 0 (minimum value) to 10 (maximum value).

By algebraically summing the two registers we would obtain:

x+y=[3 4 **12** 4 6 **11** 2 0 1];

REGISTERS (ONE STEP BEHIND)

As we can notice two of the values are already **out of scale**!

In this way, by summing a few registers, we would saturate immediately all the volume values of the harmonics.

Clearly this is not the right approach...

To understand how to manage this problem we have to dig into the physic of the pipe organ.

REGISTERS (THE TRUE STORY)

In an organ pipe, when two registers are activated simultaneously, the effect is that more ranks of pipes sound together and we will perceive a higher volume.

What really changes is the overall **sound intensity level**.

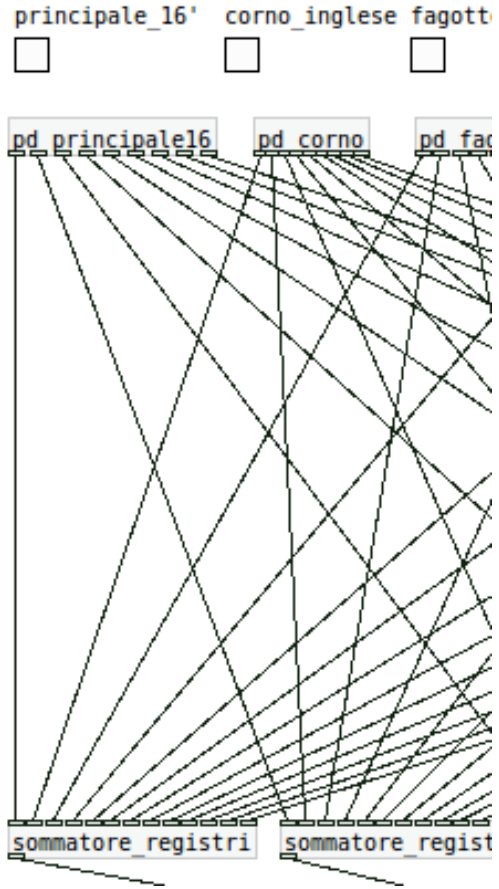
Hence, what happens is an **energetic sum** between the levels generated by the two registers, that is:

$$L_1 \oplus L_2 = 10 \log_{10} \left(10^{\frac{L_1}{10}} + 10^{\frac{L_2}{10}} \right) [dB]$$

So, in order to reproduce the natural working principle of the registers, **we have to perform logarithmic additions between the registers values**.

REGISTERS

Let's go back to the register block:



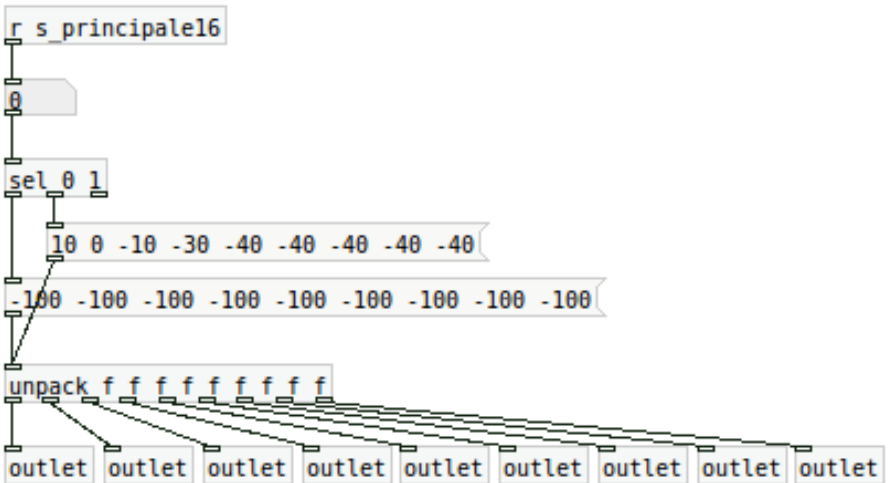
The buttons (toggles) above need to turn the corresponding register ON and OFF.

Each register is assigned a **one-off subpatch** that contains the register values. At their outputs we have the fader values that will go into the block **sommatore registri**.

Each of them sums all the corresponding harmonics of the registers.

REGISTERS (ONE-OFF SUBPATCH)

This is what we find inside the one-off subpatch **pd principale 16**.



