Applications of Group Theory on Granular Synthesis

Renato Fabbri, Adolfo Maia Jr.

Núcleo Interdisciplinar de Comunicação Sonora – Universidade Estadual de Campinas (UNICAMP)
Caixa Postal 6166 – 13 091-970 – Campinas – SP – Brazil
{renato,adolfo}@nics.unicamp.br

Abstract. *This paper presents an application of the theory of finite groups on granular synthesis. We show how to apply finite groups to an ordered set of grains in order to get sound streams with cyclical or permutational characteristics. We point out that internal content of the grains can also be transformed creating linked structures. Using different layers of time sequencing we get a rich "polyphony" of granular sound streams. In addition we presents a computer implementation of our model named FIGGS (Finite Groups in Granular Synthesis), a rich and flexible interface developed in the Python programming language and SAGE, a software for algebra and geometry experimentation.*

1. Introduction

It is well known that complex symmetry principles have been used in musical composition since ancient times. To point just a few examples, there are a number of studies of the music of J. S. Bach, the Mozart´s dice game, or the recurrent and recursive use of the golden proportion by B. Bartók in the past century. In present days, the study of symmetries in an arbitrary number of dimensions makes use of the modern Theory of Finite Groups (see [Budden 1972] for an overview and also a musical application). The composer I. Xenakis [Xenakis 1991] dedicated a chapter, in his now, classical book *Formalized Music*, to applications of groups in algorithmic composition. On the other hand, due greatly to the exponential growth of digital industry in the second half of the twentieth century, electroacoustic music had its major developments, and now early tedious procedures can rely on fast computer processing. Roughly speaking, composition in electroacoustic music incorporates sound construction and temporal organization methods of these sounds. In this way, it is natural to argue whether group theory, that is a power tool to explore and represent cyclic and symmetric structures, can be applied in electroacoustic music for creating symmetric and cyclic organization of sound materials. This work is our contribution in this direction. In order to have focus on this approach we are most interested in the well known *Granular synthesis* technique (see [Roads 1996, 2001] for an overview). In this way we propose an exploration of symmetries in granular synthesis through applications of finite groups actions on sonic granular structures.

In the next section, we present some theoretical preliminaries and some simple examples of finite groups and also some comments on Granular Synthesis. In Section 3, we show a model, that is, an application of finite groups in granular synthesis. In Section 4 we describe shortly some of the computer implementation based on the

Python programming language using external packages for numeric manipulations and SAGE, a software for algebraic manipulation. A number of models are possible and this work is a demonstration of our method´s potential. Nevertheless, as far as we know, this is the first time one makes use of Finite Groups in Granular Synthesis. Some sound results are available on the Internet and are briefly discussed in Section 5. In Section 6 we make a conclusion and list some perspectives of the method we named FIGGS (Finite Groups in Granular Synthesis). In last section we present some bibliography.

2. Theoretical Preliminaries

2.1. Group Theory

Formally, a group G is a set with a binary rule (which we will denote by ' • ' in this work) that satisfy these four fundamental properties:

1) if $g_1, g_2 \in G$ than $g_1 \bullet g_2 \, G$ (Closure)

2) $g_1 \bullet (g_2 \bullet g_3) = (g_1 \bullet g_2) \bullet g_3$ (Associativity)

3) $\exists\ e \in G : g \bullet e = e \bullet g = g$ (Identity)

4) $\forall\ g \in G,\ \exists\ g^{-1}\ :\ g \bullet g^{-1} = g^{-1} \bullet g = e$ (Inverse Element)

We denote as (G, •) a group with an operation •. A group is finite if it has a finite number of elements, else it is called infinite. A group is called *commutative* or *abelian* if the commutative property is satisfied for all its elements, that is:

$\forall\ g_1, g_2 \in G, g_1 \bullet g_2 = g_2 \bullet g_1$

Some common simple examples:

a) (Z, +) where Z is the set of the integers.

The set of the integers form a group with the sum operation. The identity element is the number zero and for each integer $a \in Z$, its inverse is $-a$. (Z,+) is abelian.

b) (Q^+, x), where Q^+ is the set of the positive rationals.

The identity element is the number 1, and for any $q \in Q^+$, $q^{-1} = 1/q$, and $1/q \in Q^+$. Q^+ is abelian. This is the simplest example of a multiplicative group.

c) $(Z_p, \oplus)$, where Z_p is the set of the integers modulo p, with p a natural number.

