

## Grain\_Gaim

“(...) so that gamers everywhere can become musicians, and musicians can learn to play again.” (Money, 2004a)

### ELA2500 – Interactive and Algorithmic Systems

**Tutor: Martin Robinson**

Grain\_Game is a performance tool that can be *played* like a videogame. It is composed of four main components:

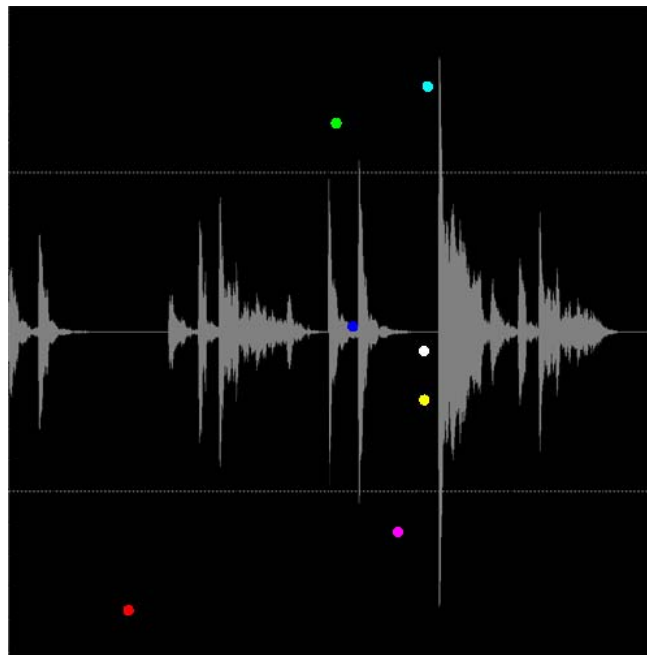
- a; A granular synthesis engine used for sound generation.
- b; Several real-time simulations of physical behaviour used to trigger this synthesis.
- c; Movement (and sound) visualisation.
- d; A game pad used for intuitive control of the system.

Grain\_Gaim is realised with the open-source programming environment PD<sup>1</sup> (0.38.3) and several extensions (GEM<sup>2</sup>, PMPD<sup>3</sup> and the [joystick]<sup>4</sup> object) on a Win XP machine.

## A; Granular Synthesis

The granular synthesis engine is largely based on the *Particlechamber*<sup>5</sup> abstraction by Derek Holzer.

In essence Grain\_Gaim is a loop player. The user can set up several sample-sets which are each composed of 8 samples. The selected sample is then *scanned* by up to seven grains, each of them displayed in another colour. In the screenshot below all available grains are activated.



The grain length ranges from a minimum of 5ms up to about half the sample length; it is represented through the size of the shapes (circle or square), which is in the correct relation to the selected waveform. The position of these shapes on the visual-waveform (the *sound space*) represents the actual reading position in the sound file.

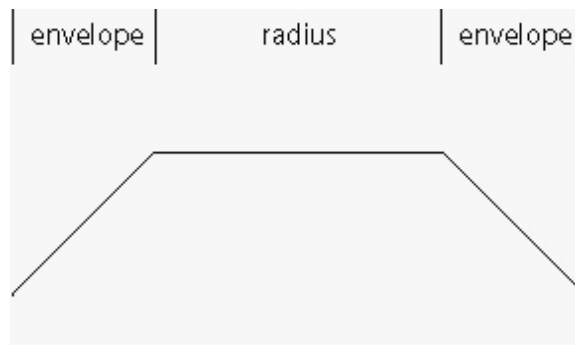
Movement in the vertical direction of the sound space affects the amplitude of the grain; the higher the louder.

