Oficina Granular: a tool for composing with granular synthesis

Carlos Eduardo Mello
melloedu@unb.br, Depto de Música, Universidade de Brasília

Abstract*. This article describes a software project called Oficina Granular (OG). OG attempts to provide composers with an intuitive software interface for creating and assembling granular sound masses into a complete musical composition. Studies and software implementation of granular synthesis tend to concentrate on grain generation. OG uses a simple grain implementation and focuses on higher level manipulation of sound masses. The main libraries for OG were written in standard C++. The Graphical Interface was planned for Mac OS X and is currently under development.*

1. Introduction

The idea of utilizing granular components to produce complex sounds is not new. Several studies and implementations have spawned among computer music researchers in recent decades. Many articles, like those of Roads (1985, 1988) and Truax (1988, 1994), have helped bring to light the possibilities and limitations of this kind of synthesis technique. Many of these works have focused on the possibilities of the resulting sounds and their spectra. Others provide examples of specific implementations and their particular strategies for controlling grain generation. Jones and Parks (1988) discuss two methods for working with grains: extracting them through the granulation of natural sounds and synthesizing them with frequency modulation. Truax (1988) reports an implementation of real time granular synthesis using a dedicated DSP microprocessor. Behles, Starke and Röbel (1998) propose a combination of various methods of granular synthesis, including techniques developed in speech-processing research, into one program.

Despite the great interest generated by a granular model of sound, the large amount of computational resources required has imposed, in the past, substantial limitations to the widespread use of this kind of technique. Roads (1985) reports that his early experiments with granular synthesis involved specifying parameters for each of hundreds of grains in punched cards, which were then processed by a large computer. Also, the complexity involved in the studies about the subject have often kept granular synthesis techniques away from the grasp of musicians who do not have previous training in computer sciences. Recent developments in the computer industry, however, have permitted the average musician to access sophisticated sound synthesis software.

The software described in the present article, *Oficina Granular,* was intended to bring granular synthesis closer to the 'non-engineering' musician. The idea was to provide composers with a tool for manipulating granular sound masses, without the need to write a complete computer program. The software was initially designed around the end of 1999. Its current implementation, a test version for standard C++ console (OG - console v1.0), was concluded in 2002. Instead of discussing this test console

program, a transitional version, this article describes the program's main library (see **Implementation**) and the main features of the Graphical User Interface, which is currently being developed (see **Future Work: Interface)**.

2. Implementation

Oficina Granular is constructed around three main building blocks: **events**, **envelopes** and **compositions**. Each one of these parts has a corresponding class in a C++ library and contributes to build a virtual machine, which is the logical center of the program. The following sections describe this library and its components. User interaction with this virtual machine is provided through the Graphical User Interface described in the last section of this paper.

Events

Events are the main musical units of *Oficina Granular.* Each event object stores information about a single sound mass, which may be used to create a long and evolving sound or a note in a more traditional sense. In either case, the event describes all the parameters involved in the creation of sound grains. This model is closely related to Roads' implementation (1978), in the sense that the grains are generated linearly, and distributed in time according to a specific rate of succession. In OG, the grains are synthesized sounds, which have a fixed wave form (sinusoid) and whose frequencies are randomly picked from within a limited bandwidth. Because of the massive amounts of grain data used in the program, the actual grains are created only at *build* time. Every new build produces a new set of (*pseudo*) randomly chosen grains, according to currently assigned parameters.