The identity element is zero, and for any $a \in Z_p$ its inverse a^{-1} is given by

$(p - a) \bmod(p)$. When p is a prime number Z_p becomes a multiplicative group.

d) $(M_n^+, .)$, where M_n^+ is the set of invertible quadratic matrices $n \times n$, and . is the usual matrix product. The identity element is the *Identity Matrix* I_n and the inverse element of an element A is the inverse matrix A^{-1}.

We are most interested in permutation groups of a number n of objects. Not all groups have permutations as elements, but each finite group is isomorphic to a

permutation group. So, we can restrict ourselves to permutation froups without loss of gnerality. Besides that, permutation groups are a mathematical tool for describing symmetries, which are strong factors of artistic thinking. For more on Group Theory see, for example, [Budden 1972].

2.2. Granular Synthesis

Granular synthesis is commonly known as a technique that works by generating a rapid succession of tiny sounds, metaphorically referred to as sound grains or yet as microsounds. Granular synthesis is mostly used by musicians to compose electronic or computer music as it can produce a wide range of different sounds, but it has also been used in speech synthesis. Granular synthesis is largely based upon D. Gabor's idea of representing a sound using hundreds or thousands of elementary sound particles [Gabor 1947]. His approach to "elementarity" was inspired by the *Uncertainty Principle* of Quantum Mechanics. He proposed the basis for representing sounds combining frequency and time domains information. D. Gabor's point of departure was to acknowledge the fact that the ear has a time threshold for discerning sound properties. Below this threshold, different sounds are heard as clicks, no matter how different their spectra might be. The length and shape of a wave cycle defines frequency and spectrum properties, but the ear needs several cycles to discern these properties. D. Gabor referred to this minimum sound quantity as an acoustic quantum and estimated that it usually falls between 10 and 30 milliseconds.

The first formal musical thoughts using D. Gabor's sound representation (*time x frequency* space) in music were, probably, initiated by composer I. Xenakis. A computer-based granular synthesis system did not appear until works of C. Roads [Roads 1988] and B. Truax [Truax 1988]. As far as the idea of sonic grains is concerned, any synthesizer capable of producing rapid sequences of short sounds may be considered as a granular synthesizer. However, it is an important fact that the very concept of grain differs in many cases, as for D. Gabor and I. Xenakis. The latter describes a grain as the instantaneous measured of a Fourier Partial associated with a particular frequency and amplitude, which is, for him, the most elementary characteristic of a sound. In this case, a cloud of grains can be thought as a cluster of points relatively close to each other in the *frequency x amplitude* space. And indeed, I. Xenakis proposed a density parameter to measure the compactness of a cloud and the duration of a grain as an external parameter to control the sound as a whole. For D. Gabor, however, duration is an internal parameter of the grain itself, which can have a complex content in terms of Fourier partials. D. Gabor borrows the concept of quantum of action from Quantum Mechanics to define a quantum of sound. The quantum action of sound A is in the order of unity, that is

$$A = \Delta\omega.\Delta T = 1 \qquad (1)$$

In early development, C. Roads suggested the following definition: *"A grain is a signal with an amplitude envelope in the shape of a quasi-Gaussian bell curve"* [Roads 1988]. This is close to D. Gabor's original definition in the sense that it accepts grains with complex spectral content. However, the concept of "elementary" (or quantum) is not strictly taken into account in C. Roads' definition. The interpretation of "quasi-Gaussian bell curve" can be very general. In addition, his concept of grain density is actually a measure of the number of grains occurring within a given time interval.

C. Roads also suggests that granular synthesis can be classified as a form of additive synthesis. We therefore prefer to consider this definition as being for a specific type of granular synthesis, which we refer to as Short-Time Additive Synthesis (STAS). A key problem in granular synthesis is controlling the evolution of sound grains in time. Most granular synthesis systems have used stochastic methods to control evolution of hundreds or even thousands of grains. A handful of different methods have been proposed. For instance, E. Miranda devised Chaosynth, a granular synthesizer of the STAS type that uses cellular automata to manage the spectrum of the sound grains [Miranda 2002]. Chaosynth explores the emergent behavior of cellular automata to produce coherent grain sequences with highly dynamic spectra. The states of the cellular automata defines frequency and amplitude values for an additive synthesis engine that produces the granular streams. A critical review and present status of Granular synthesis can be found in [Thomson 2004].