For each grain three basic states can be selected:

- 0- off
- 1- circle (ambient)
- 2- square (rhythmic)

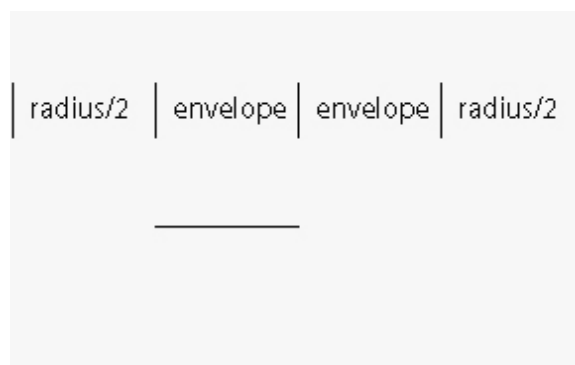
The average grain length is composed of the two factors *radius* and *envelope*.

Depending on the selected state the grain window looks significantly different:



*1-circle:*

To generate atmospheric soundscapes a smooth ramp is utilized. Attack and Release time are defined by the factor *envelope*. Sustain-time is defined by the factor *radius*. The Sustain level is dependent of the current vertical position in the sound space.



*2-square:*

To generate rhythmic textures a rather erratic amplitude function is applied. The silence is only disturbed by a sudden jump to full level for only the length of the factor *envelope*. Clicks and crackles appear and contribute to the rhythmic character.

Every time a grain has fully played back its assigned loop it starts over again. Consequently an average grain length of 250ms will result in 4 loops per second. By playing around with various grain length for both types of grain windows the best results appeared at about 60ms

length for *1-circle* and about 120ms for *2-square*. As the factors radius and envelope can only be controlled globally but not individually for every single grain, I have decided to filter out every second trigger reaching a square grain in order to get the best results when combining both types. Consequently an average grain length of 250ms will now result in 4 loops for *1-circle* but only in 2 sound bursts per second for *2-square*.

Moreover the factor *asynch* can be applied to introduce some variations in the triggering of grains. It randomly filters out triggers according to its value. If *asynch* is set to 100 (maximum) all triggers will pass unaffected; set to zero (minimum) no single trigger will come trough.

Moreover small *jitter* deviations can be applied to the factors *position*, *amplitude*, *envelope* and *radius*. *Jitter* meaning randomly generated deviations from the current value up to a set maximum. However *jitter* is not really needed for *position* and *amplitude* as the physical modelling that controls the grain movement provides already enough interesting alternation.

To make the output more atmospheric a reverb algorithm is implemented.

## **B; Physical Modelling**

The library PMPD (Physical Modelling for PD) by Cyrille Henry is used to set up four different real-time simulations of physical behaviour. All of the PMPD objects work with control data - as modelling dynamic behaviour at audio-rates would be too CPU intensive.