Event parameters are defined as follows: 1. start time; 2. duration; 3. grain rate; 4. grain size; 5. base frequency; 6. band; 7. amplitude; 8. balance (pan). The **start time** of an event is used to insert the event in the composition's time-line. When the grains are generated, each one has its own starting point which is referenced to its containing event's start time. This facilitates moving the event in time. The **duration** parameter is the total length of the event. **Grain size** and **grain rate** are applied to every grain in the event object. These parameters describe, respectively, the duration of individual grains and the rate at which the grains are started. The **base frequency** and **band** parameters are used to determine an interval from which to choose a grain's frequency content. Band values may be linear -- in which case they merely describe a specific frequency boundary -- or proportional to the base frequency. This last feature is useful for setting frequency intervals in a more musical way (e.g.: 4 = 2 octaves). Although the base frequency usually represents a lower bound for frequency choice, this assignment is completely arbitrary; in other words, these boundaries may cross at any time. The **amplitude** of the event is the maximum amplitude of each grain at any given moment. The actual perceived volume for a sound mass can vary with density changes. In the beginning of an event, for instance, an attack (ramp) effect is usually noticed if the grains are allowed to overlap. Finally, every grain in an event may be 'spatially placed' according to a **pan** value, which distributes its signal between a left and a right channel.

Event objects use the `Build()` function to generate all grain data. This is a central method, which creates the grains by checking the parameters at a given time, depending on the grain rate of succession. Since the grain rate may be variable,

`Build()` checks it before writing each grain. This way, the starting point for each grain will be directly affected by changes in the desired density (**`grainOffset = 1/grainRate)`** Changes in this and in all other parameters are verified by internal function calls, which calculate the new values based on the start time of the next grain.

Envelopes

This section describes the strategy utilized by *Oficina Granular* to provide dynamic changes to event parameters. Each one of the parameters described above may be defined by a fixed value (static) or according to an assigned envelope (dynamic). The term **envelope** is used in *Oficina Granular,* in a rather generic way, to describe an array of values, which combined together create a discrete curve. The values in this curve are used as control points for the `Build()` function. Envelopes may be applied to any given parameter, including the grain rate of succession. This gives the events the flexibility to behave as either a static block or a constantly evolving sound. Although each event stores its own envelopes locally, a separate class of **envelope objects** is also provided. This way the composer may modify and reuse previous envelope data; also, envelope objects can be used as a storage units for data generated by graphics tools

The envelope class stores the value points for the envelope and a few other information fields such as its name, type and size. As in the case of events, the names are optional descriptors; the type of an envelope defines whether it is used to store integers (16 bit) or floating point values; the size field stores the length of the sequence of values in a 32 bit integer. This length value determines the resolution of the envelope. However, when the event is applied to specific synthesis parameters, the amount of time each control value is in effect depends on the duration of the event. For instance, an envelope with 25 values used in an event which lasts for 1 second, has 40 miliseconds between each control point. For an event of 5 seconds, this same envelope will have 200 ms between values.

Composition

As mentioned above, one of the main goals of *Oficina Granular* is to provide the composer with useful editing features to manipulate granular synthesis data. This is done by encapsulating the configurations for a sound mass in the event unit. The composer can then create as many events as he/she wishes and modify them as needed. In order to assemble all the events and store user defined envelopes, OG implements a **composition** class. Each **composition** stands as a unit, assigned to a file which can be loaded from and written to disk. The composition methods include functions for handling file input and output, controlling the lists of events and envelopes, and providing the editing APIs for the graphical interface.

Besides being able to alter the synthesis parameters (grain rate, frequency band, etc.), the composer can edit the events inside a composition in the following ways: removing an event, duplicating an event, moving an event in time, transposing an event's base frequency and creating a melodic line with a given event's definitions. When an event is created, it is inserted in the composition's list of events. The program gives it a new location in the time line by simply assigning it a new start time. Other features like event duplication are achieved internally by copying event objects. Since

grain data is calculated only when needed, there is very little memory overhead involved in copying events during edit operations. This way, a composition can receive and pass events as parameters to its own methods and to communicate with the interface. Melodic lines can be created through a series of events with similar configurations, each bearing a different base frequency value. This works best when band values are kept static and proportional. Instead of duplicating each event manually, the composition class provides a specific API for this, which takes an event ID and a melodic sequence as parameters.

