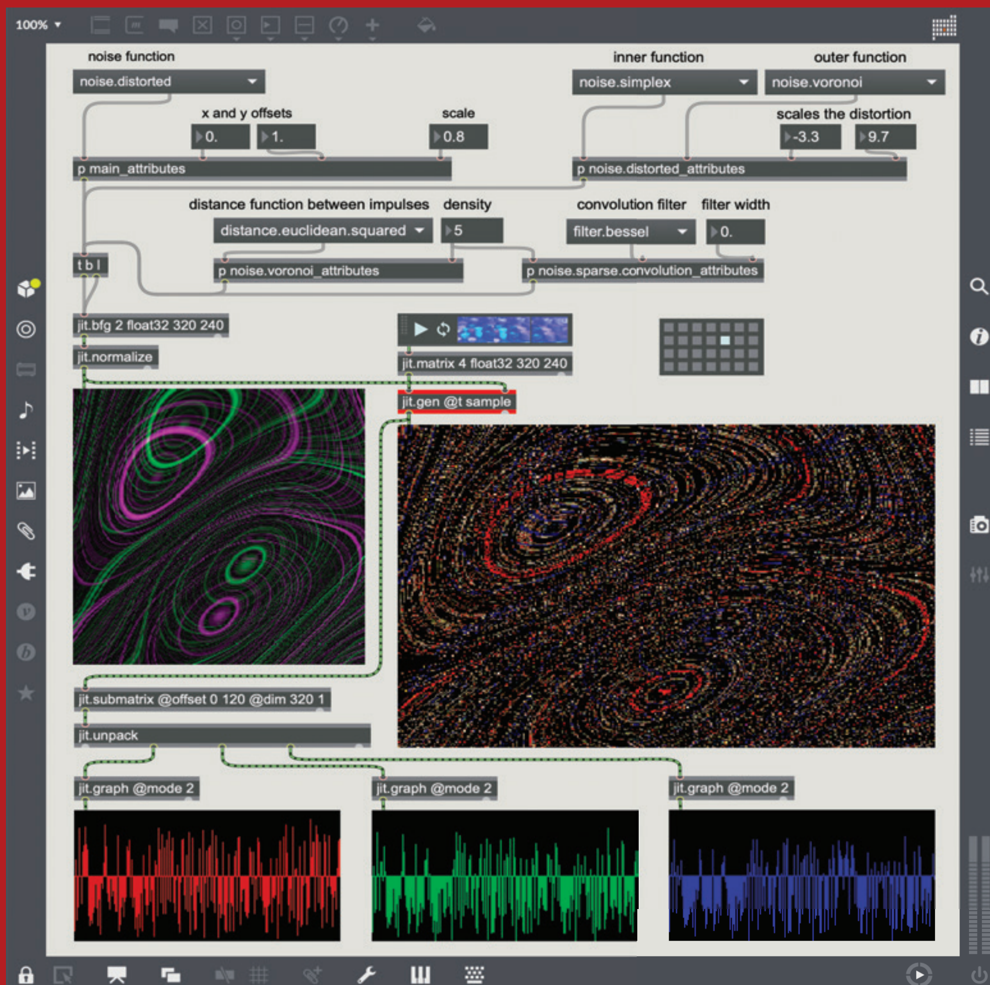
Alessandro Cipriani • Maurizio Giri



Electronic Music and Sound Design

Theory and Practice with Max 8 • volume 3

ConTempoNet

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Alessandro Cipriani • Maurizio Giri

This is a demo copy of

ELECTRONIC MUSIC AND SOUND DESIGN

Theory and Practice with Max - Volume 3

more info at:

www.contemponet.com

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

CIPRIANI, Alessandro - GIRI, Maurizio
Electronic Music and Sound Design : theory and practice with Max. Vol. 3. / Alessandro Cipriani, Maurizio Giri.
Includes bibliographical references and index.
ISBN 978-88-99212-24-7
1. Computer Music - Instruction and study. 2. Computer composition.

Original Title:
Musica Elettronica e Sound Design - Teoria e Pratica with Max vol. 3
Copyright © 2021 Contemponet s.a.s. Rome - Italy

© 2023 - Contemponet s.a.s., Roma
First edition 2023

Translation by Simone Micheli (first draft), Richard Dudas (Theory sections) and Benjamin Thigpen (Practice sections)

Figures produced by: Maurizio Refice

Products and Company names mentioned herein may be trademarks of their respective Companies. Mention of third-party products is for informational purposes only and constitutes neither an endorsement nor a recommendation. The use of general descriptive names, trade names, trademarks, etc., in this publication, even if the former are not especially identified, is not to be taken as a sign that such names, as understood by the Trade Marks and Merchandise Marks Act, may accordingly be used freely by anyone.

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from the publisher.

Contemponet s.a.s., Rome (Italy)
e-mail posta@contemponet.com
URL: www.contemponet.com

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

CONTENTS

Foreword by Carmine-Emanuele Cella • VI
Introduction • IX

Interlude F
AN INTRODUCTION TO GEN
LEARNING AGENDA
IF.1 The Gen environment
IF.2 Delay lines with Gen
IF.3 Subpatches and abstractions in Gen
IF.4 Data storage and management in Gen
IF.5 Sample and hold
IF.6 Rewriting MSP patches in Gen
IF.7 Boolean operators
IF.8 The `gen` object (without the tilde)
IF.9 The @expr attribute
IF.10 Gen and the multichannel system
List of Max objects
List of attributes for specific Max objects
List of Gen operators
Glossary

Chapter 10T - THEORY
REVERBERATION AND SPATIALIZATION
LEARNING AGENDA
10.1 Reverberation
10.2 The Schroeder reverberator
10.3 Freeverb
10.4 The Dattorro reverberator (plate reverb simulation)
10.5 FDN reverberator (Feedback Delay Network)
10.6 Creative uses of reverberation
10.7 Two-channel sound spatialization
10.8 Multichannel sound spatialization
Basic concepts
Glossary

Chapter 10P - PRACTICE
REVERBERATION AND SPATIALIZATION
LEARNING AGENDA
10.1 Introduction to reverberation algorithms
10.2 The Schroeder reverberator
10.3 Freeverb
10.4 The Dattorro reverberator (plate reverb simulation)
10.5 FDN reverberator (Feedback Delay Network)
10.6 Creative uses of reverberation
10.7 Two-channel sound spatialization
10.8 Multichannel sound spatialization
List of Max objects

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

List of messages for specific Max objects
List of Gen operators

Chapter 11T - THEORY
NONLINEAR SYNTHESIS
LEARNING AGENDA
11.1 Amplitude modulation techniques: AM, RM, and SSB
Basic concepts
11.2 Frequency modulation and phase modulation
Basic concepts
11.3 Phase distortion
11.4 Nonlinear distortion (NLD) or waveshaping
11.5 Wave terrain synthesis (WTS) **3**
11.6 Split synthesis
Basic concepts
Glossary

Chapter 11P - PRACTICE
NONLINEAR SYNTHESIS
LEARNING AGENDA
11.1 Amplitude modulation techniques: AM, RM, and SSB
11.2 Frequency modulation and phase modulation
11.3 Phase distortion
11.4 Nonlinear distortion (NLD) or waveshaping
11.5 Wave terrain synthesis (WTS)
11.6 Split synthesis
List of Max objects
List of attributes and messages for specific Max objects
List of Gen operators

Chapter 12T - THEORY
MICROSOUND
LEARNING AGENDA
12.1 Granular synthesis
12.2 Synchronous granular synthesis and formant synthesis
12.3 Asynchronous granular synthesis
12.4 Particle synthesis
12.5 Granulation and segmentation of sampled sounds
Basic concepts
Glossary

Chapter 12P - PRACTICE
MICROSOUND
LEARNING AGENDA
12.1 Granular synthesis
12.2 Synchronous granular synthesis and formant synthesis
12.3 Asynchronous granular synthesis
12.4 Particle synthesis

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

12.5 Granulation and segmentation of sampled sounds
List of Max objects
List of attributes and messages for specific Max objects
List of Gen operators and attributes

Chapter 13T - THEORY
ANALYSIS, RESYNTHESIS, AND CONVOLUTION
LEARNING AGENDA
13.1 The vocoder
13.2 The Fourier transform
13.3 Signal processing in the frequency domain: the phase vocoder
13.4 Time stretching and pitch shifting with phase vocoder
13.5 Convolution and cross-synthesis
13.6 Convolution reverb
Basic concepts
Glossary

Chapter 13P - PRACTICE
ANALYSIS, RESYNTHESIS, AND CONVOLUTION
LEARNING AGENDA
13.1 The vocoder
13.2 The Fourier transform
13.3 Signal processing in the frequency domain: the phase vocoder
13.4 Time stretching and pitch shifting with phase vocoder
13.5 Convolution and cross-synthesis
13.6 Convolution reverb
List of Max objects
List of attributes, arguments and messages for specific Max objects
List of Gen operators
Glossary

Interlude G
JITTER FOR AUDIO
LEARNING AGENDA
IG.1 Introduction to jitter
IG.2 Numerical operations with matrices
IG.3 Displaying audio signals in jitter
IG.4 Processing audio signals using matrices
IG.5 The jit.expr object
IG.6 The jit.bfg object
IG.7 Jit.gen
IG.8 The Fourier transform and the jit.fft object
List of Jitter objects
List of attributes and messages for specific Jitter objects
Glossary

References
Index

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

FOREWORD TO THE THIRD VOLUME
by Carmine-Emanuele Cella

Writing a foreword to the third volume of *Electronic Music and Sound Design* is, in a way, an unnecessary exercise.

The monumental work of Alessandro Cipriani and Maurizio Giri is, *de facto*, the reference for today's electronic music production: as such, it really needs no introduction. There is, to my knowledge, no electronic music course in Italy that does not refer to their books. And many foreign educational institutions also use them in some way. I myself, at the Center for New Music and Audio Technologies (CNMAT) at UC Berkeley, often turn to the first two volumes for examples and useful strategies for explaining concepts.

Explaining is, in fact, the fundamental project of the two authors. There is no shortage of books in the world that seek to demonstrate the erudition of their authors. It is harder, however, to find books that focus on the readers – taking them on a journey that will ultimately change them. The books by Cipriani and Giri belong to this rare category: they are books that *explain*.
With their smooth – yet never trivial – style, the authors guide the readers step by step through complex notions that are broken down into small, easy-to-understand chunks of information. Their approach is encyclopedic: over the course of several volumes, they cover all the key concepts of electronic music. Page after page, the authors lay the foundations of this heterogeneous field, and they do so in a measured and methodical way, calling to mind at times the style of Aristotle.

It is superfluous, therefore, to write a foreword to such a work. But it may be useful for me to try.

Like its predecessors, this third volume follows the tried and tested formula of theory and practice: first a presentation of the concepts involved, and then a demonstration of their practical realization – or as we often say today, their implementation. Although the word "implementation" may sound rather crude and unrefined, it has an important root: it comes from the Latin *in-plere*, which means to fill in or complete the interior.
Once an abstract concept has been presented, it is *filled in*, so to speak, with a concrete example: a program in a specific programming language. The language in question is obviously Max, currently considered the state of the art in computer music. But don't be fooled: the texts of Cipriani and Giri are not limited by their implementations; they will not grow old and end up gathering dust on library shelves. On the contrary, in fact, the implementations they present add new details to the concepts and merge with them. This is one of the book's strong points, in my opinion.

Returning to the Aristotelian metaphor, the formal coupling of theory and practice is somewhat reminiscent of the division between *Metaphysics and Physics* — a division which is, in fact, purely imaginary. In reality, everything

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

finally blends into a conceptual unity that encompasses the understanding of the object of study.

The concepts presented in this third volume are authentic, important and always of great interest to those who care about this field. They include sound spatialization, nonlinear synthesis, granular synthesis and a topic that is of particular interest to me, transformations in the frequency domain.

In a sense, the motif unifying all the chapters is *representation*. This term has a precise meaning from a mathematical point of view. Representing a signal means *projecting* it into a space in which some specific properties are highlighted. The projection is created by calculating the scalar product between the original signal and an appropriate set of functions known as the *kernel*. The quintessential representation is that of Fourier, in which the kernel consists of a set of sine waves and in which the frequency of the signals is the highlighted aspect (Chapter 13). However, it is possible to generalize this type of representation using any type of kernel. For instance, another important representation is the one in which both time and frequency are made explicit using a kernel consisting of sine waves that have a specific duration. This representation – known as that of Gabor – is the foundation of both granular synthesis and the micro-temporal approach to signal processing (Chapter 12). When the projection is made with kernels that exhibit other more specific properties, we may speak of modulation and nonlinear distortion (Chapter 11). In addition, Chapter 10 deals with spatialization. For the first time ever, all the algorithmic techniques for reverberation are grouped together and united by a single programming style. It is thereby possible to work out *why* Schroeder's algorithm sounds bad or why feedback delay networks are difficult to manage. I personally do not think one can do better than this pedagogically. Finally, as in a musical form, the book is enriched by *Interludes*. In these interludes, the authors reveal the key elements of Gen (a language internal to Max, useful for building more efficient algorithms) and *Jitter* (the subset of functions supporting video and matrix processing).

The book is accompanied by a collection of digital supporting materials, including patches and sound examples. The latter are of paramount importance for an understanding of the concepts introduced, particularly the more advanced concepts based on the Fourier transform (Chapter 13). By simply listening, one can come to understand the difference between bin shifting and spectral LFOs, or between spectral freeze and spectral warping. Creating good sound examples is a fine art, and in this area as well the authors succeed in being both effective and compelling.

In conclusion, the third volume of *Electronic Music and Sound Design* is a kaleidoscopic catalog of ideas and applications for analyzing, synthesizing, and transforming signals in a wide variety of ways.
It makes an important effort in the contemporary world, filling the pedagogical void created by the lack of books that *explain*. Anyone who has taught electronic music of any sort is aware of the heterogeneity of the discipline's

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

typical audience: composers, DJs, sound designers, media artists. When faced with such diversity, it is difficult to communicate with everyone without becoming superficial. Cipriani and Giri succeed in addressing everyone without weakening the theoretical basis and without unnecessary specializations – achieving a masterful balance of comprehensibility, functionality, and breadth.

To use an expression from the United States, I would say that this work is *larger than life*. And quite frankly, it’s just what we needed.

Carmine-Emanuele Cella
Assistant professor in Music and Technology
Center for New Music and Audio Technology (CNMAT) University of California, Berkeley

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

INTRODUCTION TO THE THIRD VOLUME

This is the third in a series of volumes on synthesis and digital sound processing. The work therefore also includes

- a first volume that covers several topics, including additive synthesis, noise generators, filters, subtractive synthesis, and control signals;
- a second volume that covers topics including digital audio, dynamics processors, delay lines, the MIDI protocol and real-time processing, Max for Live, and a chapter on the art of organizing sound.

PREREQUISITES

All the volumes are composed of alternating sections on theory and computer practice; they should be studied in close conjunction. This third volume can be used by advanced users who have a firm grasp of the concepts and Max practices outlined in the first two volumes. For this reason, where the understanding of a certain concept is taken for granted in this volume, the text often indicates the section of a previous volume in which the topic has been discussed.
The contents of this book can be studied either through self-learning or with the guidance of a teacher.

SOUND EXAMPLES

The theory chapters are accompanied by many sound examples which are available on the support page. By listening to these examples, one can have an immediate experience of the sound, its creation and processing, before tackling the practical work of programming. In this way, the study of theory is always connected with the perception of sound and with its possible modifications.

MAX

The parts of the book devoted to practice were realized using the software Max 8, which can be downloaded at www.cycling74.com. Patches, sound files, library extensions (the Virtual Sound Macros), and other supporting materials for the practical activities can be found on the support page

PEDAGOGICAL APPROACH

Electronic Music and Sound Design is not just a book but rather a complex, nonlinear educational system, which requires the interaction between knowledge, perception, skills, experience, self-assessment, and creativity.

More specifically, the system is based on concrete solutions to overcome the following limitations:

a) the separation between theory and practice;
b) the separation between technical knowledge and the art of organizing sounds;
c) the separation between theory and perception;
d) the absence of active participation of the student in the learning process.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

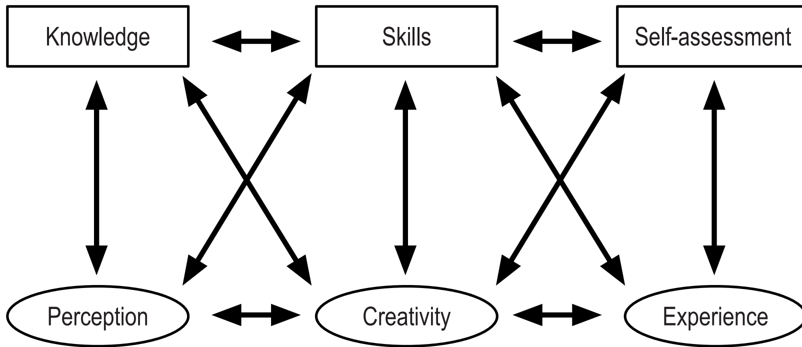


Fig. I Diagram of the educational system

Thus, the readers will find themselves interacting with a multidimensional structure where the software, the theory, the sound examples and the algorithms form a whole.

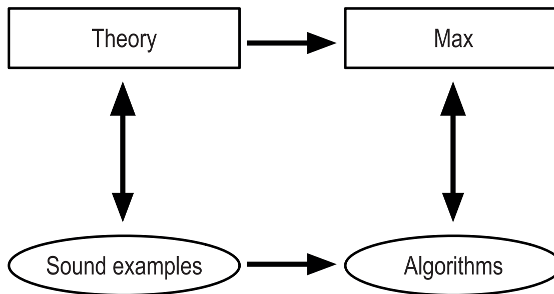


Fig. II The nonlinear structure of the learning system

For this reason, when engaging in this third part of the path, the users will be able to connect the knowledge and skills they have already acquired in the fields of programming and listening analysis with their own creativity. At this point in the itinerary, having acquired a good level of competence, users will gradually be able to invent for themselves new possibilities of interaction among the described techniques. The element of creativity will be of great importance, enabling one to develop one's own field of research and invention. At first, it was necessary to provide context-free rules for the reader to follow in order to achieve immediate goals and to build confidence through the use of systematic procedures. However, as new knowledge and skills are gained, we feel that it is especially important to promote context-based experiences, to develop critical thinking, and to increasingly encourage the use of individual perception and creativity.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

CONTENTS AND RECOMMENDATIONS FOR LEARNING

Like its predecessors, this volume should be studied by alternating the theory chapters with the corresponding chapters on practice – without neglecting the computer activities and a careful listening to the sound examples.
After working through the second volume, our typical reader will have further developed his or her skills. So, we are now addressing an advanced user, one who understands all the techniques discussed in the previous volumes and is able to program them. Therefore, some steps (in theory, and especially in practice) that we assume have been acquired are not as detailed as in the first two volumes. For this reason, we often include references to the previous volumes to refresh the reader's memory on certain concepts or techniques.
Regarding programming, there are two new additions to this volume, which are discussed in Interludes F and G. The first concerns the **Gen** environment. Gen is an extension of the Max development environment that allows us to create much more efficient algorithms than those that can be programmed using just Max. Indeed, the Gen environment enables the creation of patches that are immediately compiled into executable machine code, which processes one sample at a time instead of groups of samples as in MSP. Interlude F is both a tutorial for this environment and a description of many of the new possibilities afforded by Gen. In subsequent chapters, we will often use algorithms created in Gen, both because of its greater speed and in order to take advantage of the special features of the extension.