According to the toggle value, the corresponding message is selected.

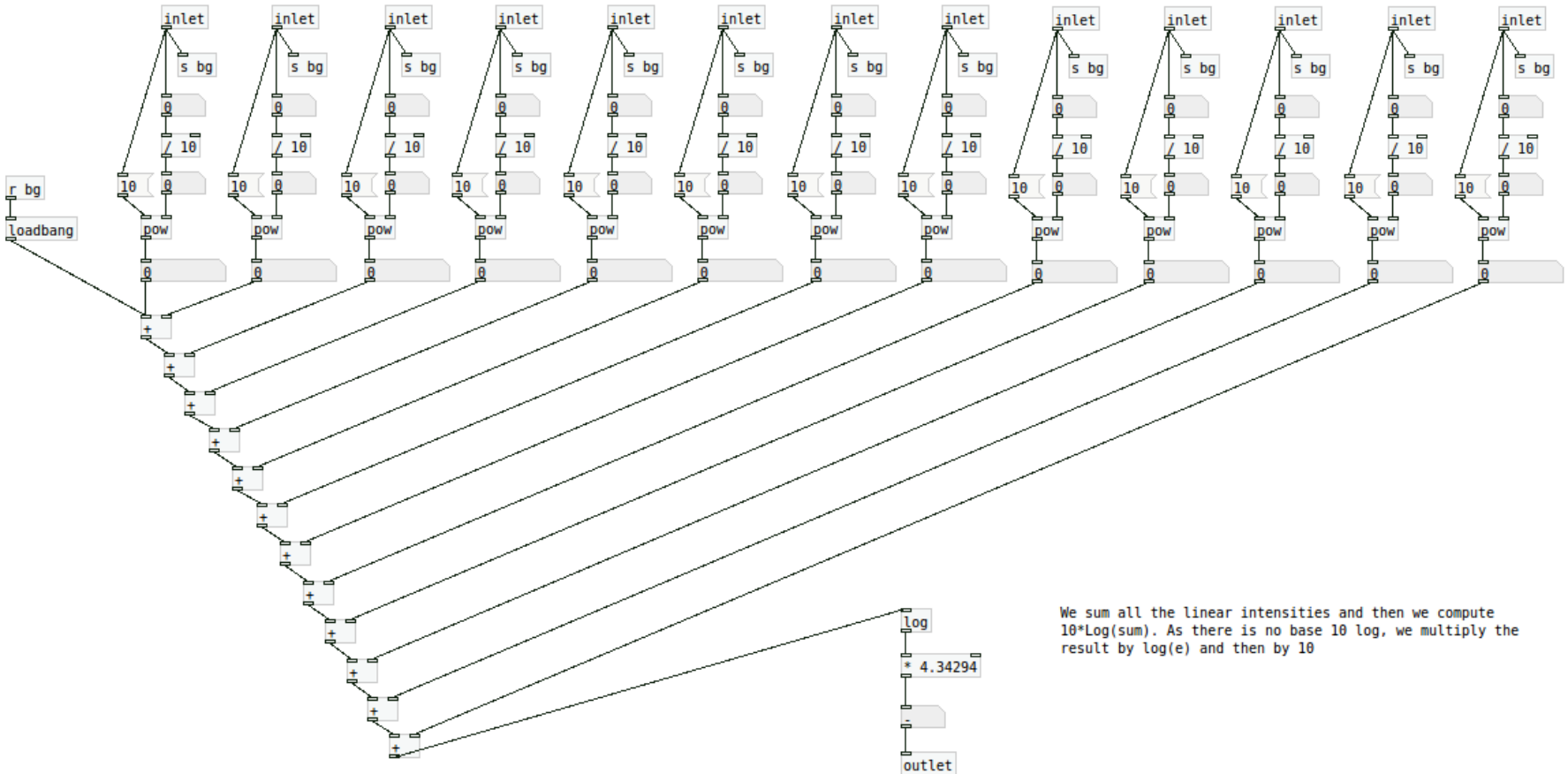
When the register is turned off it is selected a message composed of attenuation values -100 dB.

Otherwise it is selected the message that contains the volumes faders, expressed in decibels.

Notice that the volume range goes from -40 dB, to 60 dB.