Rendering

At any time during the editing process, the composer may review the results of the composition. This may be done for individual events or for the entire work. The realization of the composition, here referred to as rendering, involves calling an event's `Build()` method to generate a list of grain data according to the event's parameters. When rendering all the events, the composition methods go through the event list and call the `Build()` method on each one. The final output of an event or composition rendering is handled in two different ways:

1. Sound - this method is currently under development. It consists of creating a built-in sound engine for the grains, which will produce either a final sound file in standard format (AIFF, WAV) or mix the grains and feed them to the computer's sound card, or both.

2. Csound File format - this method has been implemented in the current version of OG. The sequence of events is written to a Csound score file (.sco), so that each grain becomes a note line. OG also generates a Csound orchestra file (.orc). containing the definition of an instrument which ultimately *plays* the grains.

3. Future Work: Interface

In order to achieve the goals set for this implementation of granular synthesis, it is fundamental to provide the composer with control over all features of the program, through an easy-to-use, intuitive and effective user interface. *Oficina Granular* addresses this issue by coupling the class library previously described with a standard graphical interface designed for Mac OS X.

Windows

The working environment for *Oficina Granular* revolves around three main windows, which address the issues of event parameter control, envelope editing, and time-line assembling of granular sound masses (Figure 3). The main editing window in *Oficina Granular* is the **Composition Screen**. Every event in the composition is displayed in it, in the form of graphical shapes of different colors; each shape outlining the frequency boundaries for a specific event.

One of the difficulties encountered with the console version of *Oficina Granular* was the large amount of typing required to assign each event parameter, monitor its effect on the event and make new changes. The **Event Editor** window allows the composer to set all parameters in a single dialog box, change the parameters at any time, apply the changes and monitor them. Another problem with the text interface was assigning values for parameter envelopes. This became very cumbersome when dealing

with envelopes of resolution higher than a dozen values. In these cases, the composer would have to type all the values at the prompt, or pre-type them into a file for loading. The graphical interface deals with this by providing an **Envelope Editor** window, in which the composer will be able to 'draw' the envelope's curve, either with free mouse movements or by applying drawing tools. Two other windows complete the working environment for *Oficina Granular*: the **Event List** and the **Envelope List**. These show the objects created in the current composition, arranged by name, by start time (events), or in the order they were created.

User Interaction

The user of *Oficina Granular* interacts with the program's Graphical Interface by creating and editing events, designing envelopes for event parameters and rendering the composition (or part of it). The first step in using the program is to create a new composition, using the appropriate command in the file menu. The program then presents a blank **Composition Screen**, which indicates time on its horizontal ruler and frequency on its vertical ruler. At this point, the composer may create a new event by 'double-clicking' at the desired point in the time line, or by issuing the corresponding command in the **Composition menu**. These actions open up the **Event Editor window** so that the user can choose the parameter configuration for the newly created sound mass. Once in the Event Editor, each parameter can be set in one of two ways: (a) moving the slider control or typing a specific number in the text field - for static values; (b) clicking the Edit button to create or load an envelope - for dynamic values. Finally, clicking the Apply button stores all the new values in the current event an closes the Event Editor.

Existing events are displayed in the **Composition Screen** and in the **Event List window**. These two windows are linked together, so that any changes in one are immediately reflected in the other. Moving an event in time is accomplished by dragging the corresponding shape along the Composition Screen's time axis. If the Event List is opened and ordered by start time, the user sees the event item move to another position in the list. Moving an event's shape along the vertical axis translates into changes in the base frequency parameter for the event being moved. If base frequency is set to dynamic mode, the reference is taken from the first value in its envelope. The remaining values are transposed accordingly. The user can duplicate an event by selecting it in the Event List and calling the **Duplicate Event** command in the **Edit menu**. He/she may accomplish the same result by 'option-dragging' the event's shape in the Composition Screen. This alternative permits the user to create and relocate the new event to another position in time, all in one movement. To open the event and edit its contents, the user 'double-clicks' its shape.