Once the sonic grains are discrete entities, it is possible to think them as objects that in which we can impose internal and external symmetries, the last one related to temporal organization of grains.

3. A Theoretical Model

In this section we presents one of our models. The idea is to show a way to construct an application from groups to granular synthesis.

Here we are going to consider four permutation groups.

a) *The Symmetric Group of degree n*. the group of all the permutations on an ordered set of *n* elements.

b) *The Alternating Group of degree n*: the group of even permutations on a set of *n* elements, that is the set of permutations obtainable from an even number of two-element swaps.

c) *Dihedral Group*: the symmetry group of a n-sided regular polygon for n > 1.

d) *Cyclic Group*: a group which can be generated by a single element and the group operation. In our case we use Permutations Cyclic Groups.

In this first model we take one of the groups mentioned above. We are, currently, taking random elements of these groups, i.e. permutations belonging to specific classes, and applying it to the state of our granular parametric set. The above model can be applied to any set of grains in order to define a sound stream whose one of the main psychoacoustic characteristic is the cyclical sound structure in time. In addition, it is possible to apply group transformations inside the grain itself. This can be done, for example, using a set of transformations on granular parameters (frequencies, duration, amplitude, etc) and imposing on this set of transformations a group structure. This approach leads to construction of granular structures with strong internal correlation.

4. Algorithm and Computational Implementation

4.1. Development Framework

We have based our implementation on *Python*, an object-oriented and high-level programming language that uses automatic memory management and dynamic typing. The language has an open, community-based development. It have been used for a wide range of tasks: GUI design, web development, stand-alone programs, among others. It has been extensively used by Google and NASA, for example.

Python's standard library comes with a bundle of modules that provides basic functionalities, but the language is truly extensible for audio manipulation by accessing some of the innumerable third party modules such as PySndObj, NumPy and Matplotlib. For computational algebra, we can point the excellent SAGE (Software for Algebra and Geometry Experimentation), which is written in Python and a modified version of Pyrex (SageX) unifying specialized open-source math software and filling in functionality gaps. SAGE is a stand-alone software that uses an individual python interpreter to run scripts, so we can install python modules to it and extend SAGE's functionalities[1]. For our present implementation, besides standard functionalities in SAGE, we used:

- wxPython: For GUI design.
- NumPy: For fast numerical array processing.
- PyAudioLab: for writing NumPy arrays to sound files.
- Matplotlib: for plotting graphs like waveforms and spectrograms.

4.2. Algorithms

For the sake of complementarity, in this section, we show how we can combine Python facilities with the calculational power of SAGE for algebraic structures such as, in our case, groups. Within this framework we can create a lookup table for a sinusoidal waveform like this:

```
import numpy

cycle = numpy.linspace( 0, 2*numpy.pi, 2**11, endpoint=False )

table = numpy.sin( cycle )
```

Here, we imported the NumPy module and used it's linspace function. That function takes as arguments a starting number, an ending number, and a number of steps, respectively, and returns an array with “number of steps” elements that specifies the number of evenly spaced samples from start to stop. Writing x**y in python is evoking the number x raised to power y. The end number is not included in the constructed array if endpoint is declared “False”, which is our case since sin(2*pi) is sin(0). Our lookup table is the sin function called upon the defined cycle array. We can use this table by calculating an increment SI = f * N / SR (where f is the frequency of the signal we

1 There are plans for future versions of SAGE supporting it's use as a normal python module. See discussion thread “Is it possible to use SAGE from regular python?” at: http://groups.google.com/group/sage-support/

want to use, N is the length of our lookup table, and SR is our sample rate), and iterate over the table:

```
for i in range( int(duration*SR) ):
        sig[i] = table[ int(ap) ]
        ap = (SI + ap)%N
```

The function range calls a list of integers from zero to the input value. If we have a desired duration, the number of samples of the signal equals the integer part of the duration (seconds) times our sample rate.

