Abstraction of a real-time granulation system built into the Max/MSP environment

Stefano Alessandretti

***Abstract*— Since Barry Truax's early pioneering work, the increasing availability of powerful and inexpensive desktop computers has led many composers, researchers and artists to experiment and work with granular synthesis and/or granulation processes. During the last two decades a stream of granulation software has appeared, but despite this spread, all these applications have important limitations in terms of efficiency, usability, flexibility and control. This paper describes an abstraction of an efficient and flexible granular processing system built into the Max/MSP environment.**

***Index Terms*— Abstractions, Digital Sound Processing, Max/MSP, Signal Processing.**

I. Introduction

Although there are many sound granulation algorithms that have spread widely since the early experiments of the Canadian composer Barry Truax [1]-[2]-[3], it is not possible to find a system that does not show at least one of the following limitations (2-3 in most cases):

- closed architecture [4]-[5]-[7]-[9],
- with fee [5]-[7],
- no real-time support [6],
- pre-recorded samples only [7],
- implementation with a large use of externals [8],
- C or C++ programming skills needed for the algorithm development [9]-[10].

The abstraction presented in this paper has the aim to overcome all these limitations by implementing a real-time granulation system into the Max/MSP environment [11].

I started the development from a basic algorithm that I had previously implemented in Max/MSP and that I used in my piece *Krònos/Kairòs, for flute, harp and live electronics (2010-11)* [12].

The algorithm described in the abstraction has the following features:

- from 1 to *N* voices - depending on the CPU,
- dynamic pitch transposition for each voice,
- dynamic amplitude for each voice,
- dynamic playback (portion of file) for each voice,
- dynamic phase for each voice.

II. Overview

At the highest level the system is organized into 3 main patches: *Buffers patch*, *Data controls patch* and *Poly patch* (Fig. 1).

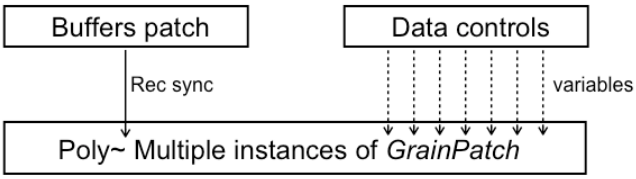

Fig. 1. System overview.

The first 2 patches generate the overall audio and control signals, while the third is a set of multiple instances of the *Grainpatch* (each instance processes only the data addressed to it). The number of instances depends on the number of grains - or voices - you want to generate and it is clearly proportional to the computing power of the processor.

III. Buffers patch

The *Buffers patch* is the algorithm section where the processes of *loop recording* and *window function* generation take place; in other words, it is the patch where the audio data are stored into the memory buffers.

A. Loop recording

In order to enable *loop recording* and audio data playback from any points of the buffer, the system has been based on the use of two parallels recorders. These recorders are identical in size (30000 ms), but they have an offset of half of their length between their recording points (recording heads, Fig. 2).

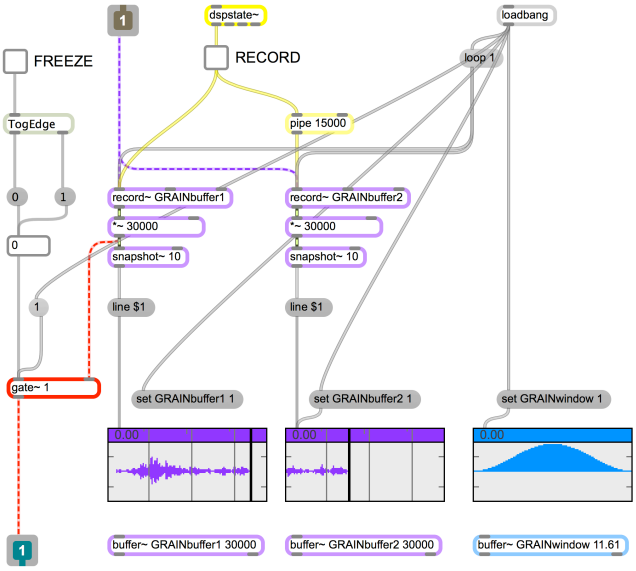

Fig. 2. *Buffers patch*.

Manuscript received January 8, 2015.