Interlude G, on the other hand, deals with the use of **Jitter** to control, visualize, process, and generate audio signals.
Jitter, which is an extension of Max, can really do much more than this: it can manage images, video and 3D graphics in complex ways. But the purpose of this volume, as of the previous ones, is to examine the techniques related specifically to the synthesis and processing of sound. Consequently, after an introduction to the Jitter environment – including, briefly, the management of images and videos – Interlude G focuses on the creation of matrices and data sets for controlling and interacting with audio.

After the Interlude F, Chapter 10 plots a theoretical and practical course through the various techniques for simulating **reverberation** using delay line algorithms. Starting with historical methods such as that of Schroeder, the discussion moves on to the techniques of Freeverb and the Dattorro reverberator, and ends with an examination of feedback delay network reverberation. The same chapter also covers basic stereo and multichannel **spatialization** techniques.

The next chapter focuses on various **nonlinear synthesis** techniques, i.e., synthesis and sound processing procedures that produce many output components that are not part of the original input signal, resulting in spectral shifting and enrichment. Specifically, we discuss the theories and techniques of amplitude, ring, and single-sideband modulations, frequency and phase modulations, feedback PM, phase distortion, nonlinear distortion, and the complex, yet extremely interesting, wave terrain synthesis.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Chapter 12 deals with **microsound**: sound events of extremely short duration. The techniques in this area of computer music range from synchronous and asynchronous granular synthesis to formant synthesis. Furthermore, several particle synthesis techniques described by Roads in his book *Microsound* will be implemented in Max – such as, for example, glisson synthesis, grainlet synthesis, trainlet synthesis, and pulsar synthesis.
In the second part of this chapter, we also discuss sound processing techniques such as granulation of sampled sounds and multi-source brassage.

Chapter 13 is very comprehensive: the first part focuses on **analysis-based synthesis** techniques; the second, on **convolution**. In particular, we describe various techniques for constructing both classical and STFT-based vocoders. The most substantial part of this chapter deals with the theory of the Fourier transform and with various processing techniques in the frequency domain. These techniques, among which there are huge differences, can be quite useful for creating interesting musical processing. Starting from the theory and technique of the phase vocoder, we move on to brickwall filters, multiband and random filters, spectral LFOs, bin shifting, cross-synthesis by STFT, bin feedback delay, freeze, etc. The second part deals with cross-synthesis between two signals by convolution, convolution with microsounds, and convolution reverbs – with details concerning the differences between direct convolution and so-called "fast convolution."

Returning to Interlude G, which concludes the book, Jitter allows us, among other things, to put some of the previously described techniques into practice more efficiently. Thus, it will be useful for the reader to delve into the techniques of this extremely interesting extension of Max, which has existed for many years and has been constantly evolving.

Comments and suggestions
Corrections and comments are always welcome.
Please contact the authors via e-mail at:
a.cipriani@edisonstudio.it (www.edisonstudio.it/alessandro-cipriani/) or
maurizio@giri.it (www.giri.it).

ACKNOWLEDGEMENTS

We wish to thank Maurizio Argentieri, Andrew Bentley, Daniel Biro, Diego Capoccitti, Emanuele Casale, Luigi Ceccarelli, Marco Cento, Sofia Cipriani (for the violin sounds), Agostino Di Scipio, Edison Studio, Samuele Grippo, Paul Lansky, Marco Massimi, Curtis Roads, Stefano Taglietti, Barry Truax, Teresa Vasselli, and Trevor Wishart.
A special thanks goes to Carmine-Emanuele Cella for his valuable advice, especially regarding to the chapter on analysis and resynthesis. We would also like to thank Vincenzo Core, Richard Dudas and Benjamin Thigpen for their attentive reading and analysis of this volume, and for their detailed and precise observations.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

DEDICATIONS

This volume is dedicated to Massimo and Francesco Cipriani, and to Alex Giri.

Enjoy making use of this volume!
Alessandro Cipriani and Maurizio Giri

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Interlude F
AN INTRODUCTION TO GEN

IF.1 THE GEN ENVIRONMENT
IF.2 DELAY LINES WITH GEN
IF.3 SUBPATCHES AND ABSTRACTIONS IN GEN
IF.4 DATA STORAGE AND MANAGEMENT IN GEN
IF.5 SAMPLE AND HOLD
IF.6 REWRITING MSP PATCHES IN GEN
IF.7 BOOLEAN OPERATORS
IF.8 THE GEN OBJECT (WITHOUT THE TILDE)
IF.9 THE @EXPR ATTRIBUTE
IF.10 GEN AND THE MULTICHANNEL SYSTEM

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

LEARNING AGENDA

PREREQUISITES FOR THE CHAPTER
- The contents of Volumes I and II

OBJECTIVES
Knowledge
- To know the primary characteristics of the Gen environment

Skills
- To be able to program and use algorithms constructed in Gen
- To be able to use Gen operators for managing delay lines
- To be able to create algorithms using subpatches and abstractions in Gen
- To be able to rewrite MSP patches in Gen
- To be able to write short functions in Gen using the *GenExpr* code
- To be able to create algorithms using Gen multichannel operators

Competence
- To be able to realize a brief etude based on creative uses of Gen

ACTIVITIES
- Constructing and modifying algorithms

SUSSIDI DIDATTICI
- List of Max objects - List of attributes for specific Max objects - List of Gen operators - Glossary

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

IF.1 THE GEN ENVIRONMENT

Before beginning to study this volume, we recommend that you download and install the latest version of the Virtual Sound Macros library. You will find it, along with all the supporting materials for the third volume, on the support page.

As we mentioned in the Introduction, this volume will make ample use of algorithms constructed in Gen. They are necessary for the implementation of various synthesis and sound processing techniques that we will present in the course of the book.
In this interlude, we present the primary characteristics of the Gen environment. In later chapters, we will develop this topic further, as the need arises.

First of all, let's try to understand what advantages there are to programming in Gen rather than MSP.
Gen is a development environment internal to Max with which it is possible to create patches that are immediately compiled into an executable machine code.[1]
In practice, a patch made in Gen is equivalent to a native Max MSP object, such as **cycle~** or **biquad~**. This makes it possible to create algorithms that would be extremely inefficient if programmed simply in Max.
Another advantage of programming in Gen is that the audio chain in a Gen patch is not processed using vectors defined by the Signal Vector Size (as it is in MSP) but one sample at a time. This makes it possible, for example, to create a one-sample delay with feedback - which allows us, among other things, to build filters with designs that are different from that of **biquad~**. As we will see, this feature is essential for the realization of numerous algorithms that operate at audio rate.
Finally, by sending the "exportcode" message to the **gen~** object, we can export the patch as C++ code, which can be used to create applications outside of the Max environment.

The patching environment is very similar to the one we already know: there are graphic objects connected by virtual patch cords, and the graphical appearance is absolutely identical to that of the "normal" Max environment.
A Gen patch can be inserted (as a subpatch or an abstraction) within five special objects: **gen~**, **gen**, **jit.gen**, **jit.pix**, and **jit.gl.pix**.
As the name suggests, the first object (**gen~**) allows us to create Gen patches for audio processing; the second (**gen**) is a control rate version of the first (we will talk about it in section IF.8), and the last three objects are used to create algorithms for processing matrices, images, and textures in Jitter.[2]

[1] That is, code that can be directly executed by the processor.
[2] Jitter is an extension of Max used to process data matrices, images, and videos. In Interlude G, we will consider its uses for managing and interacting with audio - at which time we will also discuss the object **jit.gen**.

In this interlude, we will deal exclusively with patches for audio processing or control signals made within **gen~** and **gen**.
Let's take a look at a first, very simple, patch. Open the file **IF_01_firstGen.maxpat** (figure IF.1).

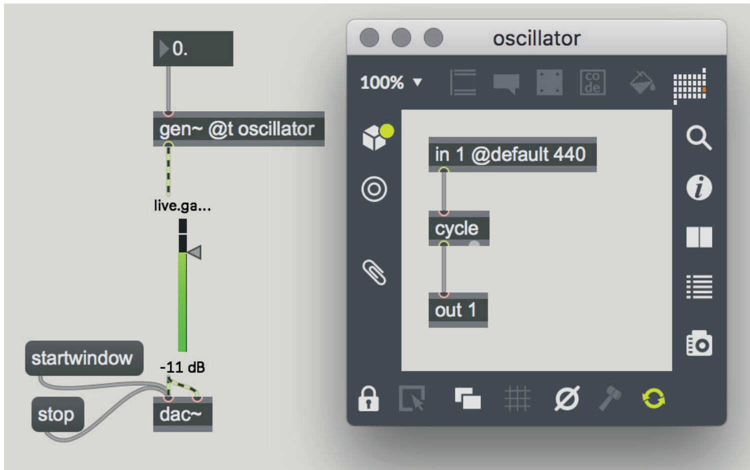


Fig. IF.1 The file **IF_01_firstGen.maxpat**

Double-click on the **gen~** object which is on the left side of the image to open the subpatch to the right. We see, first of all, that the **gen~** object has an attribute, @t (the abbreviation for @title), used to specify the title of the subpatch. Now activate the patch by clicking on the *startwindow* message: if you did not modify the value of the number box connected to **gen~**, you should hear a sine wave at 440 Hz.
Inside the subpatch "oscillator," we have an inlet [**in** 1 @default 440], very similar to the inlets of the **poly~** object; then a **cycle** sine wave oscillator; and finally an outlet [**out** 1], also similar to the outlets of a **poly~** object.
Clearly, for anyone who knows MSP, this simple Gen patch is perfectly comprehensible: the only differences between this and an analogous MSP program are that the names of the "objects" have no tilde (**cycle** instead of **cycle~**) and the cables are gray instead of yellow-black.
These differences are due to the fact that inside a **gen~** patch, there are only signals: there are no asynchronous Max messages, let alone lists or strings. So it is not necessary to use suffixes or specific cable colors to differentiate signals from other messages. Another reason is that this allows us to distinguish **gen~** operators from the corresponding MSP objects of the same name (and there are many, as we will see).
Note also that most Gen "objects" are called *operators*: we can differentiate the Gen *operator* **phasor**, therefore, from the corresponding MSP object **phasor~**.
The fact that there are only signals circulating within a **gen~** patch has several consequences, not all of which are obvious. For example, if we add an argument to an operator, the inlet corresponding to that argument disappears. Try adding the argument 440, for instance, to the **cycle** operator in the patch

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

we opened: you will see its inlet disappear. This happens because if you were to connect a signal to `cycle`, the argument would immediately be nullified and replaced by the value of the signal, thus rendering it useless.
But if there is no argument in the `cycle` operator shown in the figure, why did we hear a sine wave at 440 Hz when we turned on the DSP engine? It is because we have assigned the attribute "@default 440" to the inlet `in`, which causes it to generate a constant signal of value 440 if it does not receive an external value.
If you now try changing the value of the number box in the main patch, the oscillator's frequency will change accordingly.
Both signals and generators of Max numerical values can be connected to the inlets of a `gen~` object; the latter are converted into signals within the `gen~` patch. Audio signals are passed to the `gen~` object a number of times per second equal to the sample rate, while Max numeric values are passed once every Signal Vector (so, for example, every 32 or 64 DSP cycles; see section 5.1P of the second volume). This allows us to save a bit of CPU in cases when it isn't necessary to obtain a new value every cycle.
If we know that a certain parameter for a `gen~` patch will only be sent via Max messages, we can use the param operator, which allows us to send external parameters using names (parameters are also sent once per Signal Vector). Let's look at an example. Open the patch **IF_02_param.maxpat** (figure IF.2).

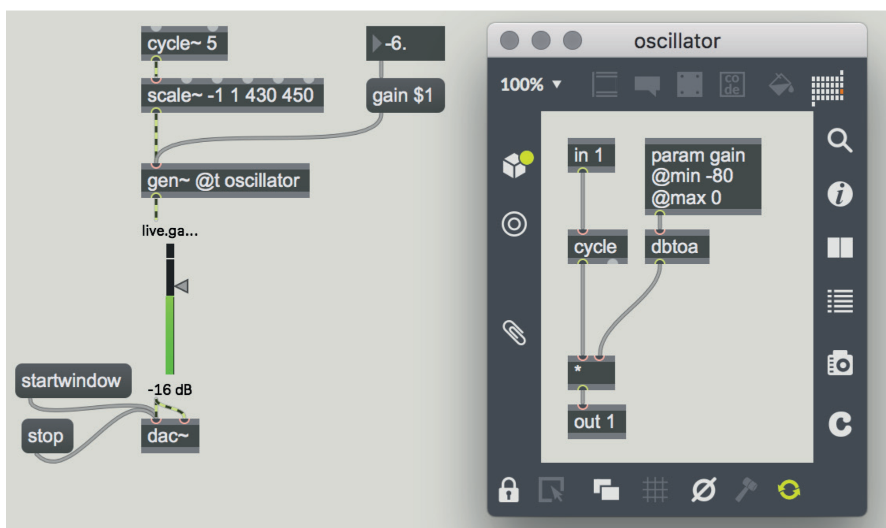


Fig. IF.2 The file **IF_02_param.maxpat**

In the `gen~` subpatch, we have added a **param** operator and given it the name "gain." If we now send the `gen~` object a message consisting of the string "gain" followed by a numeric value, that value will be passed to the `param` "gain" (see the main patch on the left side of the figure).
In this second patch, we added a vibrato to the frequency value sent to the internal oscillator in order to demonstrate that it is possible to send both signals and messages for `param` operators through `gen~` inlets.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

The value for "gain" is expressed in dB (limited between -80 and 0 by the two attributes that we added to **param**)[3] and converted into amplitude by the operator **dbtoa**.
Another advantage of **param** is that the parameter it defines becomes an attribute of **gen~**, and it is possible to set the initial value of a parameter using the familiar syntax @*attribute_name*. For example, try modifying the **gen~** object shown in figure IF.2 as follows:
[**gen~** @t oscillator @gain -30]
Now the initial amplitude of the oscillator will be equal to -30 dB.

You may have noticed that the Toolbars around the patcher window of a Gen subpatch are somewhat different from those that we already know; let's take a look at some of the differences.
The palettes have disappeared from the upper Toolbar: indeed, we cannot add interface objects to a Gen patch, only operators. The only exceptions, which are found in the Toolbar, are comment boxes and panels, which as you may recall, are used to create colored areas useful for grouping objects and for making a patch clearer.
There is also a new object, called **codebox**, which allows us to write *GenExpr* code. What is that? When we build a patch in Gen, the system creates a textual code in the *GenExpr* language (a language created specifically for Gen, which looks much like a very simplified version of the language C); and this textual code is then compiled into native machine code. You can see the *GenExpr* code generated by a patch by clicking on the C-shaped icon in the right Toolbar.
For example, this is the code generated by the Gen patch in figure IF.2:

```
Param gain(0, max=0, min=-80);
dbtoa_1 = dbtoa(gain);
cycle_2, cycleindex_3 = cycle(in1);
mul_4 = cycle_2 * dbtoa_1;
out1 = mul_4;
```

Is it possible to write directly in *GenExpr*? Yes, by using the **codebox** object, which can be found in the upper Toolbar. A discussion of *GenExpr*, however, is beyond the scope of this interlude.[4]

[3] In order to know the attributes of an operator, simply type "@" within the operator itself: an autocompletion menu will pop up displaying all the available attributes. It is also possible to display the operator's help and Reference Page by right-clicking in edit mode, exactly as one does for Max objects. An operator's help, however, does not open an executable patch like a Max object's help does, but rather a small "bubble" containing a description of the operator.
[4] We will nonetheless make some allusions to the syntax of *GenExpr* in section IF.9, when we discuss the attribute @expr.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

In the lower Toolbar, there are two new icons, shown in figure IF.3.

Fig. IF.3 Gen icons

The icon on the left (the two arrows forming a circle) enables and disables the auto-compiling function. When auto-compiling is active, new *GenExpr* code is immediately generated whenever the patch is modified. This option is active by default; it may be useful to deactivate it when working on a very large patch, so that the continuous recompilation of the code does not slow down the programming process.
The icon on the right is used to manually launch the compilation when the auto-compiling function is disabled.

As we have already mentioned, the patches loaded in **gen~** are executed at audio rate. Naturally, this makes programming in Gen more complex than programming in Max, where one can work simultaneously with asynchronous messages (like those generated by interface objects), timed control messages (such as the stream of bangs produced by the **metro** object) and signals.
On the other hand, the Gen environment has some features not found in Max that can make patches easier and more compact. Let's take a look at some of them.

There are several predefined variables and constants that make calculations easier: for example, *pi* (π), *twopi* (twice π), samplerate (the current sample rate) and others that we will see in the next few chapters. The interesting thing is that these values can be used directly as arguments: if we want to multiply a value by *pi*, we can simply use the operator [* pi].
For example, to create a sine wave oscillator using the **sin** operator, which generates the sine function and needs to receive the phase in radians as input, we can create a Gen patch like the one shown in figure IF.4.

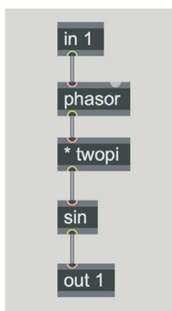


Fig. IF.4 A sine wave generator

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

We can also use expressions as arguments, utilizing the operators and mathematical functions that are available in Gen to create new constants.[5] For example, to calculate the cube root of an input value, in Gen we can use [pow 1/3], while in Max we have to write [pow 0.333333], probably resulting in a loss of accuracy.
It is also possible to include the name of an input (in1, in2, and so on) in an argument. For example, try making a patch with a **gen~** object containing the subpatch shown in figure IF.5.

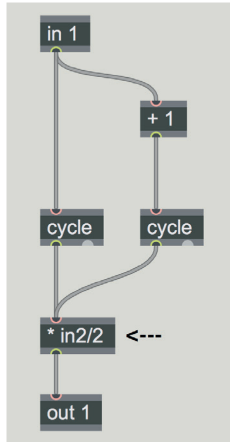


Fig. IF.5 Using an inlet as an argument

This patch generates two sine waves whose frequencies differ by 1 Hz (thus producing one beat per second); the two sine waves are added together and multiplied by the amplitude value coming from the second inlet. This amplitude value is also halved to prevent the sum of the two sine waves from exceeding the -1/1 limit of the DA converter. Note that there is not even any need to make an [in 2] object; its presence in the multiplier's argument automatically creates a second inlet in the **gen~** object.
The parameters passed through the **param** operator can also be used as arguments. See figure IF.6.

In this case, you need to create a **param** operator with the same name as the parameter (shown on the upper right of the figure). And of course, in the main patch, you need to send the amplitude value preceded by the name of the parameter "amp."

[5] Most mathematical functions and operators in Gen are identical to those found in Max and MSP. For a list of all available functions and operators, see the "Gen Common Operators" guide in the documentation (which can be accessed by selecting Reference from the Help menu).

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

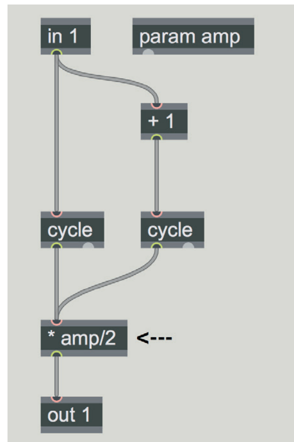


Fig. IF.6 Using a parameter as an argument

(...)