Invoking groups in SAGE is easy. Here goes some examples for permutation groups:

```
S = SimmetricGroup(5)
```

This creates a symmetric group of degree 5.

```
A = AlternatingGroup(4)
```

This creates an alternating group of degree 4.

```
C = CyclicPermutationGroup(7)
```

As mentioned above, a cyclic group can be generated by a single element g (g^2, g^3...). Its representation as a permutation group is called by the CyclicPermutationGroup(x) function (*x* is the order and the degree of the group).

Invoking elements of a group can be done in various ways. We illustrate here two of them on the SAGE command line interpreter:

```
sage: S = SymmetricGroup(3)
sage: list(S)
[(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
sage: list(S)[1]; list(S)[3]
(2,3)
(1,2,3)
sage: S.random_element(); S.random_element()
(1,3,2)
(2,3)
```

These tuples are elements of the group S_5, they are permutations. In this permutation representation we read (2,3) as element in place 2 goes to the place 3 and element 3 goes to place 2. We read (1,3,2) as element in place 1 goes to place 3, element in place 3 goes to place 2 and element in place 2 goes to place 1. We can apply this permutation to an integer:

```
sage: g = list(S)[4]
sage: g
(1,3,2)
sage: g(1)
3
```

If we have an ordered set (a list) X of parameters and a permutation (a group element) g, we can apply that permutation on that set with these lines:

```
for i in range( len(X) ):
        perm[ g( i +1 ) -1 ] = X[i]
```

This +1 and -1 occur because *Python* lists have indexes from zero to the length of the list minus one, and permutations in SAGE ranges from one to the length of the set in which it acts. This takes us to the last procedure: writing the sound file.

We are using NumPy arrays in order to add functionalities to array manipulations, to simplify its use and to improve processing speed. There is a module named Pyaudiolab that is dedicated to writing NumPy arrays to sound files, and opening sound files as a NumPy arrays. It supports all sound file formats supported by *libsndfile*, including WAV, AIFF, Ircam SF, FLAC and RAW.

We can write a 16 bit WAV sound file from a NumPy array (here instanced as the signal object) using Pyaudiolab like this:

```
import pyaudiolab
fmt = pyaudiolab.formatinfo( 'wav', 'pcm16' )                (1)
sndfile = pyaudiolab.sndfile( name , 'write', fmt, nc, sr)   (2)
sndfile.write_frames( signal, len(signal) )                  (3)
```

(1) Specifying the file format

(2) Creating the object sndfile for file writing, specifying its name, its format, its number of channels and its sample rate.

(3) Calling the write_frames method of the created sndfile object to write samples in the signal array for its entire length.

As a way of experimenting with this implementation, the user chooses the number of grains and inputs parameters for individual grains. Than chooses the parameters to be permuted, the groups that acts on each of these parameters and commands for the sound to be written.

GUI programming issues will not be discussed since they are beyond this work's subject. Even so, we will present our interface and use some screenshots to illustrate a way to approach the features presented.

4.3 FIGGS Interface

We have developed a GUI (Graphical User Interface) in Python for FIGGS (Finite Groups in Granular Synthesis). With this we aim to furnish facilities in order to represent structural ideas as settings of values allowing the user to concentrate on musical creation's symmetric aspects. We intend this GUI interface for the FIGGS model to be friendly and flexible enough in order to allow the user to construct his own random distribution. Random distributions for granular synthesis are the quickest way to input parameters, but, of course, the user can also input the parameters values in a deterministic way as he/she wishs.

This interface has two tabs: one for granular parametric specifications, and another for sound parameters on which chosen groups acts, each with its proper controls.