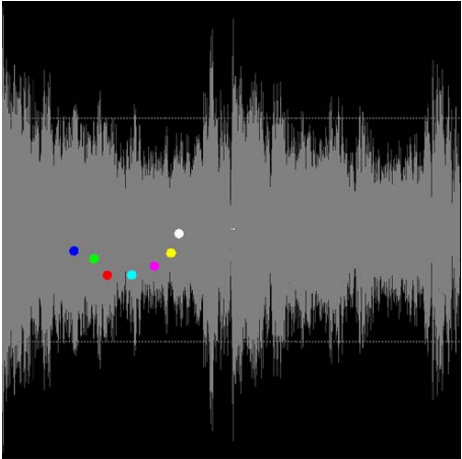
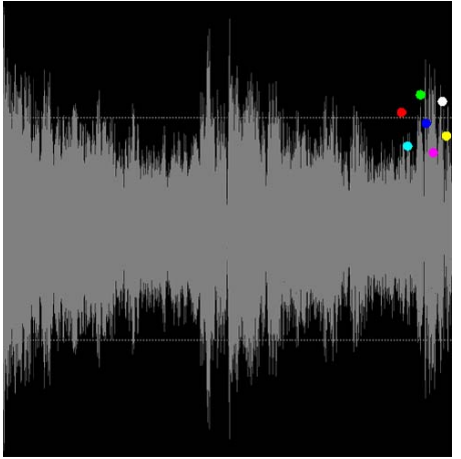
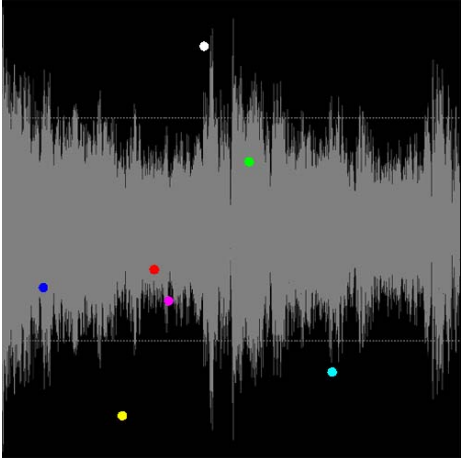
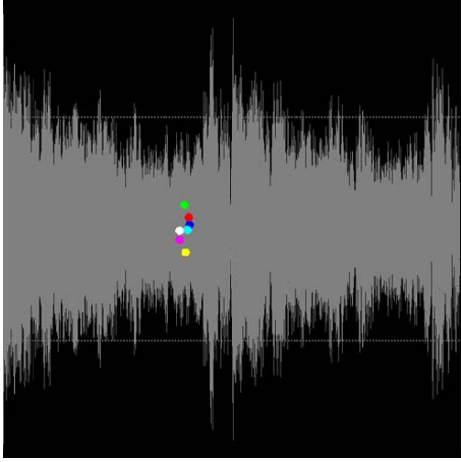
Therefore they can not be used to generate sound themselves, but rather to *control* a sound engine, for example granular synthesis like in this case.

These models provide data evolving in a very *natural* way. They represent a bridge between the user (who triggers these models) and the synthesis (which is triggered by them).

Consequently very simple information sent into the system results in very complex behaviour and therefore in interesting sound results.

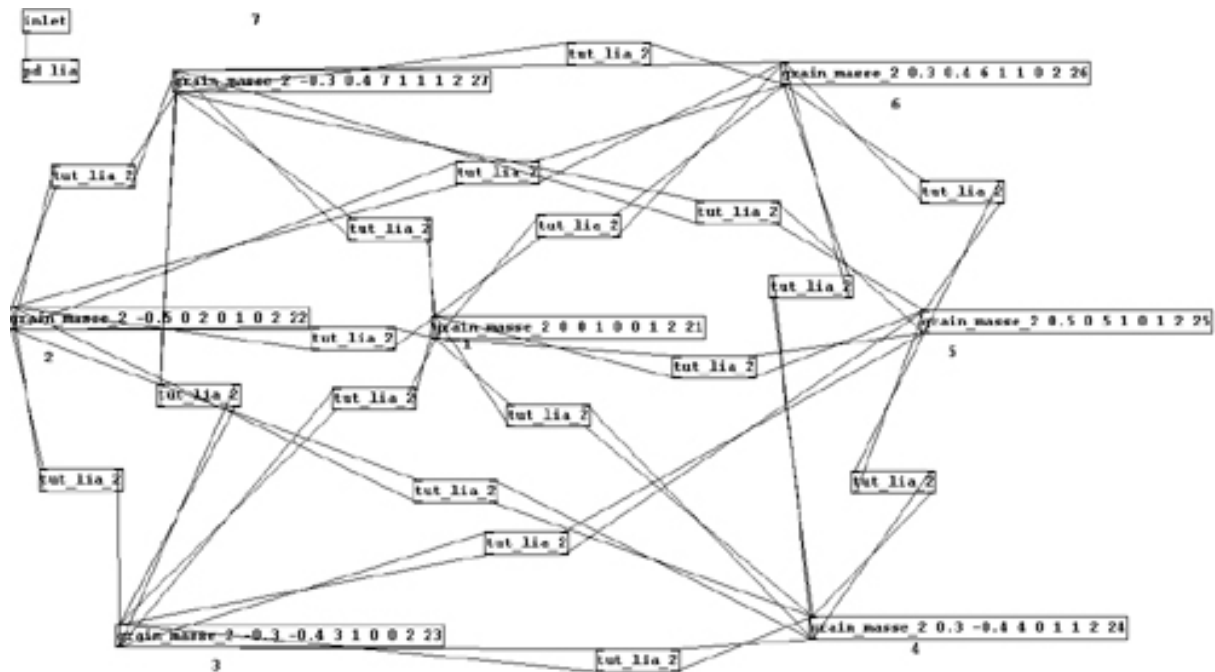
“With PMPD physical dynamics can be modelled without knowing the global equation of the movement. Only the cause of the movement and the involved structure are needed for the