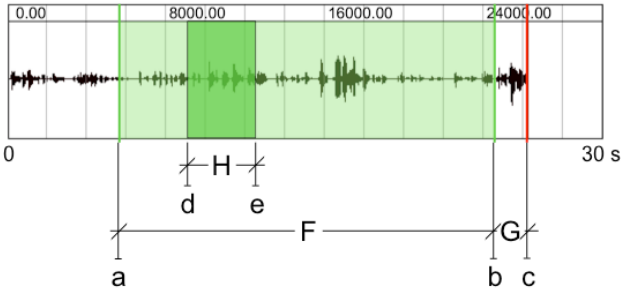

Fig. 3. Audio files variables. (*a*) minimum starting point. (*b*) maximum starting point or playback pointer. (*c*) recording pointer. (*d*) effective starting point after randomization. (*e*) effective ending point. (*F*) starting range. (*G*) cartridge delay. (*H*) file portion played.

This is a fundamental operation that allows the reproduction of those grains that reside in the memory limit of a buffer; a task not admitted by the Max/MSP players. For example, it would be impossible to reproduce a grain of 100 ms with the starting location at 29980 ms and the ending location at 80 ms of the first buffer, so you need to access to it from the second buffer (delayed) where the starting location is at 14980 ms while the ending location at 15080 ms.

B. *Window function generation*

The third memory buffer inside the patch is reserved to the *window function* storage. You can generate the function and store it into the buffer or simply recall a saved one from the hard drive.

C. *Data flow*

The audio signal from the input is recorded into the identical buffers (30000 ms), *GRAINbuffer1* (in sync) and *GRAINbuffer2* (delayed by 15000 ms in location points); the recording is in loop mode and the start is automated by the DSP state.

The pointer (the clock of the recording head) of the first recorder is passed through a gate - to allow the *freezing* - and then addressed to the output.

IV. *Data controls patch*

Inside the *Data controls patch* the variables are set with faders, tables or preset numbers and then addressed to the corresponding output and to each grain (voice) inside the *poly~* object.

There are 7 types of variables sent to each voice: *start range, cartridge delay*, *metro*, *metro delay*, *grains length*, *amplitude* and *pitch-shifting ratio*.

A. *Start range*

It is the range of the buffer memory where there are the starting points of a voice, or in other words, the portion of time where you put the random starting point of each grain (Fig. 3). If you set it to zero, there is not randomization and the effective starting point (*d*) matches with the playback pointer (*b*); in this case there is only the *cartridge delay* (*G*) between the recording pointer (*c*) and the playback pointer (*b*).

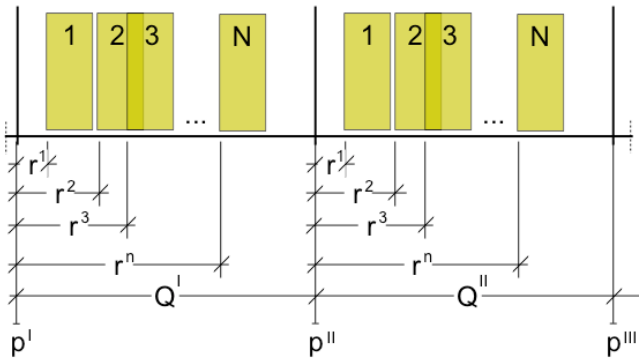

Fig. 4 Time variables. (*1*... *N*) voice number. (p^I... p^N) metro pulses. (r^I... r^n) metro delay for each voice. (Q^I... Q^N) period or pattern length.

B. *Cartridge delay*

It is the delay time between the playback pointer (*b*) and the recording pointer (*d*). It should be set according to the *input/output vector size*, *the signal vector size* or to aesthetic choices.

C. *Metro*

Like many other types of processing, granulation takes effect if managed in time with the best rhythmic detail; for this reason I have decided to organize grains as the repetition of a musical pattern (Fig. 4). In this way, *Metro* (p) is the trigger signal that represents the beginning of each pattern and which occurs in each time period (*Q*).

D. *Metro delay*

It is the delay time (*r*) between the metro pulses (*p*) and the starting point of a grain (d). For example, if you set the *metro delay* to zero for any voice, the grains are played simultaneously (homo-rhythmic grains).

E. *Grains length*

It is obviously the length of the grains, the only variable shared by all voices; it could be from 1 ms to *x*, where x is the buffer size minus the *cartridge delay* (*G*).

F. *Amplitude*

It is clearly the amplitude (in dB) of each grain.

G. *Pitch-shifting ratio*

It is the ratio of a grain pitch to its original pitch. For example, an octave upper shifting is equivalent to a *pitch-shifting ratio* of 2.

V. Poly~

As described above, a defined number of *Grainpatch* instances is loaded into the Poly~ object; this is the most important section of the system and represents the real core of the algorithm (Fig. 5). *Grainpatch* has 8 inputs, one for the recording pointer (c) and 7 for the variables.