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

other sections in this chapter:

IF.1 THE GEN ENVIRONMENT

IF.2 DELAY LINES WITH GEN

IF.3 SUBPATCHES AND ABSTRACTIONS IN GEN

IF.4 DATA STORAGE AND MANAGEMENT IN GEN

IF.5 SAMPLE AND HOLD

IF.6 REWRITING MSP PATCHES IN GEN

IF.7 BOOLEAN OPERATORS

IF.8 THE GEN OBJECT (WITHOUT THE TILDE)

IF.9 THE @EXPR ATTRIBUTE

IF.10 GEN AND THE MULTICHANNEL SYSTEM

ACTIVITIES
• Constructing and modifying algorithms

TESTING
• Integrated cross-functional project: reverse engineering

SUPPORTING MATERIALS
• List of Max objects - List of attributes for specific Max objects - List of GEN operators - Glossary

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

10T
REVERBERATION AND SPATIALIZATION

10.1 REVERBERATION
10.2 THE SCHROEDER REVERBERATOR
10.3 FREEVERB
10.4 THE DATTORRO REVERBERATOR (PLATE REVERB SIMULATION)
10.5 FDN REVERBERATOR
10.6 CREATIVE USES OF REVERBERATION
10.7 TWO-CHANNEL SOUND SPATIALIZATION
10.8 MULTICHANNEL SOUND SPATIALIZATION

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

LEARNING AGENDA

PREREQUISITES FOR THIS CHAPTER
- Contents of volumes I and II

OBJECTIVES
Knowledge
- To learn the basics of reverberation theory
- To learn the basics of the Schroeder and Schroeder-Moorer reverb techniques
- To learn the basics of the Freeverb reverberator technique
- To learn the basics of the Dattorro reverberator technique
- To learn the basics of the FDN reverberator technique
- To learn the basic concepts of stereo and multichannel spatialization

Skills
- To be able to hear and identify the ways that certain reverb parameters (decay time, room volume, dry/wet balance) affect the sound
- To be able to hear and identify the Doppler effect
- To be able to hear and identify, when using a quadraphonic or 5.1 surround sound system, the rotation effect, the alternation between channels, the distance localization, the direction of a sound, the differences between global and local reverb, the artificial distance localization of the sound beyond the speakers

CONTENTS
- Basic theory of reverberation in the acoustic field and digital domain
- Basic theory of Schroeder, Schroeder-Moorer and Freeverb reverberators
- Basic theory of Dattorro and FDN reverberators
- Basic theory of stereo and multichannel spatialization

ACTIVITIES
- Sound examples

SUPPORTING MATERIALS
- Basic concepts - Glossary

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

10.1 REVERBERATION

This chapter will introduce the complex topic of reverberation or, more precisely, the methods and techniques which can be used to recreate acoustic reverberation using algorithms. Specifically, we will focus on reverb algorithms created using delay lines. In chapter 6.6 we already mentioned that comb filters can be used in conjunction with allpass filters to simulate reverberation – this is just one of the many possible techniques available. Reverb algorithms can also be designed using convolution, but reverb algorithms based on this technique will be analyzed separately in chapter 13.
The phenomenon of natural reverberation arises in much the same way as a natural echo: the sound is reflected by various surfaces within the environment where it originates. As was explained in section 6.2, a reflection can be perceived as an echo if the distance between the sound source and the reflective obstacle produces a delay that is greater than the so-called *Haas zone* (i.e., 25-35 ms, which may vary depending on the timbre, envelope, and frequency of the sound). If the delay time is shorter than this value, the direct sound and the reflections are generally perceived as a single sound. For example, in spaces with more than one reflective surface, such as a *parallelepiped* – a three-dimensional "cube" whose sides are parallelograms – the production of many close reflections with different short delay times results in a reverberation phenomenon, instead of a series of echoes. The listener will tend to perceive these repetitions as a single sound event because of the large number of reflections and the close proximity in time between these reflections and the initial sound. Of course, the type of reverberation will depend on a combination of many factors including the material(s) which the room surfaces (and any additional obstacles within the room) are made of, the shape and size of the room itself, the distance between the sound source and the various room surfaces, and – from a perceptual viewpoint – even the listener's own physical distance from the sound source and room surfaces.

We can simulate this phenomenon algorithmically by using delay lines with feedback and various types of filters. Over the history of artificial reverberation design, several algorithms have been put forward for the purpose of simulating reverberation. In this section, we will analyze the most important ones. Let us first look at the various stages which make up a generic reverb effect. The time between the direct sound and the first reflected sound is called the **pre-delay**, and some algorithms allow you to set this parameter separately. In the digital domain, this is defined as the time between the direct sound and the first reflected sound irrespective of human perception.[1] After this time, a series of **early reflections** occurs. Since reflections travel for a longer distance than direct sound, they arrive later. As the name suggests, early reflections are the first repetitions of the sound to reach the listener. They can still be perceived separately, especially if the sound source is an

[1] Note that in a real acoustic environment, we must also consider the latency between the generation of the sound and the moment the sound reaches the listener.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

impulse or a percussive sound. In order to implement these reflections in an algorithm, the input signal is repeated multiple times to recreate the adequate timbral differences for each repetition and simulate the signal response of a specific virtual environment. In real acoustic spaces, the more times a sound is reflected, the weaker it becomes. This is because the surface of the obstacle reflecting the sound also absorbs part of the energy of each repetition. As well as experiencing a reduction in intensity, each reflection is altered progressively in timbre. Depending on the quality and shape of the materials the obstacles are made of, more energy is absorbed in specific frequency ranges than in others. This behavior is generally similar to that of a lowpass filter.
You can simulate a smaller or larger enclosed space by adjusting the pre-delay time and the times between the early reflections. Each reflection is a filtered and delayed copy of the original sound source. So, we can state that early reflections provide most of the information about the acoustic space and its characteristics. (See fig. 10.1)

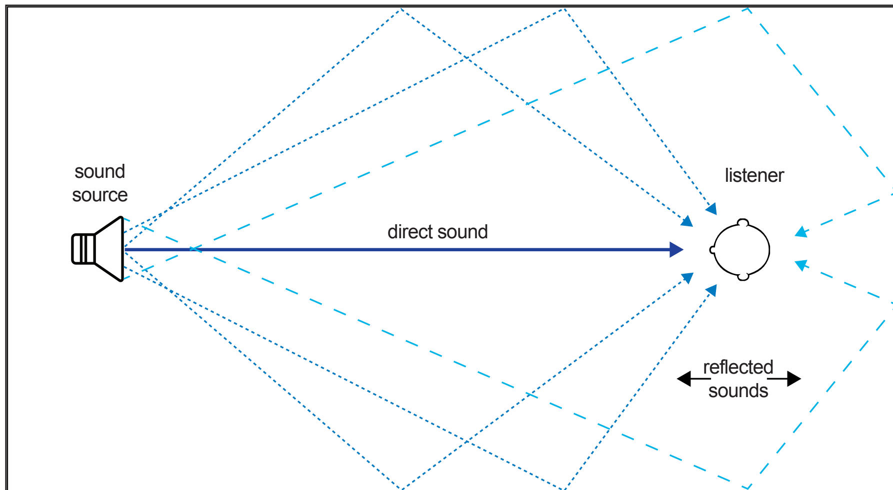


Fig. 10.1 An overview of direct and reflected sounds

Subsequently, further reflections, of which there can be thousands, arrive at the listener. These repetitions blend together, resulting in a progressive accumulation of reflections. This results in an initial increase in the overall amplitude, followed by an extended decay in amplitude over a certain period in time. This creates the sensation of a longer sound and a change in the timbre of the sound itself. We shall refer to this stage as ***late reverberation***.[2]

[2] "Late reverberation" is often referred to as "reverb tail."

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

When the sound ends, the reverberation will continue for some time before it finally dies away. Generally speaking, **reverberation time** (or **reverb time**, hereafter abbreviated to **RT60**) is defined as the time that the reverberated signal takes to decrease to $1/1000^{th}$ of its initial amplitude (that is, the time it takes to attenuate by 60 dB.[3]
In general, the decay time is inversely proportional to the absorption coefficient of the obstacles in the room (including – and especially – the ceiling, walls, and floor). At the same time, it is proportional to the room size. Thus, the physical characteristics of the obstacles and the dimensions of the environment are two key elements in designing the proper calculations to simulate reverberation.

Reverberation also provides us with information on the distance of a sound from the listener. In a more or less reverberant environment, the perception of the distance of a sound source from the listener depends on the **R/D ratio** (i.e., the ratio between the reverberated sound and the direct sound). For closer sources, the direct sound is higher in amplitude. Conversely, for distant sounds, the amplitude of the reverberated sound is greater than that of the direct sound. Direct sound decreases in amplitude faster than the reflected sound when the R/D ratio is high. We will take a more detailed look at the concepts concerning auditory localization in section 10.6.

SOUND EXAMPLE 10.1 • direct sound, early reflections, and late reverberation

a) Direct sound (*dry*)
b) Early reflections
c) Direct sound + early reflections
d) Late reverberation
e) Direct sound + late reverberation without early reflections
f) Early reflections + late reverberation without the dry signal
g) Direct sound + early reflections + late reverberation

Note that the reverberation in example e) sounds somewhat unnatural. That is because there are no early reflections as if the sound source were outside a cave, a few meters away from the entrance, and as if the microphone were close to the source. Sometimes, when you want to hear both the dry sound and

[3] *«Reverb time is one of the parameters that defines the acoustic characteristics of an environment. In reality, since the absorption coefficient of the materials is not constant with frequency, we need to carry out the measurements at various frequencies to study the reverberation characteristics of an environment in depth. The decay of lower frequencies is slower, while that of higher frequencies is faster since obstacles absorb higher frequencies more easily than lower frequencies. So, how does reverb time affect listening? Its optimal value varies depending on the type of sound and the volume of the space. The bigger the space, the greater the optimal reverb time, for which, however, it is possible to give some indicative values: for speech, it is between 0.5 and 1 second; for chamber music, it is about 1.5 seconds; for symphonic music, it ranges between 2 and 4 seconds; for organ music, it is 5 or more seconds.»* (Bianchini and Cipriani, 2000).

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

the reverberation very clearly, removing the early reflections can be helpful to audibly separate the dry signal from the late reverberation. Using this technique, the presence of the original sound can be preserved without giving up the reverberation entirely.
In summary, reverberation consists of a pre-delay time between the direct sound and the first reflection, followed by early reflections and subsequently by late reverberation. There is an increase and then a decrease in reverb amplitude, at which point all reflections blend together and decay until they reach a level that is so low that the noise floor masks the reverberation – this happens over the time defined by the RT60. Eventually, the reverberation will fade out entirely. (See fig. 10.2)

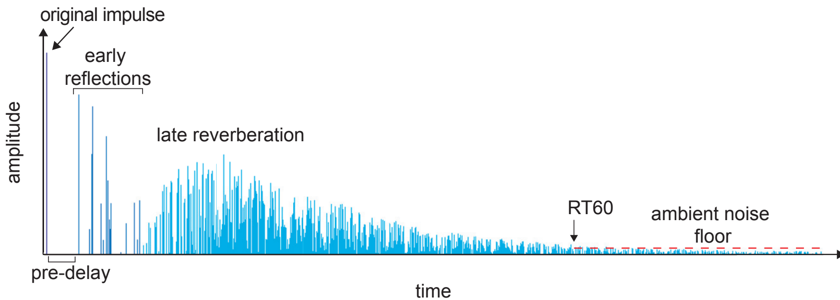


Fig. 10.2 The various stages of a reverb effect

SOUND EXAMPLE 10.2 • several reverbs that vary depending on the distance of the listener from the sound source

a) The direct sound (dry)
b) Listening in the same room – with a large distance between the listener and the sound source
c) Sound source outside the listener's room
d) Sound source upstairs from the listener

Reverb density depends on the quantity of audibly reflected waves and how close they are to each other over time. Some reflections will have a greater amplitude than others. If reflections with greater energy are repeated regularly over time, they will produce peaks at certain frequencies, resulting in a more "metallic" sound. When implementing reverb algorithms, this can sometimes be the desired effect. More often than not, however, it is preferable to have irregular time distances between the strongest reflections in order to avoid adding spurious frequencies to the original sound and, consequently, to produce a more natural effect.[4]

[4] Later, in section 10.5, we will see some other solutions to the problem of creating natural-sounding reverberation algorithms which are based on using either a larger number of elements (filters, delays), or a change over time in the parameter values of those elements.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Taking this into consideration, we can infer that both real acoustic spaces and invented virtual spaces can be simulated by constructing reverb algorithms. Moreover, we can set the positions of the various source sounds to different locations, some of which will be more present (in the foreground), others further away (in the background). We could also create figure-ground effects – to adapt a term from visual arts – for the sound sources or simulate a source getting closer to or moving away from the listener. The possibilities are virtually endless. We can also combine all of this with the position of the sounds in a real listening environment (on the *x*, *y*, and *z* axes of a multichannel context). We will talk about this in the sections about spatialization. For the moment, however, let's now take a look at one of the best-known techniques for simulating reverberations: Manfred Schroeder's recirculation technique.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

10.2 SCHROEDER REVERBERATOR

In 1961, Manfred R. Schroeder was the first to implement an algorithm for simulating a reverb effect in the digital domain.[5] Schroeder's recirculation technique uses two kinds of filters as basic building-blocks to implement a reverb algorithm: the recursive IIR comb filter and the allpass filter discussed in sections 6.6 and 6.7 of the second volume.[6] Several of these basic units are interconnected together to form an algorithm. In Schroeder's basic setup, the signal is first sent to four comb filters in parallel; each of these has different delay times (or, more precisely, loop times) so as not to create multiple simultaneous echo effects, thereby reducing repetition.[7] The use of different delay times is important if a "natural" reverb effect is to be achieved. Each comb filter generates a series of echoes, decreasing in amplitude. The outputs of the four comb filters are mixed and sent to two allpass filters in series, which are used to intensify the density of the "reflections" (see fig. 10.3). A multitap delay is then added to this configuration to simulate early reflections (see section 6.2). Note that the diagram shown in figure 10.3 illustrates a monophonic reverberator. In a stereo or multichannel context, the values of the various filters must be decorrelated in each channel in order to provide subtle variation in the output signals.[8]

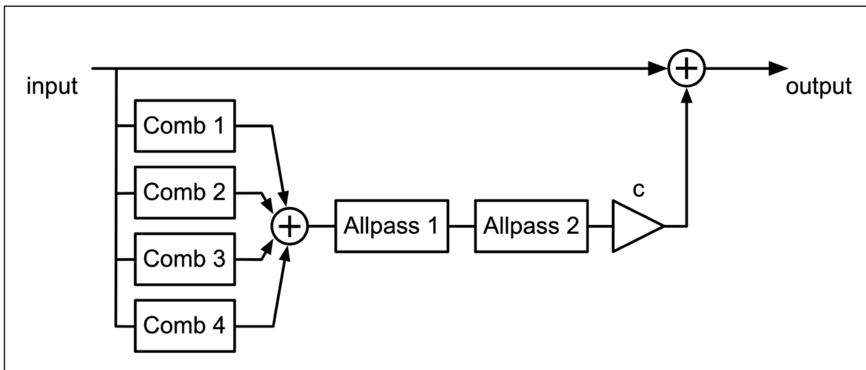


Fig. 10.3 The flow diagram of a Schroeder reverberator

Let us now take a closer look at the particular characteristics of the comb and allpass filters used in Schroeder's algorithm.

5 (Schroeder, 1961).

6 Also known as "unitary" reverberators (i.e., elementary reverberation units used to create more complex systems).

7 In reverb simulation, one way to avoid coincident echoes is to assign prime number multiples of a given value to the various delay times – although this alone does not ensure a "natural" effect. James Moorer, whose improvements to Schroeder's algorithm will be discussed later, also takes into consideration the notion that any sum or difference between two prime numbers that could be divided by 2 might also emphasize some frequencies. (Moorer, 1979).

8 In other words, for each channel, different sets of values should be used for the delay times of each filter.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

USING COMB FILTERS IN A SCHROEDER REVERBERATOR

Remember that, in a comb filter, the input signal is delayed by a certain amount of time which we will call *d*. Then, after being multiplied by a scaling factor which we will call *c*, it is mixed with the input signal going to the delay. This creates a feedback loop which, in this scenario, helps us simulate the recursive nature of some of the sound reflections which follow one another in an acoustic space. In the specific case of a Schroeder reverberator, the comb filter used is a delay with feedback that does not output the direct sound, and does not use an independent delay line for feedforward. We could adopt a simple configuration like the one shown in figure 10.4 for our reverb algorithm, or, alternately, we could use a more complex and flexible one that, such as the one which will be discussed shortly.

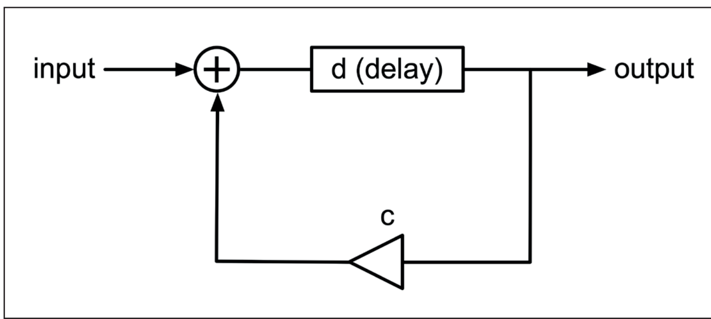


Fig. 10.4 The comb filter used in a Schroeder reverberator

The output signal of the simple comb filter shown in figure 10.4 will be a sound delayed by a time *d* followed by a decreasing series of echoes. The greater both the value *c* (which defines the amplitude of the signal that sent back to the input of the delay line) and the delay duration *d* (which defines the time between the various echoes), the longer will be the decay time of the echoes. As we already know, the factor *c* must always be less than 1, otherwise, the amplitude of the output sound would increase and exceed the maximum amplitude the system allows (0 dBFS in a digital system), instead of decreasing. The output of our comb filter with feedback ideally will have an impulse response similar to the one shown in fig. 10.5.

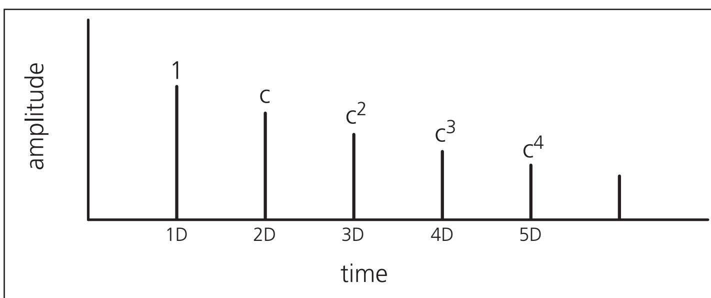


Fig. 10.5 The impulse response of a comb filter

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Although the comb filter configuration shown in figure 10.4 is computationally inexpensive and suitable enough for our purposes, it has a slight limitation in that we cannot adjust the amplitude and delay of both the feedforward and the feedback separately. To overcome this limitation, we could use the comb filter algorithm which was discussed in section 6.2 of the second volume (see fig. 10.6).