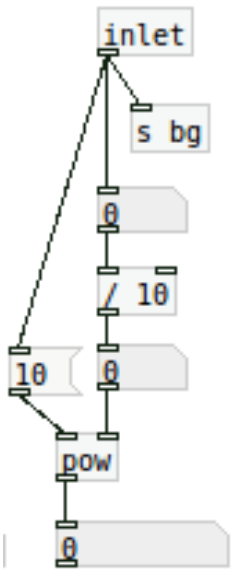
REGISTERS (SOMMATORE REGISTRI)

The register outputs go into the block **sommatore registri**.



REGISTERS (SOMMATORE REGISTRI)

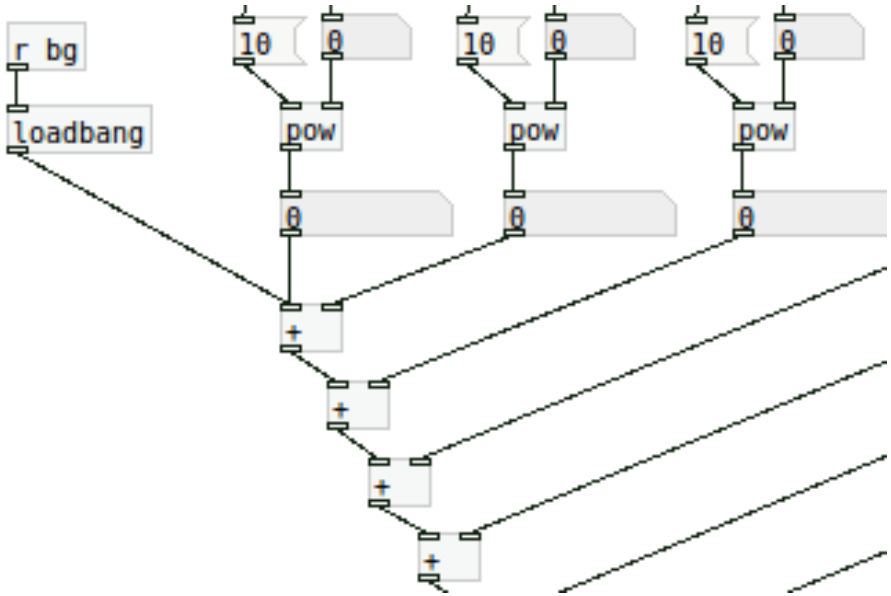
Each of the following blocks performs the operation $10^{L_i/10}$.
There are as many as the number of the registers.



Notice that each time an input is changed, the **bg** signal is sent.

REGISTERS (SOMMATORE REGISTRI)

The **bg** signal is used to trigger the first sum operation (and consequently all the others).

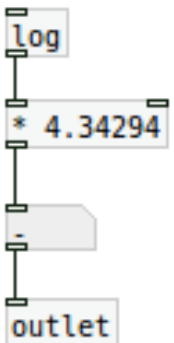


This is because every time any registers output is changed, the operation must be recomputed (remember that Pure Data triggers the operation with the hot inlet).

REGISTERS (SOMMATORE REGISTRI)

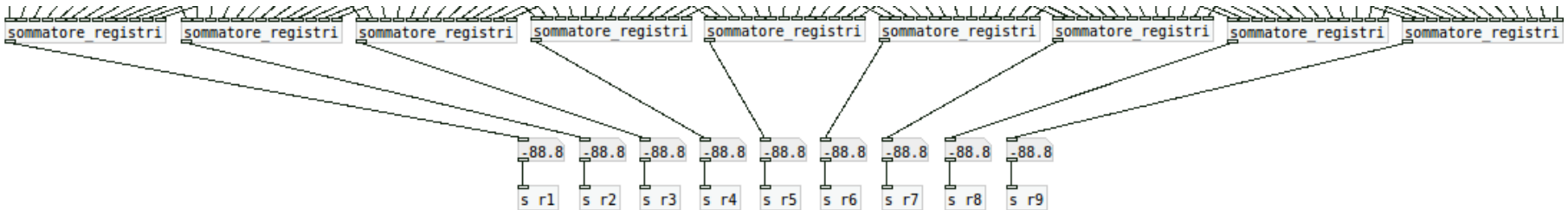
The sum signal $\sum_i 10^{L_i/10}$ must be converted in dB again, that is we must perform the operation $10 \log_{10}(\bullet)$.