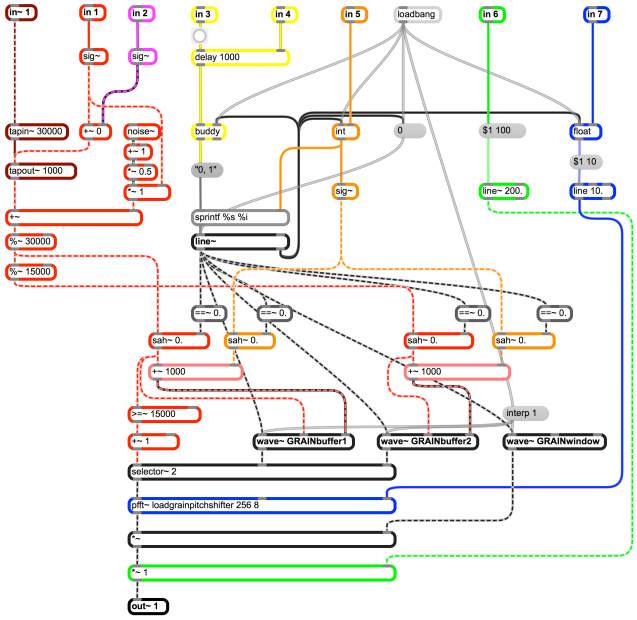

Fig. 5. *Grainpatch*.

Inputs are ordered as follows:

1) *Sync* signal from the recording pointer (c).
2) *Start range* value.
3) *Cartridge delay* value.
4) *Metro* triggers (bangs).
5) *Metro delay* value.
6) *Grains length* value.
7) *Amplitude* value.
8) *Pitch-shifting ratio* value.

A. *Starting point calculation*

This objects collection is arranged to calculate the effective starting point (Fig. 6). The data flow path is indicated below.

- The delay time of *Sync* is calculated adding the *start range* value to the *cartridge delay* value, $a = F + G$ (Fig. 3).
- *Sync* is added to a random value in the range 0-*F*, $d = a$ + random value [0-*F*] (Fig. 3).
- Modulo operators are applied to *Sync* to match the buffers length.

B. *Grain occurrences*

This objects collection is disposed to trigger the grain playing with a ramp function; the reproduction is done coupling the line~ object and the wave~ objects (Fig. 7-8). The data flow path is listed below.

- Metro bang is delayed (*r* value, Fig. 4).
- The delayed bang triggers the message when the line object is not working (bang from the right outlet of the line~ object).
- *Grains length* value is triggered as above (bang from the right outlet of the line object).
- The sprintf object sends the complete message to the line~ object.
- The ramp function is generated by the line~ object and sent to the wave~ objects and operators (==~).

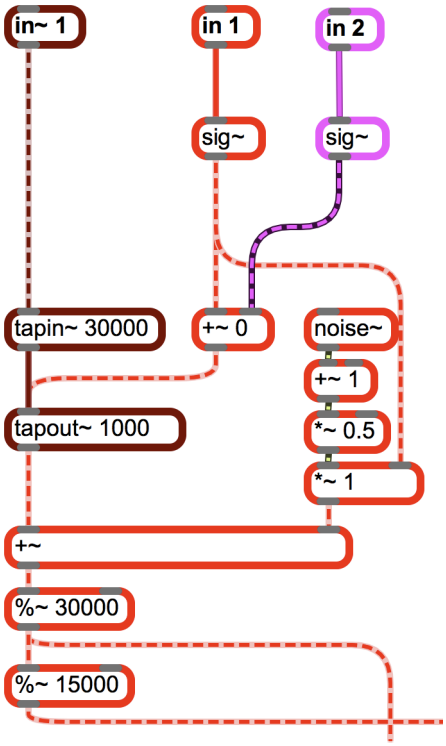

Fig. 6. Start point calculation.

C. *Grain playing*

This objects collection is set to allow the grain playing (Fig. 8). As mentioned above, the system uses 2 parallel memory buffers (see § III) and, as a result, 2 parallel grain players. The data flow path is listed below.

- The ramp function triggers the *Sync* values (red patch cords from the modulo operators) anytime it goes to zero.
- The starting points are added to the grain lengths and sent to the corresponding wave~ object.