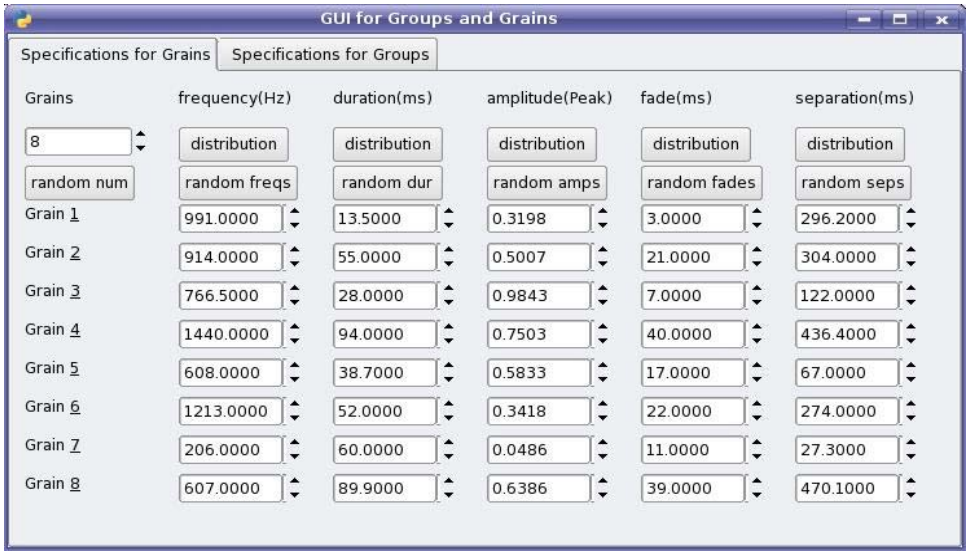

Figure 1. FIGGS Panel for Grain Specifications

Each parameter of each grain can be inputed independently or by ease of a random distribution. Different distributions are useful here but only the homogeneous one is currently implemented in the GUI. Negative separations are understood as superposition. This creates an ordered set of grains, which is going to be manipulated by group structures. Once this table is filled in, it can be manipulated by classes of permutations. Mathematically speaking, these manipulations are specific permutation groups acting on columns of this matrix. A musician can use this freedom of experimentation in order to explore proportions among the physical aspects of the grains and their evolution in time. These proportions are a significant factor of musical creation (see, for example, [Xenakis 2001]) and analysis [Shmulevich et al. 2001]. For a recent musical example using this approach and FIGGS, please see Section 5.

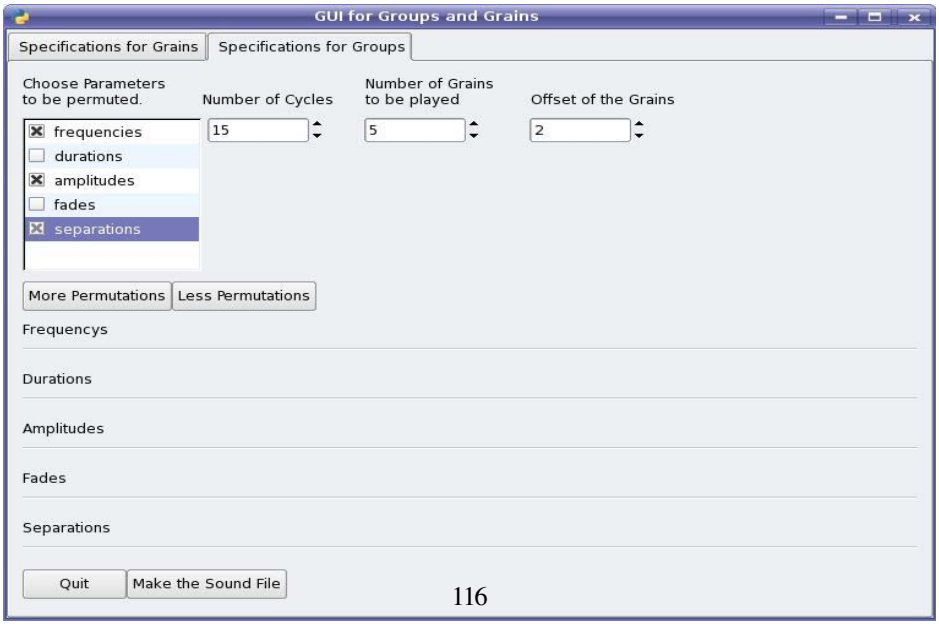

Figure 2. FIGGS Panel for Groups - Choosing Grain Parameters

Figure 2 shows a panel where we can choose independently those parameters that we want to permute. That is done by stating the number of times the specified grains are going to be played (the number of cycles), the number of grains of each cycle (a joint subset of the grains set), and an offset of this subset in the grain set. These are the necessary inputs before the sound file can be done, so the user can choose to play the grains in the exact sequence defined in the grain panel for an arbitrary number of times.