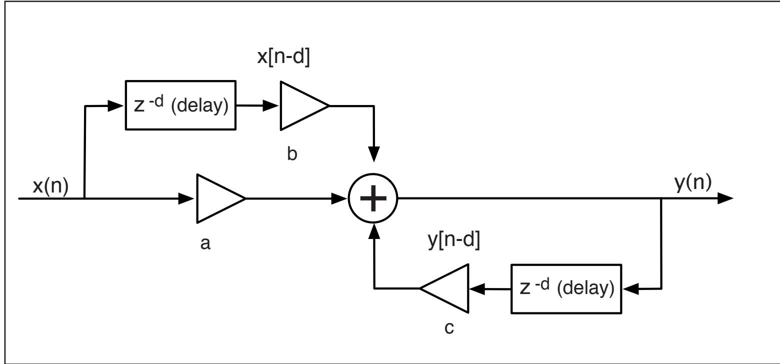


Fig. 10.6 An IIR comb filter with feedforward and feedback

To use this comb filter algorithm with the Schroeder reverberator shown in figure 10.3 and get identical results, we will need to adapt it slightly. To do that, we will need to assign a value of 0 to *a* (to remove the direct sound), a value of 1 to *b* (so that the feedforward output has the same amplitude as the input), and set *c* to the desired feedback value. This algorithm has two delay lines instead of one, so it will be slightly more computationally expensive, however it will also give us more flexibility when used in other situations.

USING ALLPASS FILTERS IN A SCHROEDER REVERBERATOR

As we already learned in section 6.7, allpass filters allow us to delay a signal, leaving the amplitude response of all the spectral components of the sound unaltered while altering their phase response (i.e., altering the way the phase varies across the frequency spectrum). Bear in mind that while allpass filters can have a "transparent" effect on sustained sounds, the effect of the phase shift becomes especially audible with a fast attack or decay.[9] As we already mentioned, in a Schroeder reverberator, the two allpass filters are connected in series. This means that each echo generated by the comb filters outputs a series of tightly-spaced echoes after passing through the first allpass filter, and each of these echoes, in turn, outputs other echoes after passing through

[9] *"We must remember, however, that the all-pass nature is more of a theoretical nature than a perceptual one. We should not assume, simply because the frequency response is absolutely uniform, that the filter is perceptually transparent. In fact, the phase response of the allpass filter can be quite complex. The all-pass nature only implies that in the long run, with steady-state sounds, the spectral balance will not be changed. This implies nothing of the sort in the short-term, transient regions." (Moorer, 1979. p.14).*

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

the second allpass filter. This is how we can generate reverberation with high repetition density. For more details about the design of Schroeder's allpass filter, please refer to section 6.7 of the second volume, where it is discussed comprehensively.

LATER IMPROVEMENTS UPON SCHROEDER REVERBERATOR
In 1970, a few years after the first reverb algorithms were developed, Schroeder added a multitap delay line to the original design to simulate early reflections (see fig. 10.7).

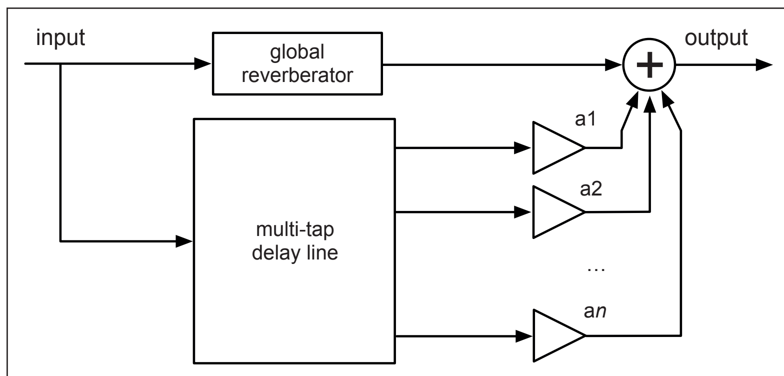


Fig. 10.7 A Schroeder reverberator with a multitap delay line

Further developments were then introduced by James A. Moorer. We will see more complex and effective algorithms in later sections of this chapter. However, Schroeder was the first to lay the foundations for digital reverb algorithms.

(...)

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

other sections in this chapter:

10.2 THE SCHROEDER REVERBERATOR

10.3 FREEVERB

10.4 THE DATTORRO REVERBERATOR (PLATE REVERB SIMULATION)

10.5 FDN REVERBERATOR

10.6 CREATIVE USES OF REVERBERATION

10.7 TWO-CHANNEL SOUND SPATIALIZATION
Interaural Time Difference
Interaural Intensity Difference
Head Related Transfer Function (HRTF)
Sound in motion: The Doppler effect
The Mid/Side technique

10.8 MULTICHANNEL SOUND SPATIALIZATION
Panning a sound in a multichannel system
Panning a sound "within the room"
5.1 Surround System
The double M/S technique

ACTIVITIES
- **Sound examples**

TESTING
- **Questions with short answers**

SUPPORTING MATERIALS
- **Fundamental concepts - Glossary**

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

10P
REVERBERATION AND SPATIALIZATION

10.1 INTRODUCTION TO REVERBERATION ALGORITHMS
10.2 THE SCHROEDER REVERBERATOR
10.3 FREEVERB
10.4 THE DATTORRO REVERBERATOR (PLATE REVERB SIMULATION)
10.5 FDN (FEEDBACK DELAY NETWORK) REVERBERATION
10.6 CREATIVE USES OF REVERBERATION
10.7 TWO-CHANNEL SOUND SPATIALIZATION
10.8 MULTICHANNEL SOUND SPATIALIZATION

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

LEARNING AGENDA

PREREQUISITES FOR THIS CHAPTER
- The contents of volumes I and II, Chapter 10T and Interlude F

OBJECTIVES
Skills
- To be able to program and use reverb algorithms for Schroeder reverberator, Freeverb, Dattorro reverberator and FDN reverberator
- To be able to use reverberation in creative, nonstandard ways
- To be able to program and use spatialization algorithms for two or more channels
- To be able to program and use spatialization algorithms in 5.1 surround

Competence
- To be able to create a brief etude based on creative uses of reverberation and spatialization

ACTIVITIES
- Constructing and modifying algorithms

SUPPORTING MATERIALS
- List of Max objects – List of messages for specific Max objects

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

10.1 INTRODUCTION TO REVERBERATION ALGORITHMS

In this chapter, we will explain some algorithmic reverberators created using various combinations of delay lines. The fundamental elements of these algorithms are:
- Comb filters and Schroeder allpass filters (we discussed these in sections 6.6 and 6.7 of the theory and practice chapters of the second volume). We will also use a comb filter with a lowpass filter in the feedback chain, like that discussed in section IF.2 (Interlude F of this volume).
- Multitap delay lines (sections 6.2T and 6.2P of the second volume), which will be used for Dattorro reverb.
- Low-Frequency Oscillators (LFO, Chapter 4 of the first volume), which will be used to modify the delay lines over time in order to avoid repetitive patterns in the reverberation.

These processes are already well-known to us. What is new at this point is the number of delay lines that come into play and the different ways the elements are combined.

10.2 THE SCHROEDER REVERBERATOR

Open the patch **10_01_Schroeder_Rev.maxpat** (fig. 10.1) and listen to all the presets while observing the parameter values.

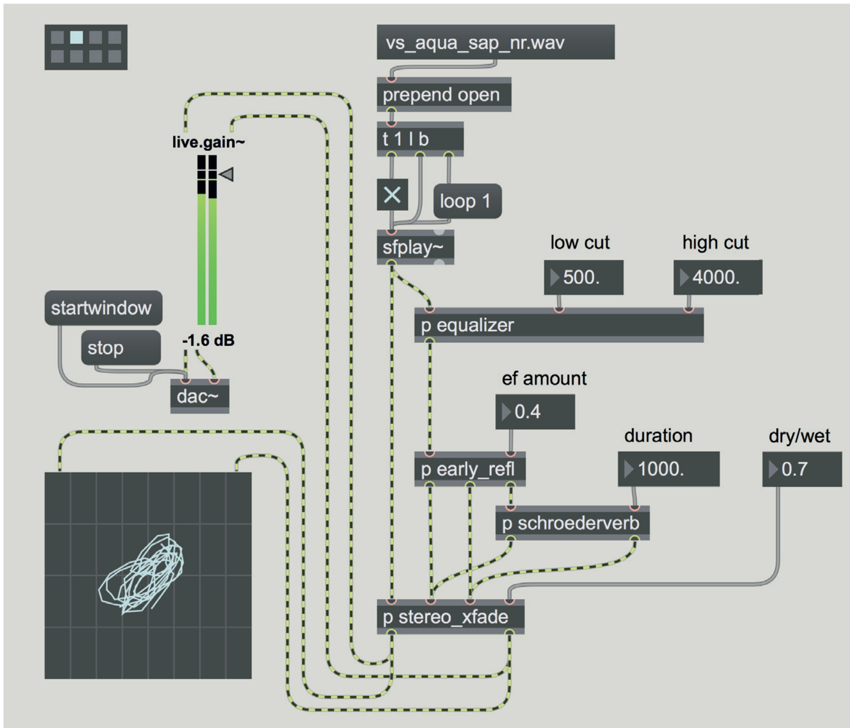


Fig. 10.1 The patch **10_01_schroeder_rev.maxpat**

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

The effect produced by this algorithm is quite elementary by today's standards; you can hear repetitive echoes and metallic resonances in several presets. However, at the time when it was proposed by Manfred Schroeder (the early '60s of the last century), it constituted an efficient system for the digital simulation of reverberant environments, and today it is useful precisely in order to reproduce these "primitive" and metallic effects.

As we can see from the figure, the signal is routed through four subpatches. The first subpatch filters the signal by cutting some of the low and high frequencies. This filtering is used to define (very approximately) the frequency absorption characteristics of the environment we want to simulate; it is useful above all to attenuate the metallic resonances that may occur in the high frequency region and an excessive rumble that may arise in the lows. The algorithm contained in the subpatch is fairly simple, and we suggest that you analyze it by yourself.

The second subpatch adds the early reflections (see section 10.1T). This algorithm is also quite simple (a multitap delay). The delay values were taken from the article by J.A. Moorer, "About This Reverberation Business," published in the Computer Music Journal in 1979.[1]

The third subpatch contains the actual reverberator (see figure 10.2).

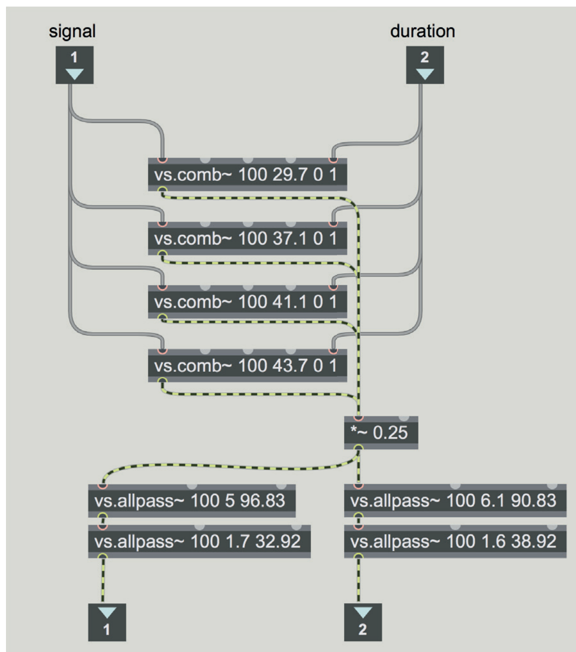


Fig. 10.2 A Schroeder reverberator

[1] Moorer, J.A., 1979, pp. 13-28

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

As we explained in section 10.2T, a Schroeder reverberator is made up of four parallel comb filters[2] connected to two cascaded allpass filters. In this implementation, we used two pairs of cascaded allpass filters and gave each pair slightly different values for delay and decay. This decorrelates the left and right channels and produces a stereophonic effect.[3] The duration value in milliseconds (the number box "duration" in the main patch) sets the decay time for the four comb filters.
Note that we used the **vs.comb~** and **vs.allpass~** objects from the Virtual Sound Macros library, which allow us to adjust the decay time in milliseconds. (You may remember that in the standard **comb~** and **allpass~** objects the decay time is controlled by the feedback value.)

ACTIVITIES

- Make new presets for the patch shown in figure 10.1.
- In the subpatch [p schroederverb], try modifying the values of delay and decay for the comb and allpass filters; try to create interesting models of reverberation.
- Try varying the quantity of comb and allpass filters.

(...)

[2] The configuration of these comb filters was explained in section 10.2T; see in particular the description of the parameters for the figure 10.6.
[3] The parameters used are taken, with some modifications, from the book *Computer Music*, by C. Dodge and T. A. Jerse (p. 301).

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

other sections in this chapter:

10.2 THE SCHROEDER REVERBERATOR

10.3 FREEVERB

10.4 THE DATTORRO REVERBERATOR (PLATE REVERB SIMULATION)

10.5 FDN REVERBERATOR

10.6 CREATIVE USES OF REVERBERATION

10.7 TWO-CHANNEL SOUND SPATIALIZATION
The Doppler effect
Mid/Side encoding

10.8 MULTICHANNEL SOUND SPATIALIZATION
Moving sounds in a multichannel system
Moving sounds in a 5.1 system
Double M/S technique

ACTIVITIES
- **Substituting parts of algorithms**
- **Correcting algorithms**
- **Completing algorithms**
- **Analyzing algorithms**

TESTING
- **Integrated cross-functional project: reverse engineering**

SUPPORTING MATERIALS
- **List of Max objects - List of messages for specific Max objects - List of GEN operators**

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

11T
NONLINEAR SYNTHESIS

11.1 AMPLITUDE MODULATION TECHNIQUES: AM, RM, AND SSB
11.2 FREQUENCY MODULATION AND PHASE MODULATION: FM, PM, AND FEEDBACK PM
11.3 PHASE DISTORTION
11.4 NONLINEAR DISTORTION (NLD) OR WAVESHAPING
11.5 WAVE TERRAIN SYNTHESIS (WTS)
11.6 SPLIT SYNTHESIS

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

LEARNING AGENDA

PREREQUISITES FOR THIS CHAPTER
- Contents of chapters 1-10

OBJECTIVES
Knowledge
- To learn the theory and use of amplitude and ring modulation synthesis techniques
- To learn the use of DC Offset in modulation
- To learn the theory and use of single-sideband modulation
- To learn the theory and use of frequency and phase modulation synthesis techniques
- To learn the theory and use of the PM feedback technique
- To learn the basic concepts on spectra families using FM
- To learn the theory and use of phase distortion and nonlinear distortion (waveshaping)
- To learn the theory and use of the waveterrain technique
- To learn the theory and use of distortion techniques
- To learn the theory and use of split synthesis

Skills
- To be able to listen and identify sounds generated using various nonlinear synthesis techniques

CONTENTS
- Nonlinear synthesis techniques for modulation
- Amplitude, ring, and single-sideband modulation
- Frequency modulation with multiple modulators and/or carrier signals
- FM spectra families
- Phase modulation and feedback PM
- Phase distortion
- Nonlinear distortion (NLD, or waveshaping) and distortion
- Wave Terrain Synthesis
- Split synthesis

ACTIVITIES
- Sound examples

TEST
- Short-answer questions

SUPPORTING MATERIALS
- Basic concepts - Glossary

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

NONLINEAR SYNTHESIS

The various types of nonlinear synthesis differ from linear techniques, such as additive synthesis or subtractive synthesis, in two principal ways: in the partials which are added to the resulting spectrum and/or how it is transposed. Linear techniques produce a signal whose components are not altered in frequency or different from those of the input signal. By contrast, such a change of frequency components can be achieved using nonlinear techniques. A typical example, which we will learn about in detail a little later, is frequency modulation, which allows us to create complex sounds using just two sine wave oscillators – this is something which is impossible to obtain using a linear technique like additive synthesis.

11.1 AMPLITUDE MODULATION TECHNIQUES: AM, RM, AND SSB

Modulation is the alteration of the instantaneous amplitude, instantaneous frequency, or phase of a signal using another signal. In its simplest form, modulation can occur between two oscillators; it can also be used in very complex algorithms, such as those which involve modulating sampled – or even live – sounds. As we learned in section 4.4T of the first volume, in a basic modulation algorithm having two oscillators, the oscillator being modulated is called the **carrier**, and the oscillator causing the modulation is called the **modulator**.
In section 4.5, we saw an example of a tremolo created using an amplitude modulation algorithm. In this example, the modulator signal was produced by an LFO (therefore having a frequency below the audible spectrum) and its amplitude, compared to that of the carrier, was limited. The effect produced by the modulating oscillator was a slight repetitive change in the amplitude of the carrier signal. This is an effect which is perceived in the time domain.
But what if the frequency of the modulating oscillator were higher and its amplitude greater? In this case the effect would be quite different from a perceptual point of view, since it would cause new frequencies (called **sidebands**) to be generated and added to the carrier's spectrum. These frequencies are produced symmetrically on either side of the carrier frequency (i.e., both above and below it), and are known as the Upper Sideband, or **USB**, and Lower Sideband, or **LSB**. These will be discussed further in the next section. The effect, in this case, is now perceived in the frequency domain.
If we start from a tremolo effect (using an LFO), and increase the frequency of the modulator signal, at what point do we stop hearing the tremolo and start hearing distinct sidebands, instead? Generally speaking, if the modulator frequency is below 10 Hz, our auditory system can distinguish the individual amplitude variations (e.g., a tremolo) but cannot separate sidebands from the center frequency. This happens because these sidebands fall within the critical band.[1] In this case, the sidebands do indeed exist, but are too close to the center frequency to be perceived as separate frequencies.

[1] For additional details on critical bands, please refer to section 2.2T.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

When we increase the modulator frequency above 10 Hz – bringing this frequency over half of the critical band[2] – we will start hearing sidebands and lose the perception of the individual repeated variations in amplitude. In-between these two perceptually different effects, a gray area is created, similar to the effect which we perceived with beating as the distance between two frequencies is increased.

SOUND EXAMPLE 11.1
From dry sound to tremolo effect to distinct sidebands

We will now take a look at three modulation techniques in which the instantaneous amplitude is modulated.

– **RM = Ring Modulation**
– **AM = Amplitude Modulation**
– **SSB = Single Sideband modulation**

AM and RM are closely related effects. Their difference lies in the modulator signal: amplitude modulation uses a unipolar modulator, ring modulation uses a bipolar modulator. As we learned in section 1.1, **bipolar** signals oscillate between positive and negative amplitude values, while **unipolar** signals oscillate only within the positive (or negative) value range.
In chapter 4, we used both bipolar and unipolar modulator signals, although in those cases the signals never fell in the audible spectrum. In this chapter, modulator signals will no longer be limited to the sub-audio range.
With the SSB technique, only one of the two ring modulation sidebands is generated – one of the main reasons why it is most often employed to implement real-time frequency shifters.[3]

RING MODULATION (RM)

The name ring modulation comes from the original analog implementation, in which a “ring” of diodes was used to multiply the input signal with a bipolar square wave.[4]

[2] Two sounds can be perceived as distinct only when they excite sufficiently distant nerve endings, and therefore when they fall into two distinct critical bands. Below 200 Hz, the width of the critical bands is more or less constant. From about 200 Hz upwards, the width of the critical band increases as the frequency increases. Please refer to section 2.2T for the calculation of critical band values.
[3] For details, you may refer to Bode and Moog, 1972.
[4] RM, AM, SSB, and FM techniques were mainly used in radio broadcast. The idea of using ring modulation to modify the sounds was born in the WDR studio in Cologne. Karlheinz Stockhausen used it in *Momente, Mixtur, Mantra*, and *Mikrophonie I*.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

The most common algorithm used to create **ring modulation** is to simply multiply two bipolar audio signals in the time domain.[5] For the sake of simplicity and shared terminology with other types of modulation, we shall call these two signals the carrier and the modulator – even though, in reality, the two signals play the same role with respect to one another, since multiplication is a commutative operation regardless of the order in which the two signals are multiplied. Fig. 11.1 shows an example of multiplying two sine waves: a carrier, labeled *c*, and a modulator, labeled *m*; f_C and f_m are their respective frequencies. As we will see later, more complex signals could be used, too.