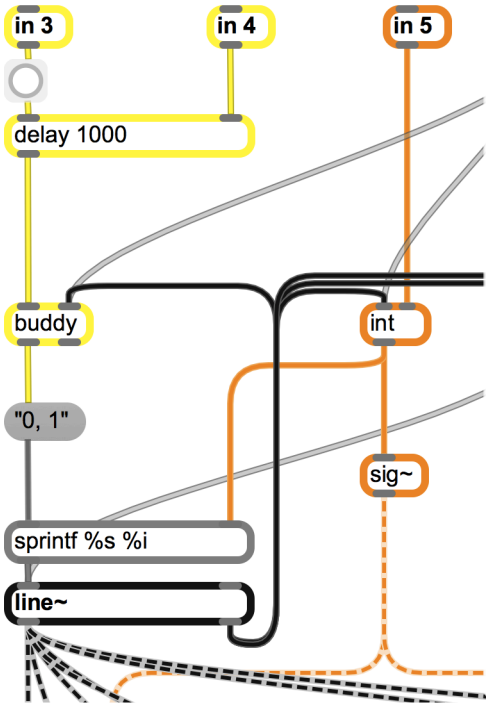

Fig. 7. Grain occurrences.

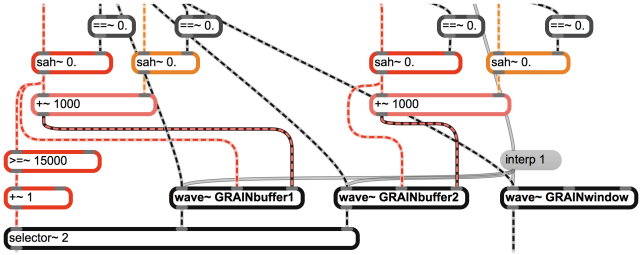

Fig. 8. Grain playing.

- While the wave~ objects are playing, a selector driven by an operator (equal or greater then 15000) decides which wave~ object signal passes through it (see § III).
- The selected signal is sent to the pitch-shifter module.

D. Amplitude and pitch-shifting ratio interpolation

This simply objects collection is arranged to interpolate the amplitude and the pitch-shifting ratio (Fig. 9). The data flow path is as follow.

- The amplitude value is passed through a line~ object and interpolated in a time period of 200 ms (this is the only variable not synced with a zero value of the ramp).
- The pitch-shifting ratio is synced with the ramp then passed through a line object and interpolated in a time period of 10 ms.
- The pitch-shifting ratio is sent to the corresponding module.
- The amplitude value is sent to a multiplier object.

E. Amplitude scaling and pitch-shifting

This final objects collection is set to apply pitch-shifting, windowing and amplitude scaling (Fig. 10). The data flow path is listed below.

- The signal is passed through a frequency-domain pitch-shifter based on the gizmo~ object.
- The pitch-shifted signal is multiplied by a synced window function.

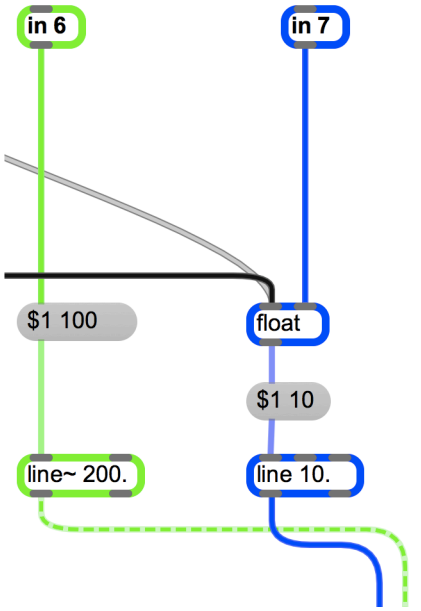

Fig. 9. Amplitude and pitch-shifting ratio interpolation.

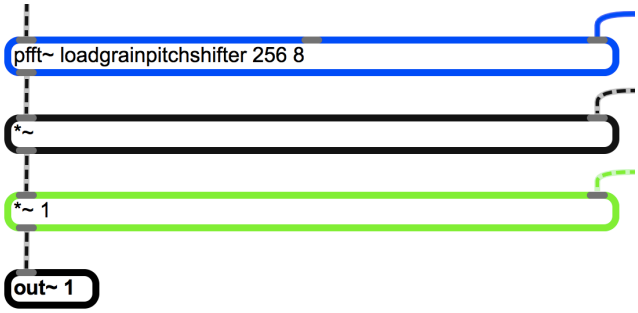

Fig. 10. Amplitude scaling and pitch-shifting.

- The windowed signal amplitude is scaled.
- The result is sent to the output where grains are added together by the poly~ object.

VI. Algorithm Performance

The algorithm performance is strictly related to its overall variables setting (obviously to the audio scheduling settings too); in this way the most critical part of the system is represented by the real-time pitch-shifting process and its artifacts production [13]-[14]. For this reason, it is really important to set the pitch-shifter module variables (window size and overlap factor in particular) considering the grains length and the grains spectral content (Fig. 11).