As Pure Data doesn't have the base 10 logarithm, we multiply the result by $\log(e)$ and then by 10, that is 4.34294.

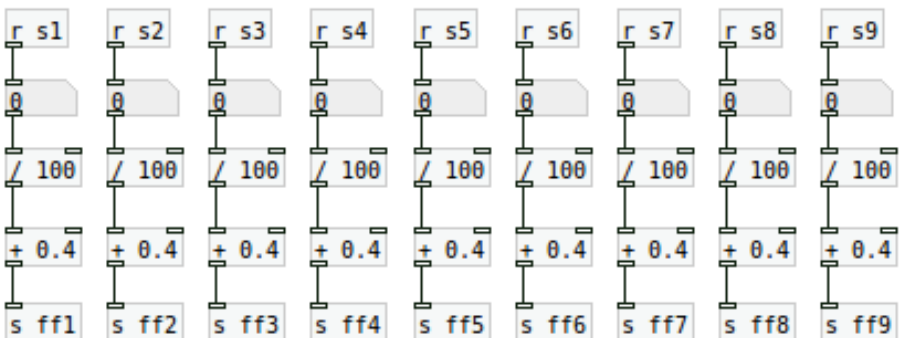


REGISTERS (SOMMATORE REGISTRI)

So, the **sommatore registri** blocks are 9, one for each harmonic.



Before reaching the oscillator bank, they are scaled between by the block we saw before.



REGISTERS

In practice, we want to **map the volumes in dB**, that go from -40 dB to 60 dB, **into a set of values that goes from 0 to 1** (remember the saturation of the audio signal!).

This corresponds to the line that passes for the points (-40; 0) and (60; 1), that is:

$$y = x/100 + 0.4$$

Finally, the correct amount of volume, that comes from the combination of the registers, multiply each sinusoid of the oscillators bank.

PRESETS

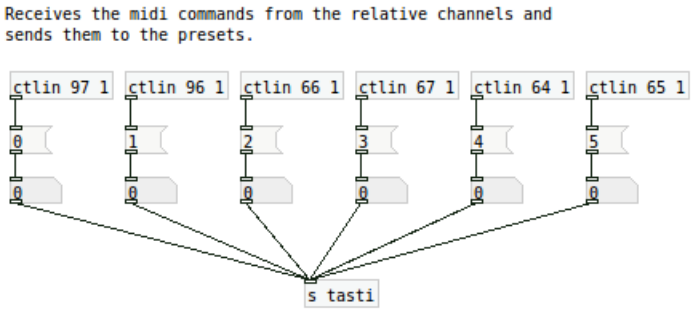
The last block we need to analyze is the **presets** block.

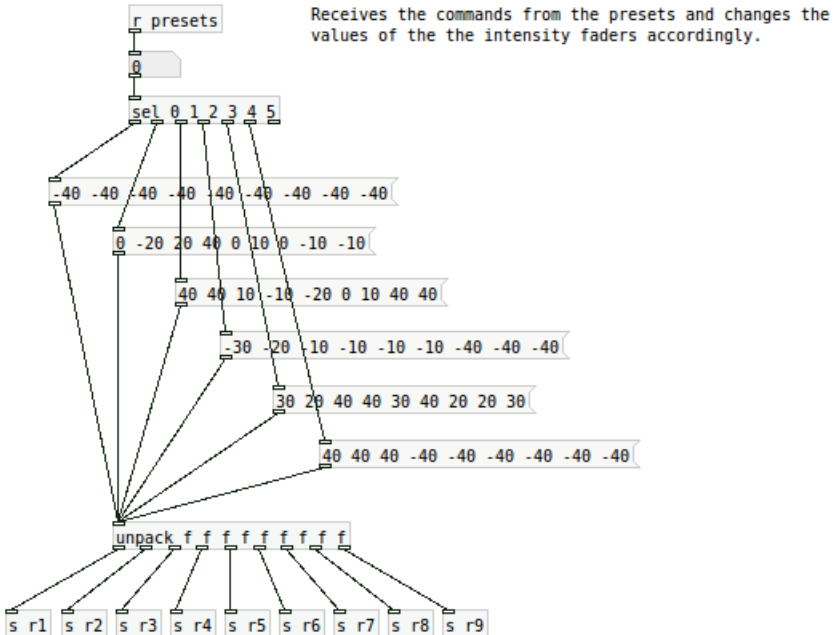
It allows to switch rapidly among different preset sounds.