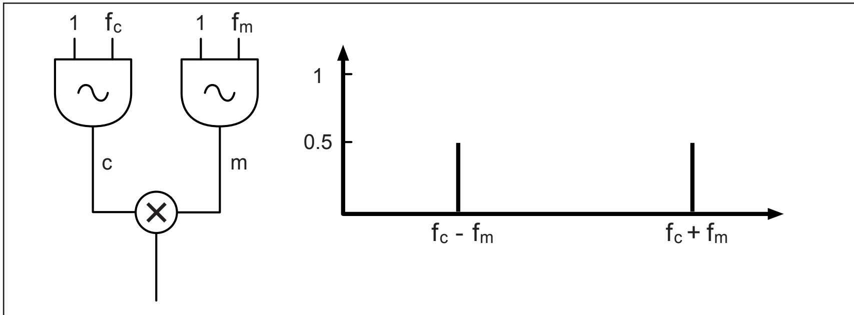


Fig. 11.1 Ring modulation obtained by multiplying two bipolar sine waves

When performing ring modulation with two sine waves, if f_C is equal to 700 Hz and f_m is equal to 200 Hz, the modulated output will produce two sidebands consisting of sine waves whose frequencies are the sum and difference of the input frequencies:[6]

$f_C - f_m$	$f_C + f_m$
700 - 200 = 500 Hz	**700 + 200 = 900 Hz**

To explain this phenomenon, we can use the second Werner formula[7]

$$\cos\alpha\cos\beta = \frac{1}{2}\left[\cos(\alpha+\beta)+\cos(\alpha-\beta)\right]$$

By multiplying two cosines with angle α and β respectively, the result will be equal to the cosine of the sum of the angles (α + β, which, in our case, corresponds to the upper band) plus the cosine of the difference of the angles (α - β, which corresponds to the lower band). The resulting signal is then divided by 2 (i.e., its amplitude is scaled).

[5] Later in Chapter 13T, we will see why the multiplication of two signals in the frequency domain (i.e., the spectral representation of the signals) is equivalent to a convolution.
[6] The name used in broadcasting is double-sideband suppressed-carrier modulation.
[7] Johann Werner (1468-1522) was a German mathematician and cartographer. His four formulas (conceived in the field of trigonometry), which relate sines and cosines in various identities, are also used to describe how sidebands are generated in AM for radio engineering.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

As previously mentioned, we can modulate either a sampled sound or a sound used as the carrier in real-time. We can also multiply two complex synthesis signals, or even sampled signals, without the need for oscillators. In all these cases, the output sound will contain the sum and difference frequencies of all the spectral components of the two sounds.

For example, suppose that a given carrier ***c*** has 3 frequency components, and a given modulator ***m*** has 2, as shown below.

c	***m***
800, 1600, 2400 Hz	**250, 500 Hz**

If we ring modulate these two signals, 12 components will be generated on output. To know the number of output components, you need to multiply the number of components in *c* by the number of components in *m* and double the result (in the example above, that would be 3*2*2). Sometimes, however, if there are harmonically related components, identical frequencies may be generated. In this case, their amplitude values would be added together, and the number of components would be lower.

In the example above, the following frequencies would be output (see the spectrum shown in fig. 11.2):

***c+m* = 1050, 1300, 1850, 2100, 2650, 2900 Hz**
***c-m* = 300, 550, 1100, 1350, 1900, 2150 Hz**

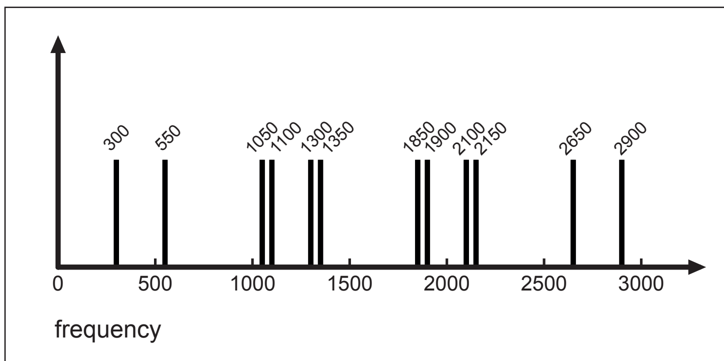


Fig. 11.2 the output spectrum generated by the modulation algorithm with complex signals *m* and *c*

As for the amplitude values of sounds resulting from ring modulation, if we use two sine waves, the amplitudes of both output components will be equal to the product of the amplitudes of the inputs divided by two – in other words: *c*m/2*. Analogously, if we ring modulate complex sounds, the amplitude of each output component will be equal to that of the respective components of *c* and *m* that generated it, multiplied together, and then divided by 2. If the amplitude of *m* (or *c*) equals 0, no signal will be output.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Frequencies with negative values can also be created in cases where the frequency of the modulator is greater than that of the carrier. For example, if f_c is equal to 700 Hz, and we set the value of f_m to 1000, we will obtain the following output values:

$f_c - f_m$	$f_c + f_m$
-300	**1700**

As explained in section 5.1 of the second volume, frequencies below zero reappear, mirrored into the positive domain, with an inverted sign (-200 Hz becomes 200 Hz, -300 Hz becomes 300 Hz, etc.). At the upper end of the spectrum, such mirroring also happens to any side frequencies exceeding the Nyquist limit.

AMPLITUDE MODULATION (AM)

Let's jump right into learning about AM by starting out with a basic example and analyzing the behavior of the modulated signal from the frequency point of view. For this example, a unipolar sine wave in the audio band (e.g., 300 Hz) will be used to modulate the amplitude of a sinusoidal carrier oscillator (e.g., 700 Hz). The resulting output signal, unlike that produced by ring modulation, contains the frequency of the carrier (700 Hz) in addition to the two sidebands. Similarly to ring modulation, the value of the first sideband will be equal to the sum of the carrier frequency plus the modulator frequency (in this case, 700+300 = 1000 Hz), and the value of the second sideband will be equal to the difference between the carrier frequency and the modulator frequency (in this case, 700-300 = 400 Hz).
As before, we will call the frequency of the carrier f_c and the frequency of the modulator f_m. The bandwidth of the output signal (i.e., the difference between the lowest and the highest components) will be equal to 2* f_m.
The output signal will consist of the following three frequencies: the carrier frequency and the two side frequencies generated by the modulation.

$f_c - f_m$	f_c	$f_c + f_m$
400	**700**	**1000**

If the carrier and the modulator frequencies are harmonically related, the result will be a harmonic sound. Otherwise, an inharmonic sound will be generated. If f_m = 200 Hz and f_c = 400 Hz, the following frequency values will be output (resulting in a harmonic sound with a fundamental frequency of 200 Hz, with the second and third harmonics):

$f_c - f_m$	f_c	$f_c + f_m$
200	**400**	**600**

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Let us now take a look at the amplitude values of these three components.
First of all, note that in order to create a unipolar signal (i.e., the signal which is used as the modulator), we need to take a bipolar signal and add a DC Offset[8] to it. In the classic AM setup, the value of this DC Offset is equal to the amplitude of the sinusoidal modulator. For example, if the amplitude of the modulator is equal to 1 (in other words, it oscillates between 1 and -1) and we add a DC Offset with the same value (1), we will get a unipolar signal that oscillates between 0 and 2.
After the modulation, the output amplitude of the center component will be equal to the amplitude of the DC Offset (1). We also get two side frequencies whose amplitude values are equal to half the amplitude of the center frequency (.5). Because the sum of the amplitude values of the sidebands and the carrier might exceed the maximum amplitude value, we need to rescale the amplitude of the output signal. Figure 11.3 summarizes this example, graphically.

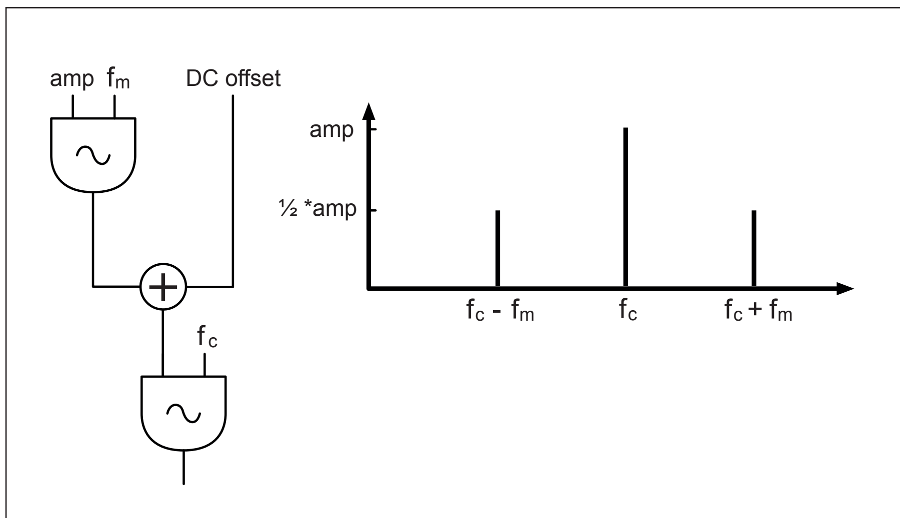


Fig. 11.3 The flow chart of a classic AM setup and its resulting spectrum

As we said earlier, the result is similar to that of RM for the sidebands. However, the output of AM will also contain a signal with the same frequency as the carrier since there is a DC Offset in the modulator which determines the output amplitude of the carrier component. As you already know, the DC offset can be considered as a 0 Hz component. So, actually, two signals are modulating the carrier: the modulator and the DC Offset. We can therefore state that the "modulation" between the DC Offset and the carrier generates the frequency of the carrier itself on output.

$$f_C \pm 0 = f_C$$

[8] As we already know, the DC offset is a component with a fixed frequency of 0 Hz. It therefore does not generate any sidebands.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

This explains why AM outputs the carrier while RM does not.
That having been said, we can actually make the amplitude of the DC Offset and that of the modulator independent from one another. To do this, we simply need to build an algorithm that allows us to vary the output amplitude of the sidebands and that of the carrier independently, even using envelopes, in order to obtain a continuous variation between RM and AM in the output spectrum. A graphical representation of this is shown in figure 11.4.[9]

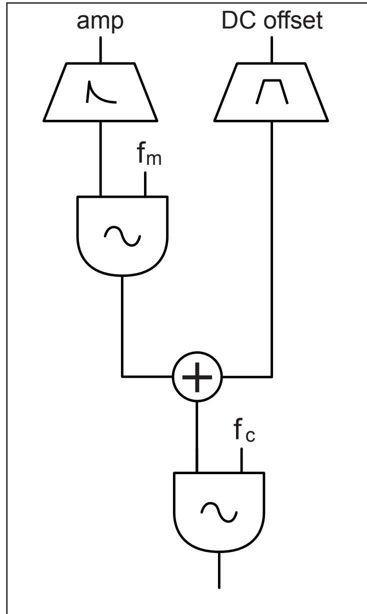


Fig. 11.4 An AM algorithm whose DC Offset is independent of the amplitude of the modulator

SOUND EXAMPLE 11.2
A continuous transition from AM to RM and vice versa

(...)

[9] In many cases, we will find a parameter called the modulation index. This is basically a multiplier that allows us to vary the ratio between the modulator amplitude and the DC Offset. If the modulation index is equal to 1, the modulator amplitude will be equal to the DC Offset value. Changing the modulation index, while keeping the same DC offset, will cause a change to the ratio between the amplitude of the sidebands and that of the carrier.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

other sections in this chapter:

Single-sideband modulation (SSB) and frequency shifting

11.1 AMPLITUDE MODULATION TECHNIQUES: AM, RM, AND SSB

11.2 FREQUENCY MODULATION AND PHASE MODULATION: FM, PM, AND FEEDBACK PM
FM: Basic theory
Spectra families
The C:M ratio families in normal form
FM: complex modulation
The use of mathematical rules In John Chowning's *Stria*
Phase modulation (PM)
Feedback PM

11.3 PHASE DISTORTION

11.4 NONLINEAR DISTORTION (NLD) OR WAVESHAPING
Sine wave nonlinear distortion
Use of Chebyshev Polynomials for NLD
Distortion of complex signals
Other types of nonlinear distortion

11.5 WAVE TERRAIN SYNTHESIS (WTS)

11.6 SPLIT SYNTHESIS

ACTIVITIES
- **Sound examples**

TESTING
- **Questions with short answers**

SUPPORTING MATERIALS
- **Fundamental concepts - Glossary**

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

11P
NONLINEAR SYNTHESIS

11.1 AMPLITUDE MODULATION TECHNIQUES: AM, RM AND SSB
11.2 FREQUENCY MODULATION AND PHASE MODULATION: FM, PM, AND FEEDBACK PM
11.3 PHASE DISTORTION
11.4 NONLINEAR DISTORTION (NLD) OR WAVESHAPING
11.5 WAVE TERRAIN SYNTHESIS (WTS)
11.6 SPLIT SYNTHESIS

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

LEARNING AGENDA

PREREQUISITES FOR THIS CHAPTER
- The contents of volumes I and II, Interlude F, Chapter 10 and Chapter 11T

OBJECTIVES
Skills
- To be able to program and use algorithms for AM, RM, PM and FM
- To be able to use algorithms for phase distortion, waveshaping, waveterrain synthesis, split synthesis and distortions

Competence
- To be able to create a brief study based on creative uses of nonlinear synthesis techniques

ACTIVITIES
- Constructing and modifying algorithms

SUPPORTING MATERIALS
- List of Max objects - List of attributes and messages for specific Max objects - List of Gen operators and attributes

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

11.1 AMPLITUDE MODULATION TECHNIQUES: AM, RM AND SSB

RING MODULATION

Performing a ring modulation (RM) between two sine waves is very easy: all we need to do is to multiply the outputs of two oscillators (figure 11.1).

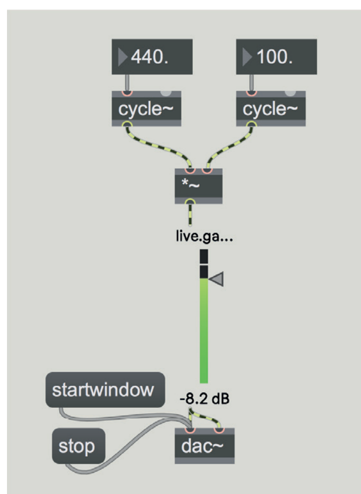


Fig. 11.1 Ring modulation between two sine wave oscillators

If you make the patch shown in the figure and set the frequency values of the two oscillators to 440 Hz and 100 Hz, you will obtain at the output two sine waves with frequencies of 340 Hz (440 - 100) and 540 Hz (440 + 100). Try gradually increasing or decreasing the frequency of the second oscillator: you will hear the two resulting sounds moving farther apart or closer together.
The phenomenon of ring modulation might seem to apply only within the range of audio frequencies (20-20000 Hz). Try setting the frequency of the second oscillator to 1 Hz, thereby changing it into an LFO: the sound produced is a 440-Hz sine wave whose amplitude follows the progress of the LFO. In this case, apparently, we no longer have the sum and difference of two frequencies at the output, but rather a single frequency whose amplitude varies over time. But in reality, this distinction is due only to the limits of our perception: naturally, the trigonometric formulas (Werner formulas) that govern the RM are valid regardless of the frequencies used. If we apply these formulas to the case in question, the resulting frequencies will be 439 Hz (that is, 440 - 1) and 441 Hz (440 + 1). Being very close together, these two frequencies give rise to the phenomenon of beats, and the perceptual result, in this case, is a 440-Hz sound that oscillates twice per second. In fact, the difference between 439 and 441 is 2, while the average of the two frequencies is 440 (i.e., (439+441)/2; see section 2.2 of the first volume).

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

An RM between two sine waves at 440 Hz and 1 Hz, therefore, produces the same effect as that produced by the sum of two sine waves at 439 Hz and 441 Hz. Make the patch shown in figure 11.2 and verify that the sound produced by the RM is identical to that produced by the sum of the two sine waves.

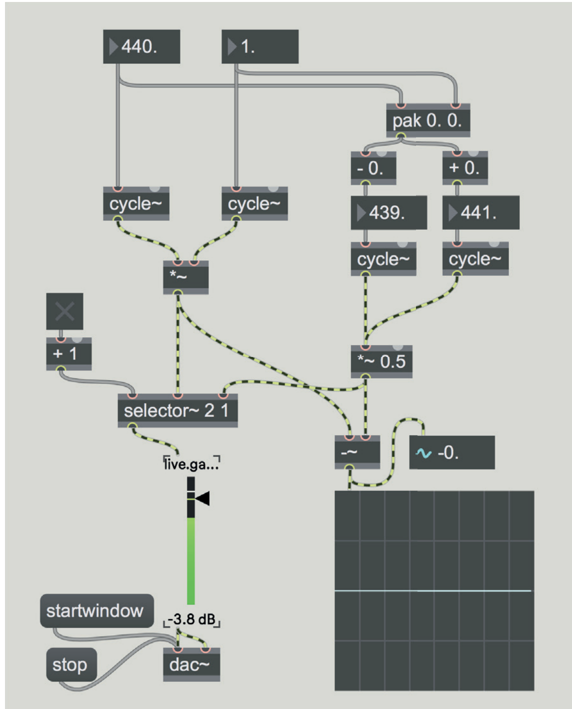


Fig. 11.2 RM and beats

Notice that we halved the sum of the two sine waves so that the resulting signal would vary between -1 and 1, as does the multiplication, rather than between -2 and 2. Switching back and forth between the first and second signals (with the `selector~`), you should not hear any clicks produced by discontinuities, because the two signals are identical. Moreover, as shown in the figure, we can verify that two signals are equal by subtracting them: if the result is a constant signal of 0, the two signals are the same.

To obtain a more complex sound, we can multiply multiple sine waves in series. The frequency of each new sine wave is added to and subtracted from all the pre-existing components, doubling their number. The number of resulting components for *n* oscillators multiplied together is thus equal to 2^{n-1}. With 4 oscillators, for example, we have 8 components (i.e., 2^3). Make the patch shown in figure 11.3.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

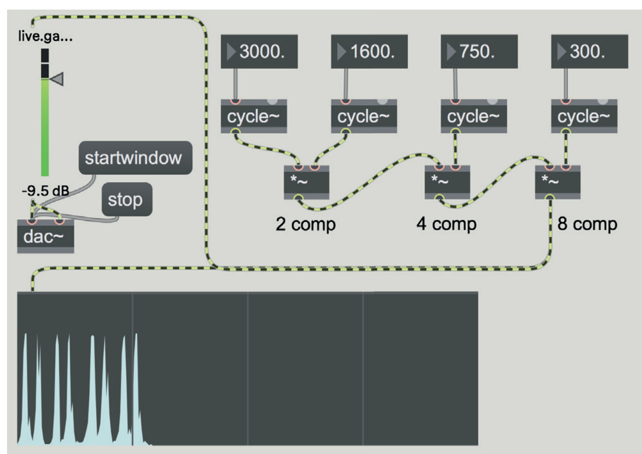


Fig. 11.3 Multiple RM

As we saw in the theory chapter, it is possible to perform an RM between a concrete sound (the sound of an instrument, for example) and a sine wave oscillator – and this will produce far more interesting results that what we can obtain with simple sine waves.
Open the patch **11_01_instr_RM.maxpat** (figure 11.4).