simulation.” In Grain\_Gaim the cause of the movements are *forces* applied to the system by the user. Moreover there are four different *structures* to select from:

<p>1. Floating Cord</p> 	<p>2. Bouncing Ball</p> 
<p>3. Gas Molecules</p> 	<p>4. Helmholtz Instability</p> 

These structures are generated by setting up simple topologies using *mass* objects and *link* objects to connect them. These objects have to be initialised with data corresponding to physical values: weight, damping factor, length, etc.

The screenshot below shows the subpatch of the second structure (bouncing ball). For orientation reasons the mass objects are organised in a ball-like manner:



Moreover PMPD offers various objects that allow global interaction. For example the *iLine2D* object, this is used several times to set up borders in order to “capture” masses inside the sound-space.

All computation for the PMPD structures is triggered by one single *metro* object. By changing the metronome speed movements can be speeded up or slowed down. However, the faster the computation gets, the more CPU power is needed.

## C; Visualisation

As already obvious from above screenshots the visualisation of the processes going on is very simple – but effective. It uses the OpenGL-based PD extension GEM (Graphics Environment for Multimedia) that was originally written by Mark Danks to generate real-time computer graphics, especially for audio-visual compositions.

The background of the GEM Window consists just of a picture of the current waveform – as known from common sound editing tools. Unfortunately there is no effective way in GEM to draw a waveform; to do so one would have to use at least thousand instances of a single

object named *curve*. The simple work around is to provide an appropriate .jpg image for every sample used.

As already mentioned the various grains are rendered as simple two dimensional *circle* or *square* shapes with homonymous GEM objects.

If the user knows the loaded sample or has already some experience in using Grain\_Gaim he/she is pretty much able to predict the generated sound through the visual image – as information about grain-length, loudness, position, movement, state and force are instantly recognisable. One is intuitively able to grasp what's going on.

However I have to annotate one big limitation: As the grain-length gets smaller than about 30ms it is impossible to perceive it getting smaller visually, as the shape would become smaller than one pixel. Still there are huge modifications in the auditory output. For further development it would be an option to incorporate zoom mechanism in order to make the user able to *watch* even the smallest grains in right proportion.

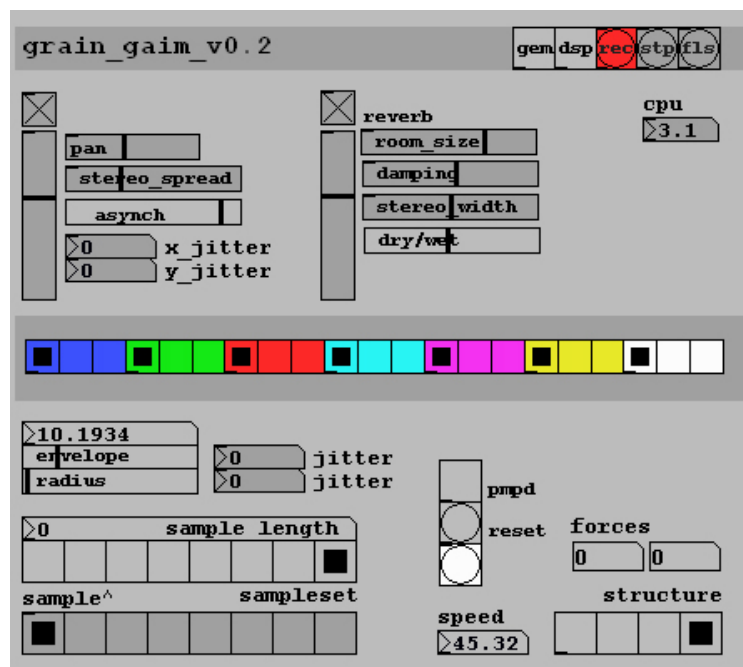
For me the most exciting feature of Grain\_Gaim is to watch/listen to the relation between sound and image - to perceive an audiovisual unity. Even more exciting is to control the system, to *scan* a sound, to *play* with it. In order to make this playing as intuitive as possible I decided to use a common gamepad as user interface.