The **Envelope Editor** is summoned each time the user clicks the **Edit button** for a parameter or an item in the **Envelope List window**. Once opened, The Envelope Editor allows the user to: create a new envelope; load an existing envelope; change an envelope's resolution; modify an envelope's curve; save an envelope with a different name. All this is handled by clicking a button, using a menu command or, in the case of editing the envelope's values, by dragging the mouse across the Editor's window. This

last action set's the values according to the vertical position of the cursor. Value ranges in the Envelope Editor's vertical ruler vary according to the type of envelope.

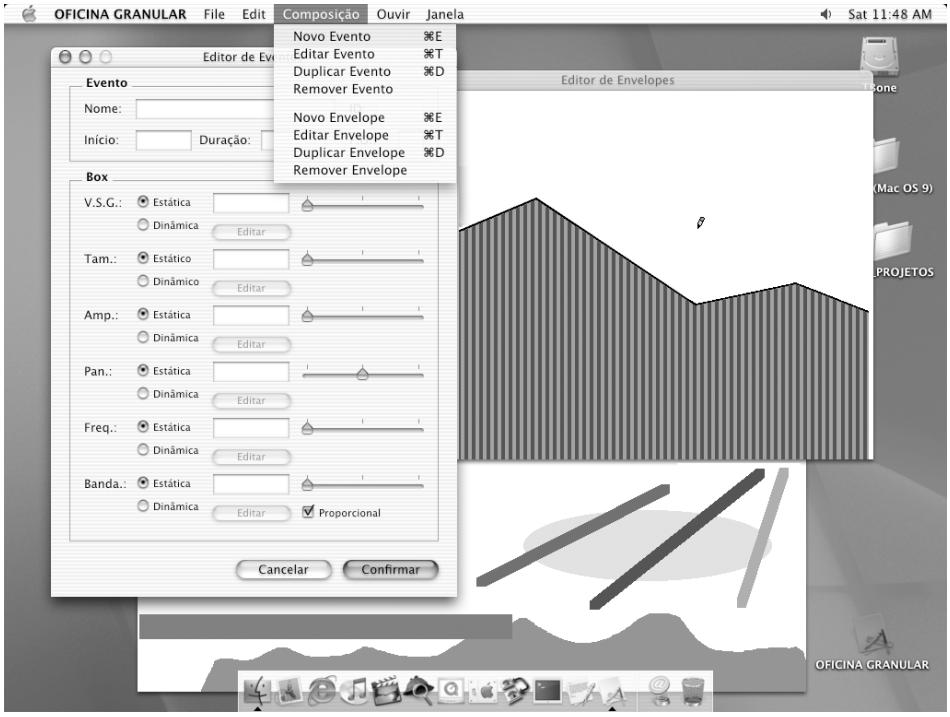

Fig. 1 - Graphical User Interface

References

Behles, Gerhard, Sascha Starke, and Axel Röbel. "Quasi-Synchronous and Pitch-Synchronous Granular Sound Processing with Stampede II". Computer Music Journal, Vol. 12, No. 2, Summer 1988.

Jones, Douglas & Thomas W. Parks. "Generation and Combination of Grains for Music Synthesis." Computer Music Journal, Vol. 12, No. 2, Summer 1988.

Roads, Curtis. Introduction to Granular Synthesis. Computer Music Journal, Vol. 12, No. 2, Summer 1988.

Roads, Curtis and J. Strawn. "Granular Synthesis of Sound" *in Foundations of Computer Music*. Cambridge Massachusetts: MIT Press, 1985, pp. 146-159.

Truax, Berry. *Real-Time Granular Synthesis with a digital Signal Processor*. Computer Music Journal, Vol. 12, No. 2, Summer 1988. Computer Music Journal, Vol. 12, No. 2, Summer 1988.