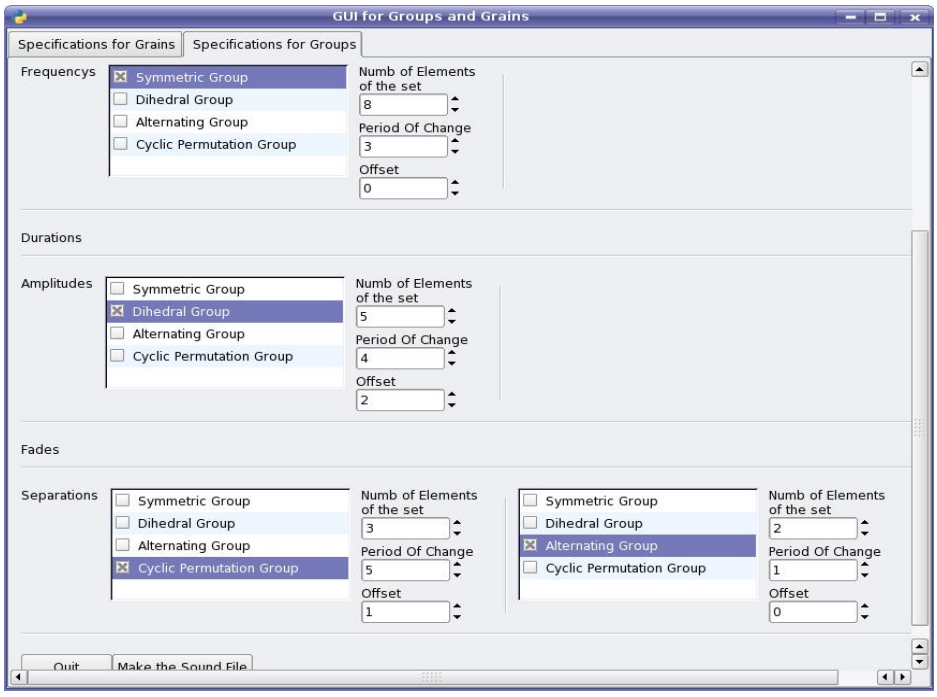

Figure 3. FIGGS Panel for Groups - Specifying Groups

Figure 3 shows some group usage. The number boxes besides each group list are the number of elements in which the group acts (top), the number of cycles between each group action, and an offset of the action set in the grains set. There are numerous ways in which a group can act on a parameter set, and there are other groups to be explored as well, and more options are going to be available in future developments. In this screen shot's version, the action is performed by successive actions of random elements of a group (i.e. permutations) on the set. Notice that the number of the permuted elements need not to be equal to the number of grains played or the number of grains in the grains set. In fact, that is one of the main features of this FIGGS implementation. The user can freely move back and forth from the grain panel and the group panel, and command the sound file to be written any number of times. This allows making any number of sounds with related behavior and related sounds.

Finally, it should be mentioned that is possible to have group actions on spectral contents of a set of grains. This will be implemented in a near future.

5. Resulting Sounds

One of the main tools used in computer based audio recognition is the representation of sounds as a long term statistical distribution of its local spectral features. Such an analysis of structures created within FIGGS methods would be about the same if there were no group actions involved, for statistical occurrences of the events remains the same. Even so, there are strong indicatives that musical cognition often rely on other symbolic and analytical level than a physical holistic description can reveal [Aucouturier, Defreville, Pachet 2007]. Musical patterns can be thought as being concurrent relations of frequencies and relations of grain onset timings in the metric pattern. Melodies are an ordered set of these relations [Shmulevich et al. 2001]. In general, FIGGS structures, if build with densities in the range about 0.1 to 10 grains per second, resembles melodies that, due to the set of frequencies choosed, has fixed melodic scales. With this respect, the users can find a sound bank with some recent patterns examples, at: http://cortex.lems.brown.edu/~renato/sonic-art/nics/FIGGS-patterns-ex.zip

Different granular synthesis techniques can be used in the current FIGGS interface. One can, clearly, make synchronous and asynchronous granular streams, as well as quasi-synchronous granular synthesis**.** Irregular onsets of the grains leads to a controllable thickening of the sound spectrum [Truax 1988]. Here is a link for downloading a sound bank with according examples:

http://cortex.lems.brown.edu/~renato/sonic-art/nics/FIGGS-clouds-ex.zip

It should be pointed that the only amplitude envelope currently implemented in the GUI is trapezoidal.