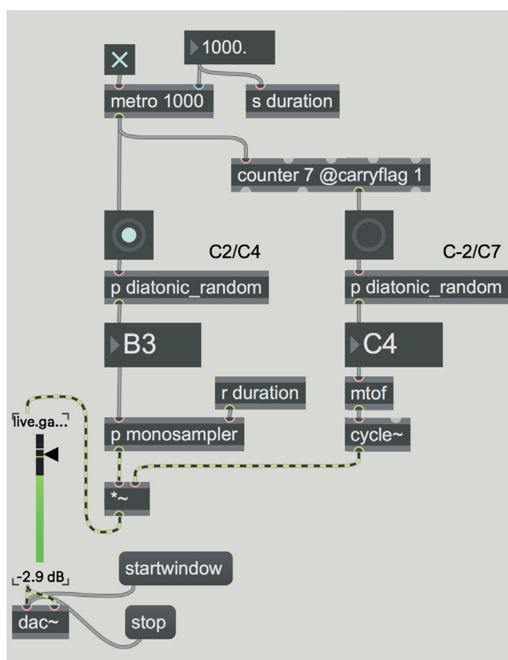


Fig. 11.4 The patch **11_01_instr_RM.maxpat**

In this patch, each bang of the `metro` object generates random diatonic notes (corresponding to the white keys of a piano). The subpatch [`p` monosampler]

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

on the left contains a small monophonic sampler that plays the sound of a vibraphone. The other random diatonic-note generator (which outputs a new value every 8 bangs) is connected to a sine wave oscillator that performs a ring modulation with the sampled sound. Listen carefully to the timbral variation produced by this patch: notice that each time the frequency of the sine wave oscillator changes, there is a new "family" of similar timbres.
Now let's see what happens when we perform an RM between a variable-frequency noise generator and a sine wave: open the patch **11_02_noise_RM.maxpat** (figure 11.5).

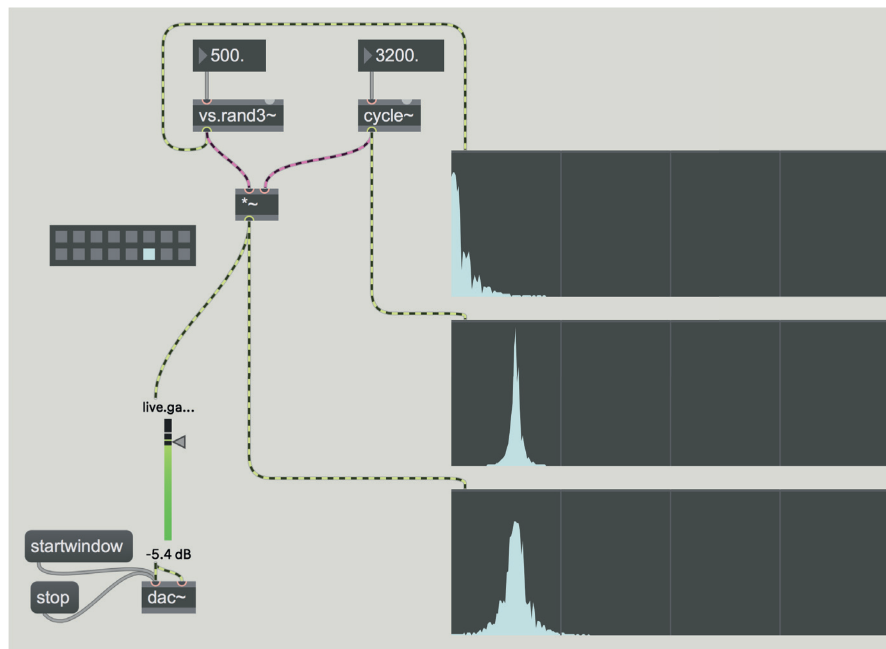


Fig. 11.5 The patch **11_02_noise_RM.maxpat**

As we learned in section 3.1 of the first volume, a variable-frequency noise generator produces frequency bands whose width is proportional to the selected frequency. If we perform an RM between this signal and a sine wave, these frequency bands are shifted both above the frequency of the sine wave and – specularly (i.e., inverted) – below it.
The three spectroscopes in figure 11.5 show, from top to bottom, the spectrum of the noise generator, the spectrum of the sine wave oscillator (a single peak), and the positive and negative spectrum of the noise generator shifted to the frequency of the sine wave.
Try gradually decreasing the frequency of the noise generator: the spectral bands narrow around the frequency of the sine wave oscillator, which becomes more and more clearly audible; the effect is similar to that obtained by increasing the Q of a bandpass filter which is filtering white noise.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

ACTIVITIES

– In the patch shown in figure 11.5, replace the sine wave oscillator with a sampled sound and observe how the noise generator alters the spectrum of the sound source.
– Perform an RM between a sampled sound and a sine wave at the Nyquist frequency (for example, [cycle~ 22050] if the sample rate is 44100 Hz). Observe with the spectroscope that this produces an inverted spectrum (in essence, the highest frequencies take the place of the lowest frequencies and vice versa): can you explain why? (Reread the subsections on foldover and aliasing in section 5.1 of the second volume.)

AMPLITUDE MODULATION

Now let's look at an AM patch that allows us to mix the carrier and the modulator independently: open the file **11_03_instr_AM.maxpat** (figure 11.6).

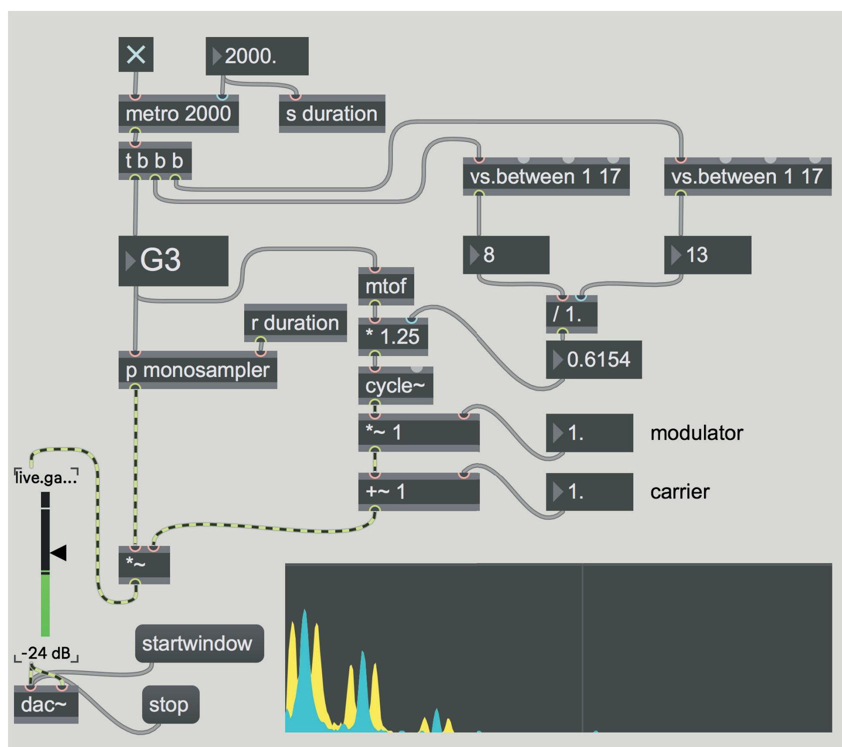


Fig. 11.6 The file **11_03_instr_AM.maxpat**

In this patch, the carrier is the vibraphone sound that we used earlier, while the modulator is a sine wave oscillator whose frequency is calculated in relation to the carrier frequency. This relation is expressed as a faction whose denominator and nominator are both random values between 1 and 16.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

The output of the modulator is first multiplied by the value indicated in the number box "modulator"; then a constant value is added to this signal (DC offset) via the number box "carrier." By modifying the value of the number box "modulator," we can vary the amplitude of the sidebands, while the number box "carrier" allows us to vary the amplitude of the carrier.
The spectroscope on the lower right displays the spectrum of the resulting sound: the components of the carrier are shown in light blue, while the sidebands generated by the modulation are shown in yellow.
We leave the analysis of this patch up to the reader. Notice that if we set the value "carrier" to 0 while leaving "modulator" at 1, we obtain an RM; and that if instead we set the value "modulator" to 0 while leaving "carrier" at 1, we simply obtain the signal of the carrier. Other combinations of these values allow us to obtain timbres with varying degrees of inharmonicity.
If we use envelopes to control the parameters "modulator" and "carrier," we can vary the timbre dynamically. Let's look at an example: open the patch **11_04_instr_AM_env.maxpat** (figure 11.7).

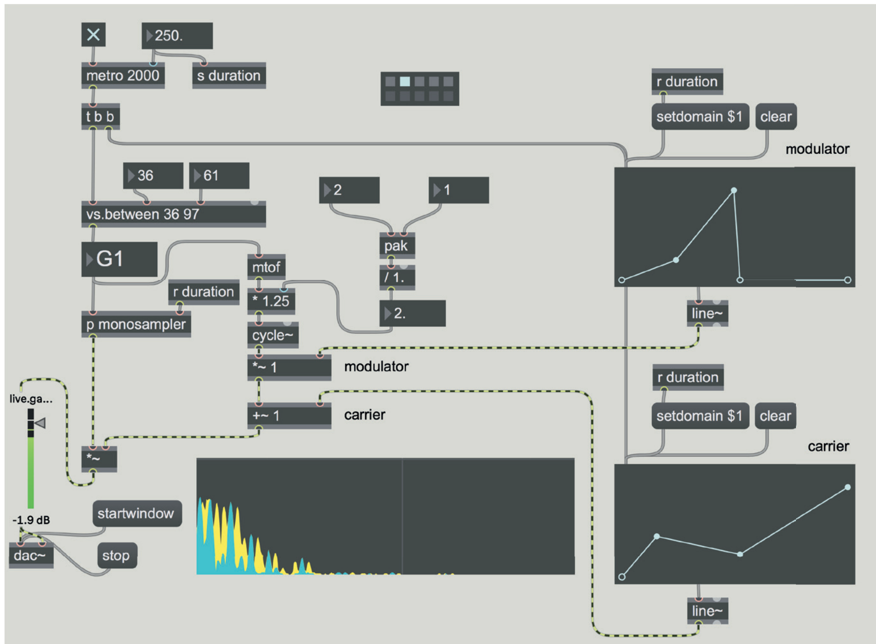


Fig. 11.7 The patch **11_04_instr_AM_env.maxpat**

This patch is a variation of the previous one and is fairly easy to analyze. Listen to the various presets and notice that the frequency ratio between the modulator and the carrier is not generated randomly but is fixed for each preset, while the notes played by the carrier are generated randomly. Make more presets.

(...)

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

other sections in this chapter:

Single-sideband modulation (frequency shifting)

11.1 AMPLITUDE MODULATION TECHNIQUES: AM, RM, AND SSB

11.2 FREQUENCY MODULATION AND PHASE MODULATION: FM, PM, AND FEEDBACK PM
Spectral families
Feedback modulation
Complex modulations

11.3 PHASE DISTORTION

11.4 NONLINEAR DISTORTION (NLD) OR WAVESHAPING
Chebyshev polynomials
NLD without lookup table

11.5 WAVE TERRAIN SYNTHESIS (WTS)
Lissajous curves
Rhodonea curves
Two variable functions for the terrain
Quasi-periodic and aperiodic orbits

11.6 SPLIT SYNTHESIS

ACTIVITIES
- **Analyzing algorithms**
- **Completing algorithms**
- **Substituting parts of algorithms**
- **Correcting algorithms**

TESTING
- **Integrated cross-functional project: reverse engineering**

SUPPORTING MATERIALS
- **List of Max objects - List of messages and attributes for specific Max objects - List of GEN operators**

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

12T
MICROSOUND

12.1 GRANULAR SYNTHESIS
12.2 SYNCHRONOUS GRANULAR SYNTHESIS AND FORMANT SYNTHESIS
12.3 ASYNCHRONOUS GRANULAR SYNTHESIS
12.4 PARTICLE SYNTHESIS
12.5 GRANULATION AND SEGMENTATION OF SAMPLED SOUNDS

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

LEARNING AGENDA

PREREQUISITES FOR THIS CHAPTER
- Contents of volumes I and II, chapters 10 and 11

OBJECTIVES
Knowledge
- To understand the basics and learn about the various types of synchronous granular synthesis
- To learn the underlying mechanisms for timbre and pitch change in synchronous granular synthesis
- To learn the techniques of asynchronous granular synthesis and formant synthesis
- To learn various methods of granulating sampled sounds and performing particle synthesis
- To learn the concepts of stream, cloud, and intermediate granular configurations
- Conoscere le caratteristiche qualitative dei parametri e la loro randomizzazione
- To learn basic concepts about other techniques in the field of microsound: multi-source brassage and micromontage

Skills
- To be able to hear and identify changes in various key parameters applied to granular and particle synthesis techniques
- To be able to hear and identify changes in various key parameters used for the granulation of sampled sounds

CONTENTS
- Synchronous and asynchronous granular synthesis and formant synthesis
- Glisson synthesis, grainlet synthesis, trainlet synthesis, and pulsar synthesis
- Granulation of sampled sounds
- Multi-source brassage and micromontage

ACTIVITIES
- Sound examples

SUPPORTING MATERIALS
- Basic concepts - Glossary

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

"Beneath the level of the note lies the realm of microsound, of sound particles.
Microsonic particles remained invisible for centuries. Recent technological advances let us probe and explore the beauties of this formerly unseen world. Microsonic techniques dissolve the rigid bricks of music architecture — the notes — into a more fluid and supple medium. Sounds may coalesce, evaporate, or mutate into other sounds. (…)

When the particles line up in rapid succession, they induce the illusion of tone continuity that we call pitch. As the particles meander, they flow into streams and rivulets. Dense agglomerations of particles form swirling sound clouds whose shapes evolve over time…"
(Curtis Roads)[1]

This chapter focuses on using various techniques to create microsonic phenomena (i.e., short-duration sounds). Many authors have carried out experiments and created pieces of music based on microsounds, including physicist Dennis Gabor[2] and composers Iannis Xenakis,[3] Curtis Roads, Barry Truax, Horacio Vaggione, and Trevor Wishart.

[1] Roads, 2001, p.VII.
[2] Gabor, 1947.
[3] Xenakis, 1960.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

12.1 GRANULAR SYNTHESIS

Granular synthesis is a technique used to create audio signals based on the combination of sound particles called grains. This type of synthesis constitutes a very effective method for creating extremely rich and complex sound textures resulting from the generation, overlap, and spatialization of a great number of sounds of very short duration.

As we already mentioned in section 5.4 of the second volume, a **grain** is an extremely short sound event that generally lasts between 1 and 100 milliseconds. A grain is typically produced by an oscillator, with an envelope applied to its waveform – this is done to avoid instantaneous amplitude variations (resulting in clicks) as well as to shape its spectrum (see fig. 12.1).

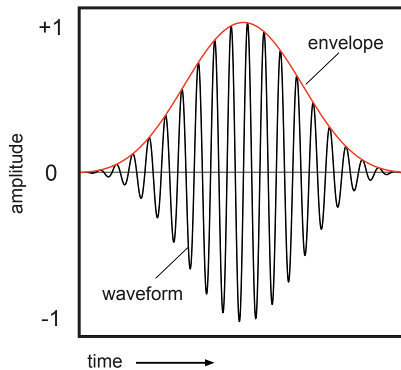


Fig. 12.1 A sine wave grain and an envelope

There can be pauses between each grain, in which case, we would hear them as single sound events, or there can be many overlapping grains, in which case, we would hear them as a single, uninterrupted sound stream. An example of this could be the sound of rain, which could be comprised of a few individual drops or a continuous sound caused by a very heavy rain. Barry Truax adopted the metaphor of water (although he used synthetic sounds) to compose "*Riverrun*," which is one of the seminal compositions in the field of granular synthesis. From this metaphor, we can infer that the traditional division between microstructure and macrostructure – especially in this type of synthesis technique – dissolves into a continuum (as Jean Claude Risset stated about granulation.[4]

[4] "By bridging gaps between traditionally disconnected spheres like material and structure, or vocabulary and grammar, software creates a continuum between microstructure and macrostructure. It is no longer necessary to maintain traditional distinctions between an area exclusive to sound production and another devoted to structural manipulation on a larger temporal level. The choice of granulation, or of the fragmenting of sound elements, is a way of avoiding mishaps on a slippery continuum: it permits the sorting of elements within a scale while it allows individual elements to be grasped. The formal concern extends right into the microstructure, lodging itself within the sound grain." (Risset, 2005).

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

SOUND EXAMPLE 12.1

– Excerpt from *Riverrun* by Barry Truax (from 0:00 to 5:00)[5]
– Excerpt from *Riverrun* by Barry Truax (from 15:30 to 19:44)

Other authors, such as Truax, Dodge, and Jerse, favored limiting the maximum duration of the grains in granular synthesis to 50 milliseconds. Above this duration, we begin to hear distinct grains, and the particular interdependence between time and frequency/spectrum tends to get lost.
To elucidate this concept, let's suppose we listen to the sound produced by a sine wave oscillator with a frequency of 500 Hz for a relatively short time (e.g., 100 milliseconds or more). In this situation, we are still able to perceive its frequency, so we would essentially hear a sound whose spectrum contains most of its energy at 500 Hz. But if we gradually decrease the duration of the sound, the distribution of the spectral energy will broaden around its 500 Hz center frequency, and we would subsequently start to perceive its pitch less clearly. In other words, the dispersion of the spectral energy around the frequency of the sound is inversely proportional to the duration of the sound itself – this is what we mean by interdependence between time and sound/spectrum. Therefore, the shorter the grain duration is, the wider the resulting frequency band (see fig. 12.2).

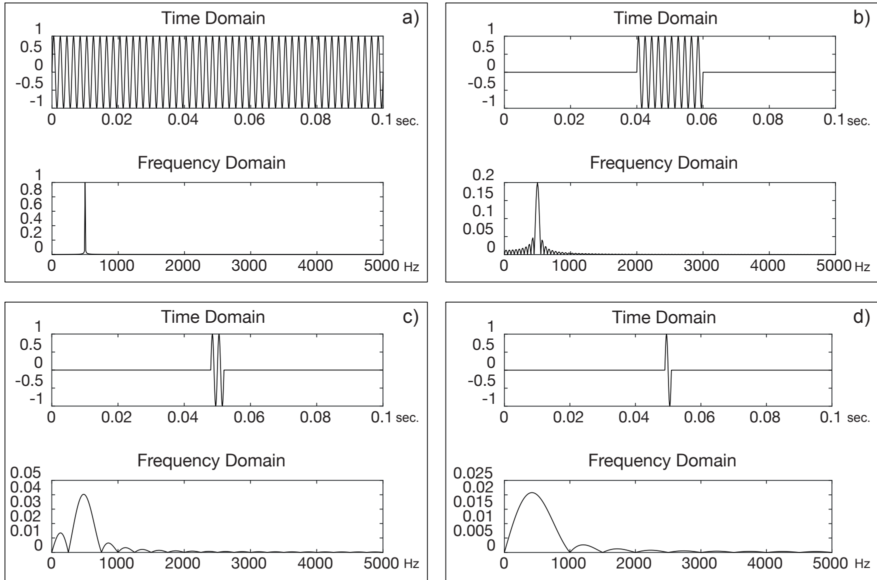


Fig. 12.2 Interdependence between time and sound/spectrum

[5] 24-bit stereo version courtesy of Barry Truax. The 16-bit stereo version was published on CSR-CD 8701 *Digital Soundscapes*. Cambridge Street Records. The original version is quadraphonic. Truax has subsequently produced an octophonic version for concert presentations.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

The discrete unit impulse (a single sample other than 0 followed by samples with value 0) represents an extreme case of this interdependence. As we learned in chapter 3, such an impulse has a spectrum that contains energy at all frequencies (namely, from 0 up to the Nyquist frequency).
Sound grains of very short duration (below 10 milliseconds) are consequently perceived as impulses whose spectral characteristics depend on grain duration, sound waveform, envelope shape, and sound frequency. Furthermore, for grains whose duration is the same, but whose frequency is different, we may perceive them as having higher or lower pitch.
Another characteristic is that we start to perceive pronounced amplitude modulation when grains with a duration of less than 50 milliseconds follow each other regularly.