On the other hand, it would be impossible to obtain linearity in a system that uses processes that are non-linear by nature. Therefore, the open architecture of such a system, that allows the modification, replacement or removal of any part of the algorithm (a different pitch-shifting method for example), represents a considerable advantage over the use of pre-built systems, computationally efficient, but hard or impossible to edit.

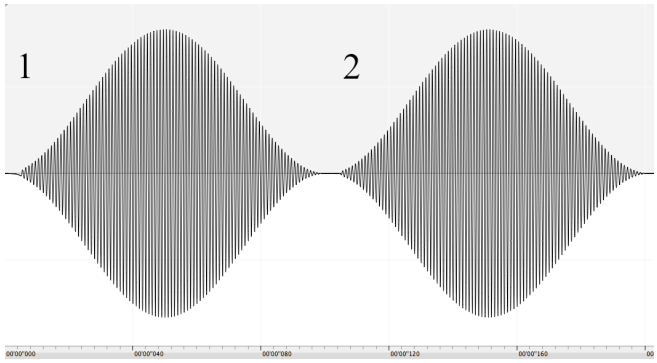

Fig. 11. Two 100 ms 1 KHz Sine wave grains. (1) Window size = 256 samples. (2) Window size = 128 samples. Overlap factor = 8, for both. Note the difference in distortion on the first samples of each grain.

References

[1] B. Truax, "*Real-time granular synthesis with the DMX-1000*," *Proceedings of the 1986 International Computer Music Conference*, Paul Berg, San Francisco, pp. 231-35, 1986.

[2] B. Truax, "*Real time Granulation of sampled sound with the DMX-1000*," *Proceedings of the 1987 International Computer Music Conference*, James Beauchamp, San Francisco, pp. 138-145, 1987.

[3] B. Truax, "*Composing with Real-Time Granular Sound*," *Perspectives of New Music*, vol. 28, no. 2, pp. 120-134, 1990.

[4] D. Trueman, "*Munger~*, external for Max/MSP," cited 2014, available from http://www.music.columbia.edu/PeRColate/.

[5] SAMPLESUMO, "*Saltygrain*," cited 2014, available from https://www.samplesumo.com/product/saltygrain.
[6] S. Nobayasu, "*Granular Synthesis v. 2.5*," cited 2014, available from http://formantbros.jp/sako/download.html.
[7] UI SOFTWARE, "*Methasynth v.5.4*," cited 2014, available from http://www.uisoftware.com/MetaSynth/index.php.
[8] N. Wolek, "*Granular Toolkit v1.0 for Cycling74's Max/MSP,*" *Journal SEAMUS*, vol. 16, no. 2, pp.34-46, 2002.
[9] I. I. Bukvic, J.-S. Kim, D. Trueman and T. Grill, "munger1~: towards a cross-platform swissarmy knife of real-time granular synthesis," Procedings of the 2007 International Computer Music Conference, Paul Berg, San Francisco, pp. 349-354, 2007.
[10] Grill, T. "flext - C++ layer for cross-platform development of Max/MSP and pd externals", Cited 2007; Available from http://grrrr.org/ext/flext/.
[11] Cycling '74. "Max/MSP: A graphical programming environment for music, audio, and multimedia", Cited 2014; Available from http://www.cycling74.com/products/maxmsp.
[12] S. Alessandretti, "*Krònos/Kairòs, per flauto, arpa e live electronics*," ARS Publica, Pistoia, 2011.
[13] C. Roads, "The Computer Music Tutorial," MIT Press, Cambridge, pp. 440-448, 1996.
[14] U. Zölzer, "DAFX - Digital Audio Effects, John Wiley & Sons, Chichester, pp. 237-297, 2002.

Stefano Alessandretti was born in Assisi (Italy) in 1980 and studied at the Florence Conservatory of Music and at the Venice Conservatory of Music, getting degrees in computer music (2008), sound direction and live electronics (2011) and composition and new technologies (2014). He was selected to attend workshop by: S. Sciarrino (2009), IRCAM (2011) and IanniX (2012).

He is a composer, an electronic musician and an independent researcher; in 2011 he taught live electronics at the Giorgio Cini Foundation in Venice and from 2012 to 2013 music technologies at the Padua Music Lyceum. His fields of interest are related to music computing, live electronics and electroacoustic composition.

Prof. Alessandretti is member of Arazzi Laptop Ensemble and artistic director of SON Ensemble. He received commissions from the Music Biennial of Venice, Fenice Philharmonic Orchestra, Giorgio Cini Foundation and Venice Conservatory of Music. His scores are published by ARS Publica and Fenice Theatre Foundation.