A musical experimental piece (~10 min) was created using FIGGS capabilities. It can dowloaded at:

http://cortex.lems.brown.edu/~renato/sonic-art/nics/reflexoes-paradoxais.mp3

This piece is about two prose manuscripts by Fernando Pessoa (1888-1935), namely *Solidão* (1947, 1915?) and *Reflexões Paradoxais* (1916?), and was composed in 2007 under artistic supervision of J. A. Mannis (Music Department, UNICAMP). Passages that makes use of FIGGS are mainly before 02'32” and after 06'52”.

The current SAGE/Python code is already available at: http://cortex.lems.brown.edu/~renato/sonic-art/nics/FIGGS.zip. A brief README.txt instructs how the framework must be set to run the script.

An executable version of FIGGS, as well as the examples above mentioned, will be available, by August 2007, at:

http://www.nics.unicamp.br/atual/pessoal_renato.htm.

6. Conclusion and Perspectives

As mentioned in the introduction section, group theory can be a valuable tool for algorithmic composition. Nevertheless, as far as we know, its application in sound synthesis was not pursued seriously until now. Our model is a preliminary study in this direction. We have used, for the sake of simplicity, the well known finite permutation groups which can demonstrate the potential of the model for more complex

applications. In addition we can point out some directions and perspectives for future work:

A) In order to construct musical structures we can use superposition of sequences of grains. In other words we can construct layers which can be controlled independently (or interdependently) in terms of content and duration of the grains as well as their time sequencing. This conception can be envisaged as a tool for granular synthesis composition. In addition, new rhythmic aspects can emerge from this kind of sound design.

B) The use of other groups whose elements are not permutations, such as Fundamental Groups, which can be used to describe and impose orbits related to topological spaces.

C) In order to generate an arbitrary quantity of grains we can make use of probabilistic distributions such as Gaussian, Binomial, Bernoulli, among others. Also from the parametric point of view, different granular parameters can have linked values.

D) Sounds can be generated not only by lookup tables, but by other synthesis methods, such as FM synthesis, and have its synthesis parameters controlled by groups in order to have streams of related and coherent sound particles. Sound particles can be generated as well by using loaded sounds, or slicing it at group controlled intervals.

References

Xenakis, I., *Formalized Music*, Bloomington: Indiana University Press (1971); also, *Formalized Music*, 2d ed., New York: Pendragon Press (1991).

Budden, F.J., *The Fascination of groups*, CUP,(1972).

Gabor, D., *Acoustical Quanta and the Theory of Hearing*, Nature **159** (4044), pp. 591-594,(1947).

Roads, C., *Introduction to Granular Synthesis*, Comp. Mus. Jour., **12** (2), pp. 11-13 (1988).

Roads, C., *Microsound*, MIT Press, Cambridge,MA, (2001).

Roads, C., *Computer Music Tutorial*, MIT Press, Cambridge, MA (1996).

Thomson, P., *Atoms and errors: towards a history and aesthetics of microsound*, Organized Sound, **9** (2), pp. 207-218, (2004).

Miranda, E. R., *Computer Sound Design: Synthesis Techniques and Programming*, Oxford: Focal Press (2002).

Aucouturier, J.-J, Defreville, B., Pachet, F., *The bag-of-frame approach to audio pattern recognition: A sufficient model for urban soundscapes but not for polyphonic music*, Journal of the Acoustical Society of America, 2007.

Shmulevich, I., Yli-Harja, O., Coyle, E. J., Povel, D., Lemström, K., *Perceptual Issues in Music Pattern Recognition – Complexity of Rhythm and Key Finding*", Computers and the Humanities, Vol. 35, No. 1, pp. 23-35, February 2001.

Python Software Foundation,, "*Python Documentation*", http://www.python.org/doc/, as acessed in 02/06/2007.

Stein, W., *SAGE – "System for Algebra and Geometry Experimentation"*, at http://www.sagemath.org/sage/documentation.html, a acessed in 02/05/2007.

Scipy and Numpy documentation at http://www.scipy.org/Documentation, as acessed in 02/06/2007.

Cournapeau, D., *"Pyaudiolab homepage"*, at http://www.ar.media.kyoto-u.ac.jp/members/david/softwares/pyaudiolab/, as acessed in 02/06/2007.