As we previously learned about modulation in chapter 11, two sidebands are generated when a signal (the modulator) modulates the amplitude of another signal (the carrier). The components of these sidebands have frequencies equal to the sum and difference of the carrier and modulator components. In the case of granular synthesis, the carrier is the sound event to which we add the envelope, while the modulator is the flow of grain envelopes. As an example, let's suppose we have a regular grain stream with a unipolar sine wave envelope (**hanning window**, see below) lasting 20 ms (50 grains per second, i.e., 50 Hz) which modulates a 120 Hz sine wave. In this case, we would obtain a side component at 170 Hz (120 Hz + 50 Hz) and one at 70 Hz (120 Hz – 50 Hz), in addition to the 120 Hz frequency itself, so an inharmonic sound would therefore be generated. Of course, the longer the grain generation time, the closer the side components get to the carrier frequency until (below 20 Hz, with grains longer than 50 ms) we can no longer hear them as separate frequencies. If the envelope or the oscillator sound were not pure sine waves (as they are here), we would obtain multiple pairs of sidebands resulting from the sum and difference of all the components involved. We will come back to this idea in the next section and focus on how this amplitude modulation effect changes substantially when the phase of the carrier sound is reset to zero using a hard sync mechanism each time a new envelope is produced.
Granular techniques often use synthetic sounds (obtained through additive, table, FM synthesis, etc.) as the spectral content for the various grains. Each grain has its envelope, duration, waveform, timbre, distance in time from the next grain, and spatial localization. Additionally, each of these parameters can either be fixed or change over time. Managing the various parameters of so many events requires global control. (It is impractical to define the values of each parameter of each grain one by one, when the grains that will be generated and controlled every second can number in the thousands.) For this reason, we need to use an algorithm that has an automated process which allows us to generate grains and control the values of all the parameters globally. This high-level control enables us to define parameter trajectories and final destinations to be reached at the end of each trajectory. Such high-level global control allows the user to orient the flow of time in a precise direction or towards a specific focal point. This basically gives a coherent shape to thousands of small sound particles.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

GRAIN ENVELOPE

Grain envelopes are often symmetrical and come in a variety of shapes. Their shape can be based on a hanning window (which can be created using one cycle of a reversed unipolar cosine), or can be triangular, trapezoidal, Gaussian, quasi-Gaussian, etc. (see below). Two-stage asymmetrical envelopes can also be used (e.g., an instantaneous attack and exponential decay or vice versa, which Roads refers to as **expodec** and **rexpodec**, respectively). These shapes are called **window functions**, and the sound grain results from multiplying these window functions by the desired waveform.
This multiplication creates the amplitude modulation that was discussed earlier. Because each type of envelope has its own different spectral content, the choice of the envelope is key to defining the spectrum of the resulting grain.
Figure 12.3 shows the following window functions:

a) Hanning
b) Gaussian
c) Quasi-Gaussian (a Gaussian envelope with a sustain in the center portion)
d) Triangular
e) Trapezoidal
f) Expodec - instantaneous attack (less than 10 ms) and exponential decay
g) Rexpodec - exponential attack and instantaneous decay (less than 10 ms)
h) Bandlimited impulse (**sinc function**, i.e., sin(*x*)/*x*) – it creates a strong modulation effect)
i) Blackman
j) Welch - created using a single parabolic section

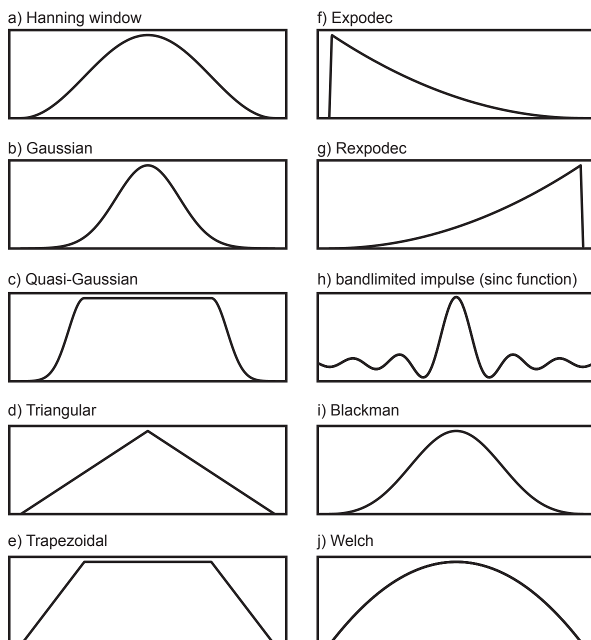


Fig. 12.3 Some examples of window functions (envelopes)

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

SOUND EXAMPLE 12.2 • Examples with various envelopes

– Hanning window
– Gaussian
– Quasi-Gaussian
– Triangular
– Trapezoidal
– Expodec
– Rexpodec
– Bandlimited impulse (sinc function)
– Blackman
– Welch

WAVEFORMS FOR GRANULAR SYNTHESIS

The waveform of the sound we want to granulate can vary over time, be constant, or even vary from one grain to another. It can be either periodic or aperiodic (even a noise generator could be used). Furthermore, the oscillator's wavetable could also be filled with a portion of a sampled sound – this can still be considered a synthesis technique and not sampled sound granulation (a topic we will deal with in section 12.5) in spite of the fact that a sampled waveform is used. We will discuss the differences between these in detail in section 12.5

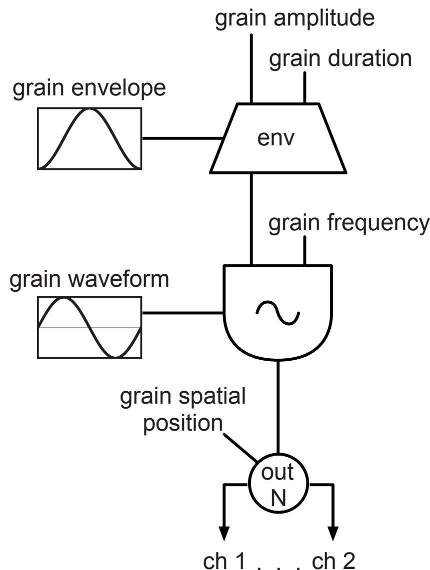


Fig. 12.4 A basic granular synthesis algorithm

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

CLOUDS AND STREAMS

As mentioned earlier, granular synthesis is performed by generating synthetic sounds and granulating them. Where the distribution of grains over time is concerned, there are two general temporal modes:

Synchronous granular synthesis. We will refer to this using the concept of "**Stream.**"
In synchronous granular synthesis, each grain follows the previous one. The grains can be spaced at regular or irregular intervals, and the linear flow is determined by a grain rate (that is, the number of grains per second). This grain rate can be constant or variable (for example, by using a glissando or generating random values within a limited range). Two consecutive grains may overlap (even partially via a crossfade), or there can be silence between each grain.

Asynchronous granular synthesis. We will refer to it using the concept of "**Cloud.**"
In this case, grains are distributed irregularly. They consist of particle agglomerations whose overall shape evolves over time with a statistical trend. (One of the properties of statistical evolution is event density.) Since grain generation in this context is not subject to a linear flow, it is more appropriate to use the term grain density instead of grain rate. A lower density corresponds to the generation of a few isolated events scattered over time and space. A higher density corresponds to denser grain clouds.
In the next sections, we will unveil the details and characteristics of both modes.

SOUND EXAMPLE 12.3 • Streams and clouds
a) Stream with a constant rate
b) Stream with a variable rate
c) Cloud with a low density
d) Cloud with a high density

(...)

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

other sections in this chapter:

12.2 SYNCHRONOUS GRANULAR SYNTHESIS AND FORMANT SYNTHESIS
The four types of synchronous granular synthesis
Using aperiodic signals
Probabilistic grain flow
Parameters used for synchronous granular synthesis
Random variation of the parameters
Formant synthesis: FOF

12.3 ASYNCHRONOUS GRANULAR SYNTHESIS
Using fragments of sampled sounds loaded in a table

12.4 PARTICLE SYNTHESIS
Glisson synthesis
Wavelet and grainlet synthesis
Trainlet synthesis
Pulsar synthesis

12.5 GRANULATION AND SEGMENTATION OF SAMPLED SOUNDS
Pointer and granular time-stretching
Variations in frequency
Variations in the grain rate
Variations in the duty cycle
Random variations in the grain rate and duty cycle
Combined variation of several parameters
Spatialized granular streams (voices)
Selective granulation
Multi-source brassage and micromontage

ACTIVITIES
- **Sound examples**

TESTING
- **Questions with short answers**

SUPPORTING MATERIALS
- **Fundamental concepts - Glossary**

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

12P
MICROSOUND

12.1 GRANULAR SYNTHESIS
12.2 SYNCHRONOUS GRANULAR SYNTHESIS AND FORMANT SYNTHESIS
12.3 ASYNCHRONOUS GRANULAR SYNTHESIS
12.4 PARTICLE SYNTHESIS
12.5 GRANULATION AND SEGMENTATION OF SAMPLED SOUNDS

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

LEARNING AGENDA

PREREQUISITES FOR THIS CHAPTER

- The contents of volumes I and II, Chapters 10 and 11 (theory and practice), Interlude F and Chapter 12T

OBJECTIVES

Skills

- To be able to program and use algorithms for synchronous and asynchronous granular and particle synthesis
- To be able to program and use algorithms for formant synthesis and the granulation of sampled sounds
- To be able to apply probabilistic processes and random variations to the various parameters of granular synthesis, particle synthesis, FOF synthesis and granulation
- To be able to program and use algorithms for multi-source brassage

Competence

- To be able to create a brief study based on granular synthesis, the granulation of sampled sounds and microsounds in general

ACTIVITIES

- Constructing and modifying algorithms

SUPPORTING MATERIALS

- List of Max objects - List of attributes and messages for specific Max objects - List of Gen operators and attributes

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

12.1 GRANULAR SYNTHESIS

As we saw in the theory chapter, the constitutive elements of a grain are the envelope and the waveform – or more correctly, the sound event "contained" in the envelope. Let's see how we can implement these elements in Max.

GRAIN ENVELOPE

Some of the envelopes that we will use in this chapter can be created directly by the **buffer~** object: this object can in fact generate or modify its contents in response to the messages it receives.
With the message *fill*, we can cause the **buffer~** object to fill its memory with specific values. The message [fill 1] causes all the samples of the buffer take on the value 1; [fill -0.5] fills the buffer with samples whose value is -0.5, and so on.
The message *apply*, on the other hand, applies a windowing function to the contents of the buffer. For instance, the message [apply hanning] makes a hanning window and [apply triangle] makes a triangular window. There are also other functions, such as *hamming*, *blackman* and *welch*.[1] As you can see, not all the windows we discussed in the theory chapter are available: we will address this issue in a moment.
Let's take a look at figure 12.1. It is important to note that window functions are applied to the content that is already present in the buffer; an "empty" buffer (that is, containing only samples with the value 0) would therefore not be modified.
Thus we use the message *fill* to fill the buffer with samples with a constant value, and we use the message *apply* to apply a window function to those samples.

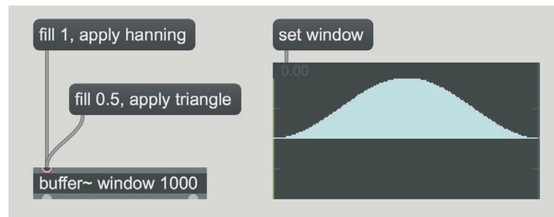


Fig. 12.1 Window functions with the **buffer~** object

The two messages in the patch shown in the figure (which we suggest that you replicate) create a hanning window and a triangular window (the latter with halved amplitude).
Add message boxes to the patch shown in figure 12.1 to create Hamming, Blackman, and Welch windows. Don't forget to use the *fill* message to fill the buffer with a constant value! Observe the different shapes of the various windows.

[1] There are other aspects of the messages fill and apply; we suggest that you study them by opening the **buffer~** object's help patch and the reference manual.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

We will now move on to functions that are impossible to create using the message *apply*. We have already seen, in section IB.9 of the first volume, how to store arbitrary envelopes in a buffer (and if you don't remember how, we suggest that you reread that section).
Let's see, for example, how to create a trapezoidal window. To simplify the task, we will start with a triangular window and then modify it. Make the patch shown in figure 12.2.

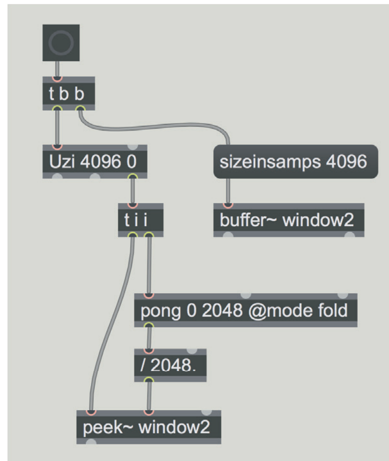


Fig. 12.2 The triangular window

This figure contains the object **pong**, which is the Max version of the MSP pong~ object discussed in section 11.4P. Like its MSP equivalent, this object can work in three modes: clip, wrap and fold (for a description of these modes, we refer you to the section mentioned above, 11.4P). It is different from pong~ in that the mode is not set by using a numerical argument but rather in textual form by using the @mode attribute.

The uzi object generates a series of values from 0 to 4095. These values are sent to the object [pong 0 2048 @mode fold], which "folds" all the values greater than 2048, thus generating a series that goes from 0 to 2048 and then goes back down. By dividing the output values by 2048, we obtain a triangular window that goes from 0 to 1.

With two simple operations, we can change the window from triangular to trapezoidal. Modify the patch as shown in figure 12.3 (add the objects * and clip).

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

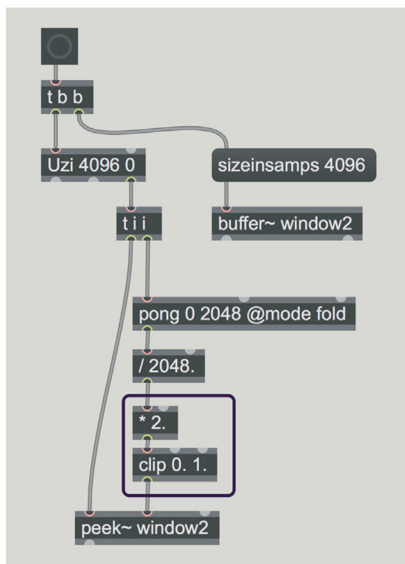


Fig. 12.3 The trapezoidal window

In this patch, the height of the triangle is doubled and then the `clip` object "cuts off" the upper part of the triangle, turning it into a trapezoid.

ACTIVITIES

- As we know from the theory chapter, the hanning window is made from a reversed unipolar cosine. Modify the patch shown in figure 12.2 so that it creates a hanning window (without using the messages *fill* and *apply*, of course!).
- The minor (upper) base of the trapezoid generated by the patch in figure 12.3 is half (50%) the length of the major base. If we multiplied the height of the triangle by 3 instead of by 2, the minor base of the resulting trapezoid would be 2/3 (66.666%) of the major base; if we multiplied it by 4, the minor base would be 3/4 (75%) of the major base. Add a small algorithm to the patch that allows you to set the upper base as a percentage of the lower base: in other words, the value 50 should generate the multiplier 2; the value 66.666 should generate the multiplier 3, and the value 75 should generate the multiplier 4 (needless to say, the other values should generate the appropriate multipliers as well).

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

And now for the good news: the Virtual Sound Macros library contains an object which allows us to create all the window functions we will need for granular synthesis – `vs.winfunc`. Open the file **12_01_winfunc.maxpat** (fig. 12.4).

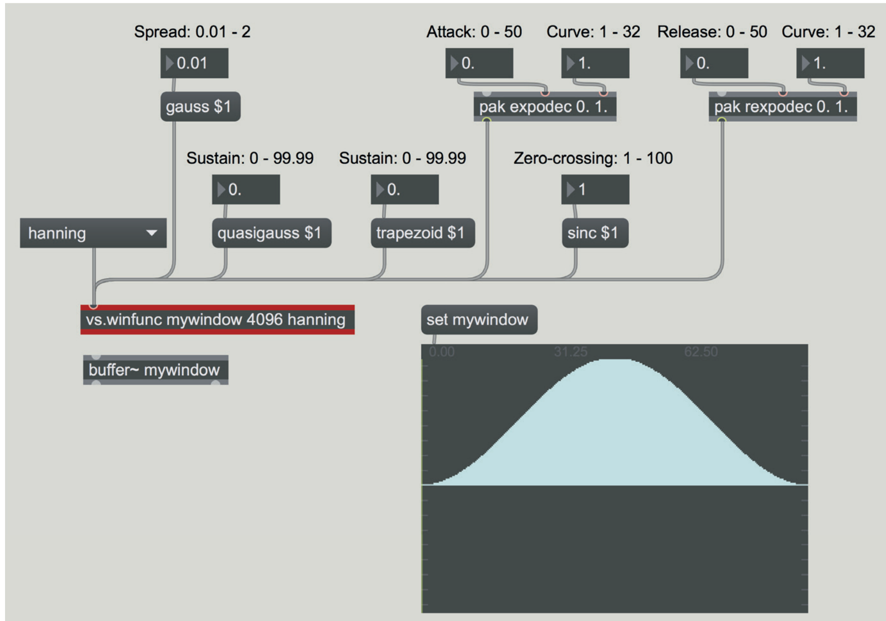


Fig. 12.4 The file **12_01_winfunc.maxpat**

This object can take the following (optional) arguments:
- the name of the buffer;
- the size of the buffer in samples;
- the name of the window function, followed by parameters in some cases.

– It is also possible to send the name of the function and the corresponding parameters to the object's inlet. Here is a list of the available functions:
– - hanning, no parameter;
– - gauss, spread parameter (the width of the bell) from 0.01 to 2;
– - quasigauss, sustain parameter from 0% to 99.99%;
– - triangle, no parameter;
– - trapezoid, sustain parameter from 0% to 99.99%;
– - expodec, two parameters: attack, from 0% to 50%, and curve (exponential factor) from 1 to 32;
– - rexpodec, two parameters: release, from 0% to 50%, and curve from 1 to 32;
– - sinc, zero-crossing parameter (how many times zero is crossed) from 1 to 100;
– - blackman, no parameter;
– - welch, no parameter.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Try all the functions in the `umenu` on the left. Then, for the functions that accept parameters, observe how the envelopes change as you modify the values in the number boxes.

ACTIVITY
Open the `vs.winfunc` object with a double-click (it is an abstraction, like all the other objects in the Virtual Sound Macros library) and compare the algorithm that generates the trapezoidal function with the analogous algorithm that we saw in fig. 12.3. Then analyze the functioning of the algorithms that generate the expodec and rexpodec functions.

WAVEFORMS FOR GRANULAR SYNTHESIS

In practice, nearly all the sound generators discussed in the previous chapters can be used for granular synthesis: from simple sine wave oscillators to bandlimited oscillators, from vector synthesis to noise generators, from tables created through additive synthesis or filled with fragments of sampled sounds to resonances produced by filtered impulses, and of course all the sounds produced by the various techniques of nonlinear synthesis.
As we progress through the chapter, we will see how and when to use the various sources.