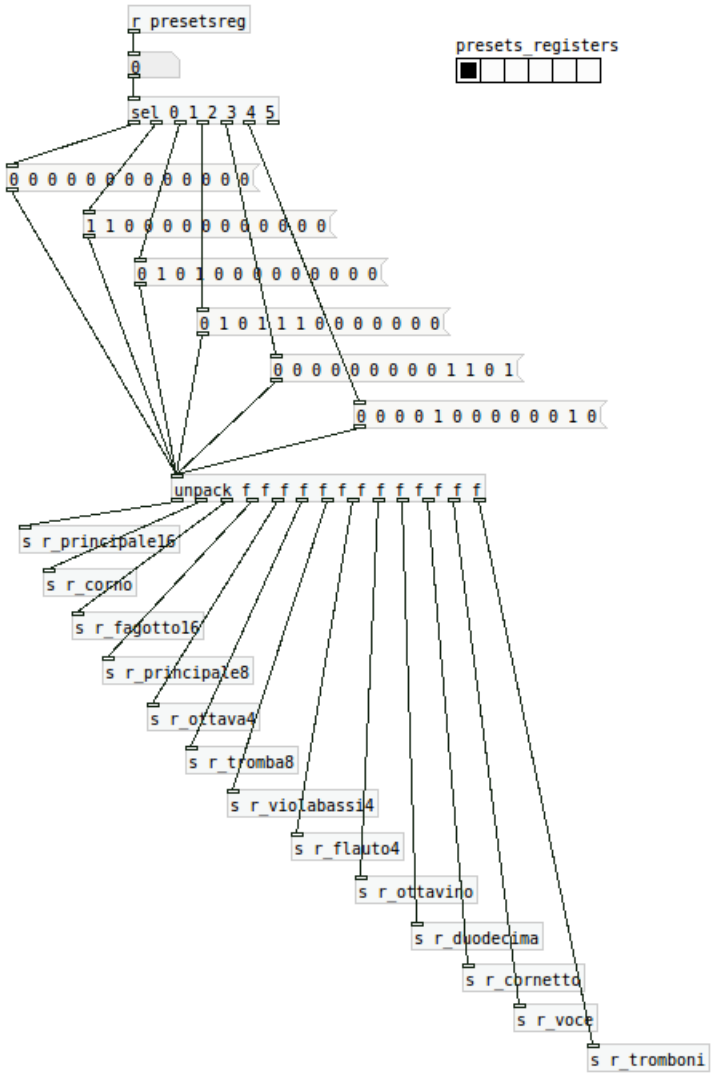
We implemented two kind of presets:

1. Registers presets: they send to the registers toggles the desired combination of ON/OFF values to activate and deactivate the registers.
2. Presets: they receive the preset number message from the MIDI keyboard and they send the corresponding volumes to the faders.

PRESETS (INSIDE)







THE TEST

That's all, now let's check how it sounds!

We will use a MIDI keyboard that integrates a MIDI USB interface.

The integrated PC soundcard is not thought to be used for these kinds of applications, moreover, its sound quality is not that good. For this we will use an external USB soundcard.

A note: while playing you could hear some pops and cracks. This is not a bug of the code, but it is due to the Linux kernel. In fact my version of Ubuntu doesn't run a low-latency kernel (it doesn't give the maximum priority to the audio processes), and it can happen that sometimes some audio data packets are lost/delayed.

REFERENCES

To create this project we used the following sources (in order of importance):

1. A bit of fancy :)
2. L'ORGANO GIOVANNI TONOLI, Consiglio Pastorale Parrocchiale di Carzago della Riviera (BS)
3. THE DEVELOPMENT OF THE PIPELESS ORGAN, G. T. WINCH, 1940
4. http://keyboardservice.com/Drawbars.asp
5. CANNE D'ORGANO AD ANIMA, articolo storico, Patrizio Barbieri
6. COMPUTATIONALLY EFFICIENT HAMMOND ORGAN SYNTHESIS, J. Pekonen, T. Pihlajamaki, V. Valimaki, 2011

And for the curious, here you find the biggest organ database:

http://intorg.org