## **D; Gamepad Interaction**

“When watching someone playing a game it is usually as easy to understand what is going on as when watching someone play a musical instrument, and so it should be for the laptop musician.” (Money, 2004b)

Through the *playing* approach much of the existing reservations against computer technology can be dissolved. Computer mediated sound production/performance can be literally laid in the hands of the user, allowing him/her to forget about the processes running in the background.

A gamepad offers very complex possibilities for user-computer interaction. The Playstation2-style pad I am using has 10 buttons and 2 wheels (4 axes) which can be easily assigned and combined to control whatever parameter in the virtual environment. The below screenshot shows the latest version of the GUI:



All elements that are transparent and the coloured preset-boxes can be controlled via the gamepad. Amplitude and position are indirectly controlled via applied *forces*.

After a while of learning the different allocations the instrument can be played fluently.

After some more practicing I plan to perform with this instrument either solo or in company with some other musicians.

## E; What's out there?

Through internet research I found two projects that - although much more complex - are based on quite similar notions as Grain\_Gaim.

*Fijuu*<sup>6</sup> is an audio visual performance engine triggered by a gamepad. It combines several synthesis methods, a non-linear beat sequencer and a graphical filter tool. *Shape Sequencer*<sup>7</sup> is an audio visual composition environment developed for simultaneous play with up to 4 gamepads. Both transport the sentiment of *playing* sound. Both emphasize the connection between the visual and the auditive.

One of the developers, Paul Money, states programmatically: “Notes on a score look nothing like they sound. We really don't need to use such limited sign-systems when we play with music, once everyone playing knows the new rules of the game. Nor is there any reason, with the technology available, to continue the tradition of a static score when music is temporal. The problem is, then, building a system that is easy to understand and yet satisfyingly flexible.” (Money, 1999b)

The systems introduced are then kind of solutions to the dedicated problem. Although much more experimenting and research has to take place, and definitely will take place in the next few years.

## References

Henry, Cyrille. (2004) “PMPD“. To download from (2005-05-11):

<http://drpichon.free.fr/pmpd/download/>

Money, Paul. (2004a) “Shape Sequencer. A new kind of musical instrument“. To download from (2005-05-11): <http://www.poil.thewonderyears.org/projects/ShapeSequencer/>

Money, Paul. (2004b) “The Shape Sequencer”.

[http://www.apfrod.com/works/2004/12#The\\_Shape\\_Sequencer](http://www.apfrod.com/works/2004/12#The_Shape_Sequencer) (2005-05-11)

---

<sup>1</sup> <http://pure-data.iem.at/> (2005-05-11)

<sup>2</sup> <http://gem.iem.at/> (2005-05-11)

<sup>3</sup> <http://drpichon.free.fr/pmpd/> (2005-05-11)

<sup>4</sup> <http://crca.ucsd.edu/~jsarlo/pd/> (2005-05-11)

<sup>5</sup> <http://testpd.iem.at/Members/derek/Particlechamber.zip/view> (2005-05-11)

<sup>6</sup> <http://fijuu.com/bin/fijuk> (2005-05-11)

<sup>7</sup> <http://www.poil.thewonderyears.org/projects/ShapeSequencer/> (2005-05-11)