(...)

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

other sections in this chapter:

Single-sideband modulation (frequency shifting)

12.2 SYNCHRONOUS GRANULAR SYNTHESIS AND FORMANT SYNTHESIS
The four types of synchronous granular synthesis
Using aperiodic signals
Probabilistic grain stream configurations
Modulating parameters with envelopes
Other parameters of synchronous granular synthesis
Duty cycle
Grain duration
Glissando
Random parameter variation
The spatial disposition of grains: panning and reverberation
The use of polyphony: voices
Overlapping grains in a single stream
Formant synthesis

12.3 ASYNCHRONOUS GRANULAR SYNTHESIS
Using fragments of sampled sounds loaded in a table

12.4 PARTICLE SYNTHESIS
Glisson synthesis
Wavelet/grainlet synthesis
Trainlet synthesis
Pulsar synthesis

12.5 GRANULATION AND SEGMENTATION OF SAMPLED SOUNDS
Synchronous granulation
Brassage, multi-source brassage, and non-real-time sound processing
Real-time signal granulation
Circular buffer
The double buffer

ACTIVITIES
- **Analyzing algorithms**
- **Completing algorithms**
- **Substituting parts of algorithms**
- **Correcting algorithms**

TESTING
- **Integrated cross-functional project: reverse engineering**

SUPPORTING MATERIALS
- **List of Max objects - List of messages and attributes for specific Max objects - List of GEN operators**

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

13T
ANALYSIS, RESYNTHESIS, AND CONVOLUTION

13.1 THE VOCODER
13.2 THE FOURIER TRANSFORM
13.3 SIGNAL PROCESSING IN THE FREQUENCY DOMAIN: THE PHASE VOCODER
13.4 TIME STRETCHING AND PITCH SHIFTING WITH PHASE VOCODER
13.5 CONVOLUTION AND CROSS-SYNTHESIS
13.6 CONVOLUTION REVERB

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

LEARNING AGENDA

PREREQUISITES FOR THIS CHAPTER
- CONTENTS OF VOLUMES I AND II, CHAPTERS 10, 11 AND 12

OBJECTIVES
Knowledge
- To learn the basics of the vocoder technique
- To learn the basics of the theory of fourier transform
- To learn the basics of sound processing through bin modification
- To learn the basics of the analysis-and-resynthesis techniques
- To learn the basics of the phase vocoder technique
- To learn the basics of the convolution technique
- To learn the basics of cross synthesis through convolution
- To learn the basics of the convolution reverb technique
- To learn the basic concepts of convolution with microsounds

Skills
- To be able to hear and identify the different effects that can be achieved with a phase vocoder
- To be able to hear and identify the vocoder effect on voice sounds
- To be able to hear and identify the difference between the effects of a vocoder and a harmonizer
- To be able to hear and identify various effects obtained through cross-synthesis

CONTENTS
- Basic theory of the vocoder
- Basic theory of the fourier transform
- Basic theory of analysis and resynthesis through phase vocoding
- Basic theory of convolution
- Basic theory of cross-synthesis through convolution
- Basic theory of convolution with microsounds
- Basic theory of convolution reverbs

ACTIVITIES
- Sound examples

SUPPORTING MATERIALS
- Basic concepts - Glossary

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

SYNTHESIS TECHNIQUES BASED ON SOUND ANALYSIS

One of the most exciting techniques for sound processing is known as **analysis and resynthesis** (or *synthesis-by-analysis*). This is ostensibly a synthesis technique, though it is based on data previously collected via sound analysis.
Let's take a quick look at the basic process, which can either be real-time or calculated offline.

1) An audio file or a real-time sound is analyzed using an analysis algorithm. The results are either stored in an analysis file or stored in memory, so they can be processed and sent to the resynthesis algorithm in real-time.
2) The data stored in the analysis file or processed in real-time can then be modified so that some sound characteristics are no longer the same as the original ones.
3) The processed data is then used as the basis for sound resynthesis (i.e., at this stage, the analysis data is used to create a new sound file or live sound). The new sound will also reflect any changes made to the analysis data.

Depending on the type of method used, the analysis file can be modified in a multitude of different ways in order to obtain a wide variety of interesting types of sound processing and interaction between different sound files.
This type of dynamic processing allows effects such as being able to lengthen or shorten the duration of a sound while simultaneously modifying its frequency independently, to perform spectral stretching, to modify individual components and their envelopes, and to create spectral morphing effects between one sound and another, among others.
There are many different techniques used for sound analysis, and there is not a single ideal one.[1] Some of these are based on Fourier analysis, a technique which is of paramount importance and which will be discussed in more detail in the second section of this chapter.
The choice of one technique over another depends on the spectral characteristics of the type of sound to be analyzed and indeed also the resulting sound we ideally wish to obtain. Many composers have been working with analysis and resynthesis techniques since the 1980s, with striking results. In particular, we would like to mention four compositions based on resynthesis systems applied to the human voice and other sounds: *Vox 5* by Trevor Wishart (1986) and *Study in White* by Joji Yuasa (1987), both of which employed the phase vocoder technique, *Idle Chatter* by Paul Lansky (1990) (in which the analysis and resynthesis technique used was Linear Predictive Coding, LPC),[2] *Mortuos*

[1] Among the many available systems are: analysis and resynthesis with phase vocoder (based on FFT), analysis and resynthesis with heterodyne filter, subtractive and LPC (Linear Predictive Coding) analysis and resynthesis, analysis and resynthesis with mixed technique (deterministic and stochastic, see Serra, X. 1989), SMS (Synthesis by Spectral Modeling), and ATS (Analysis-Transformation-Synthesis), and wavelet synthesis.
[2] Linear predictive coding is an advanced technique for analyzing voice sounds which produces the coefficients of an all-pole filter. These coefficients can then be used to simulate the response of the vocal tract in order to resynthesize the voice sound in a different way by driving the filter with a complex (harmonic or inharmonic) excitation source.The first experiments were proposed in 1966 in Japan by Shuzo Saito and Fumitada Itakura. For further information, see Makhoul, 1975.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Plango Vivos Voco by Jonathan Harvey (1980) (in which FFT-based analysis and resynthesis techniques were used on bell sounds and processed recordings of a boy soprano's voice using the CHANT system).[3] A more recent composition by Jonathan Harvey that uses a different concept of analysis and resynthesis is *Speakings* (composed in 2007-2008).[4]

SOUND EXAMPLE 13.1

a) Excerpt from *Idle Chatter* by Paul Lansky (from 0:00 to 3:29[5]
b) Excerpt from *Vox 5* by Trevor Wishart[6]
c) Excerpt from *Inferno* by Edison Studio (from 15:46 to 16:15 and from 16:51 to 17:40)[7]
d) Excerpt from *Mortuos plango*, *vivos voco* by Jonathan Harvey (from 1:42 to 4:42).[8]

[3] The CHANT vocal synthesis system was developed by Gerald Bennett and Xavier Rodet.

[4] *Speakings* is a composition for live electronics and orchestra written by Jonathan Harvey "with the artistic aim of making an orchestra speak through computer music processes." (Nouno, G., et al., 2009). The primary tool used for assisted orchestration was *Orchidée*. The current version of the *Orchid* systems developed at IRCAM in Paris (based on an original idea of Yan Maresz and the IRCAM Orchestration Workgroup) is called *Orchidea*, a framework for static and dynamic assisted orchestration developed by Carmine Emanuele Cella within a collaborative project between IRCAM (Music Representation Team), HEM, and UC Berkeley. (See Gillick, J. et al., 2019, Cella, C.E., 2020 and 2021.)

[5] CD BCD 9050. Courtesy of Paul Lansky.

[6] Courtesy of Trevor Wishart.

[7] *Inferno* is the first Italian feature film (1911) by F.Bertolini, A.Padovan, and G. De Liguoro. It is a silent film about the first canticle of Dante's Divine Comedy. The composers' collective Edison Studio (formed by Mauro Cardi, Luigi Ceccarelli, Fabio Cifariello Ciardi, and Alessandro Cipriani) composed an electroacoustic soundtrack for the movie in which voices, ambient sounds, and music blend together and interchange functions. The film with this new surround soundtrack was published in 2011 in the Book-DVD CR/10 Cineteca di Bologna, Collection Il Cinema Ritrovato. (See Cipriani et al., 2004 and Gazzano, ed. 2014.)

[8] CD Sargasso, SCD 28029. Stereo version. Courtesy of Sargasso. Harvey first composed it as an 8-channel composition and later created a 4-channel version. "The two sound sources are the voice of my son and that of the great tenor bell at Winchester Cathedral, England." (Harvey, 1981).

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

13.1 THE VOCODER (OR CHANNEL VOCODER)

"**Vocoder**" is a portmanteau (a compound word) formed from a combination of the two words voice and encoder (or coder). This type of processing (best known as the "speaking orchestra" or "robotic voice" effect since the '1970s) allows a wide-spectrum sound to be modified using the time-varying spectral envelope of a speaking voice signal. Homer Dudley invented the prototype vocoder in 1928, not for musical purposes but rather as part of a research project on encoding and encrypting the human voice for telephone transmission.[9] It was not until 1948 that Werner Meyer-Eppler (later known as one of the founders of the WDR studio in Cologne) published a thesis in which he applied the same techniques to music. The first musical instrument to include a vocoder was the Siemens Synthesizer in the late 1950s, followed by the first vocoders made by Robert Moog in the late 1960s.
The vocoder we will discuss in this section is implemented in the time domain. It is also called the channel vocoder and is different from the phase vocoder, which is implemented in the frequency domain and will be covered in subsequent sections.

The overall structure of a vocoder can be divided into two parts, namely the encoder and the decoder. Before entering the encoder, the voice sound is divided into vowels and consonants. The latter are sent directly to the output, while the vowels will serve as the input signal for the ***encoder***, which contains a bank of parallel bandpass filters. This separation is performed for better voice intelligibility. If consonants are filtered out, they become less intelligible, and a "hum" (which is generally undesirable) can be generated as they pass through the filters. A ***zero crossing detector*** – which calculates how many times a waveform crosses the zero point (i.e., between positive and negative sample values) – can be used to achieve this separation. Consonants tend to have more frequent zero crossings than vowels because they contain a more significant number of high-frequency components. In the zero-crossing detector, a threshold value is set for the amount of zero crossings that the signal's spectral energy must exceed in order for a specific sound to be classified as a consonant.[10]

[9] At the New York World's Fair in 1939, the "voder" was shown. This was an instrument used to synthesize the voice. Unlike the vocoder, no microphones were used, but it nonetheless featured a similar decoder. Various sounds and vocal pitches could be recreated using impulse trains, filtered noises, and pure mechanical manipulation to make the machine "speak" by controlling it with a series of keys and hardware controls. The vocoder we will be discussing here, on the other hand, is a voice processing tool that requires a real voice (e.g., a signal from a microphone or a recorded/sampled sound) and a series of controls.

[10] The detector performs this calculation by comparing the current sample with the previous one to check if the signal has crossed from the positive range to the negative one or vice versa.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

This threshold value can be different for different voices, so it should be determined (through trial and error) by listening to the specific sound of the voice which will be used. Based on this setting, if the output value exceeds the threshold, the sound will be classified as a consonant, otherwise, the detector will consider it a vowel. When the signal exceeds this threshold, it is sent directly to the output (or to a noise generator). Conversely, if the signal does not exceed the threshold, it is sent to the encoder, where it – the part of the voice containing the vowels – is then further "broken down" by the encoder into a certain number of frequency bands.

The number of bands relative to the frequency zones, as well as their different widths, affect the quality of the result (we will see how to fine-tune this in the practice chapter). For each frequency band, a series of envelope followers placed at the output of each filter is used to measure that band's amplitude level over time. This way, a representation of the spectral energy can be obtained for the voice sound over time. The decoder uses the envelopes generated by the envelope followers as control signals for an additional bandpass filter bank, which contains the same number and type of filters as the encoder.
Another sound with a rich spectrum (such as noise, a sawtooth wave, or an orchestral sound, etc.) is sent to the inlet of this second filter bank, whose amplitude is controlled by the envelopes mentioned above. This sound is then filtered according to the evolution of the voice's spectral energy over time.

The particular reason that this effect that gives a so-called "synthetic" quality to the voice is that information about the fundamental frequency of the voice at any given point in time is neither obtained nor calculated. For instance, if the second signal (the one being filtered) is a chord played by an orchestra, once the that sound is processed by the vocoder, the fundamental frequencies of the resulting sound will be those of the orchestral chord and not those of the original voice. Of course, it is also possible to mix the output sound of the vocoder with the sound of the dry voice – doing so would naturally depend on the musical goals we want to achieve. We could also use the vocoder with sounds other than the human voice, such as a drum kit (as demonstrated in sound example 13.2 k) or other monophonic sounds. We could even use a MIDI-controlled virtual instrument, such as a sampled piano, as a decoder for a voice sound input. In this scenario, the encoder will contain several filters that correspond to the piano keys, tuned at an equal-tempered semitone distance from each other. The amplitude values generated by the encoder will be converted into velocity values, and these values will then be mapped to the corresponding MIDI notes. This way, by playing notes on the virtual piano at specific velocities, we can make the piano appear to "speak". (You can hear this effect in sound example 13.2 l with some parameter variation).
As a second signal, instead of a sampled sound, you could also use a complex sound created through additive synthesis (produced by adding sine waves or complex waveforms together). Using this technique, you can control the output sound in a more versatile and user-definable way. (You can hear this effect in sound example 13.2 m, in which you first hear a dry flute sound and then the flute sound with the vocoder, with some additional variation in the decay and gain).

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Figure 13.1 shows a diagram that represents the signal flow of a vocoder.

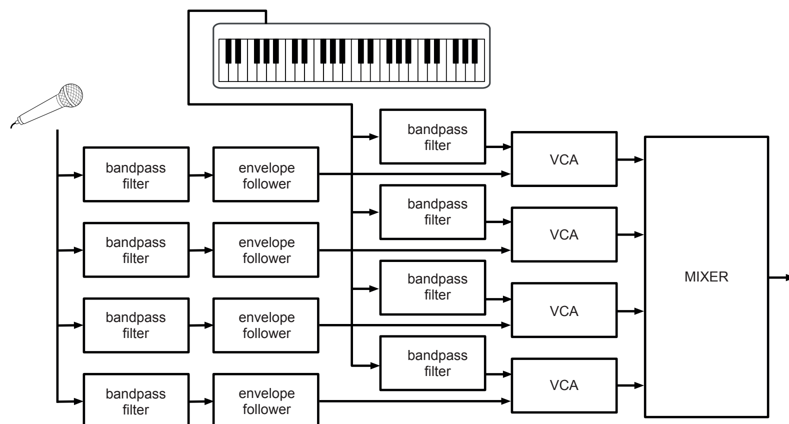


Fig. 13.1 the signal flow diagram of a vocoder

Furthermore, an interesting mixed technique combines the vocoder output with the harmonizer technique (already discussed in sections 8.3T and 6.9P).[11] Remember that the harmonizer allows us to create two, three, or more pitches that can be tuned to user-defined intervals in relation to the original sound of the voice (and therefore also to its fundamental). The difference between this and the vocoder is that, in the harmonizer, these intervals are not relative to a second signal but are created by sound transposition factors. Therefore, the frequencies of the harmonized copies continuously follow the fundamental frequency at a distance determined by the user-defined interval between the original and each copy. As an example, let's take the sound of a voice which sings a glissando from C to F and then to G and is simultaneously sent both to the harmonizer and the vocoder. If we set the transpositions of the harmonizer to a fourth, fifth, and ninth, and – at the same time – we also send a sound containing a C-E-G-Bb seventh chord to the vocoder decoder, the same sung words will produce both fixed chords generated by the vocoder and constantly changing chords created by the harmonizer, relative to the evolving fundamentals of the original sound. This method also effectively turns a spoken voice (with continuous instantaneous changes in the fundamental) into a source of non-traditional harmonies perceived within a harmonic context determined by the vocoder.

[11] We described a similar technique in section 8.4, in which we combined the harmonizer with resonant filters. Alessandro Cipriani used these mixed-voice processing techniques in the composition *Il Pensiero Magmatico* (Cipriani-Taglietti, 1996) at the Edison Studio, employing the MARS (Musical Audio Research Station) real-time digital synthesis system. MARS was developed in 1991 by a team of various researchers, including Giuseppe Di Giugno, Emmanuel Favreau, Eugenio Guarino, Andrea Paladin, and Sylviane Sapir, at IRIS in Paliano (See Cavaliere et al. 1992, Palmieri et al. 1992, and Andrenacci et al., 1997.)

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

SOUND EXAMPLE 13.2

a) The sound of a dry spoken voice
b) The sound of a granulated orchestra
c) A vocoder that uses a) and b)
d) Sound c) mixed with sound a)
e) A singing voice
f) Singing voice e) with vocoder
g) Singing voice e) with harmonizer
h) The singing voice with harmonizer and vocoder
i) The spoken voice with harmonizer and vocoder
j) Excerpt from the electronic part of *Il Pensiero Magmatico* by A.Cipriani and S.Taglietti. Mixed technique with *vocoder+harmonizer*[12]
k) Drum kit+vocoder with sound b). Variations in the decay and Q factor
l) MIDI vocoder. Electronics+MIDI piano. Variations in the gain and Q factor
m) Dry flute followed by flute sound with vocoder and variations in the decay and gain.

(...)

[12] Excerpt from track 11 *Nel magma incandescente* (from 2:30 to 5:14). The complete version for piano, percussion instruments, mixed choir, and electronics was published on PAN CD 3059. Courtesy of EDI-PAN and Stefano Taglietti.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

other sections in this chapter:

13.2 THE FOURIER TRANSFORM
The Fourier theorem
The Fourier transform
The Short-Time Fourier Transform (STFT)
Guidelines for the analysis

13.3 SIGNAL PROCESSING IN THE FREQUENCY DOMAIN:
The phase vocoder
Brickwall filters
Spectral lfo
Reduction through a spectral noise gate
Bin shifting (spectral frequency shifting)
Multiplication of bin indices (spectral pitch shifting)
Spectral stretching/shrinking
Bin feedback delay (spectral delay)
Phase modification and phase stop
Bin phase randomization
Freeze
Cross-synthesis with the stft

13.4 TIME STRETCHING AND PITCH SHIFTING WITH PHASE VOCODER
Fast wavelet transform

13.5 CONVOLUTION AND CROSS-SYNTHESIS

13.6 CONVOLUTION REVERB

ACTIVITIES
- **Sound examples**

TESTING
- **Questions with short answers**

SUPPORTING MATERIALS
- **Fundamental concepts - Glossary**

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

13P
ANALYSIS, RESYNTHESIS AND CONVOLUTION

13.1 THE VOCODER
13.2 THE FOURIER TRANSFORM
13.3 SIGNAL PROCESSING IN THE FREQUENCY DOMAIN: THE PHASE VOCODER
13.4 TIME STRETCHING AND PITCH SHIFTING WITH THE PHASE VOCODER
13.5 CONVOLUTION AND CROSS-SYNTHESIS
13.6 CONVOLUTION REVERB

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

LEARNING AGENDA

PREREQUISITES FOR THIS CHAPTER
- The contents of volumes I and II, Interlude F, Chapters 10, 11, 12 (theory and practice) and Chapter 13T

OBJECTIVES
Skills
- To be able to program and use algorithms based on the vocoder technique
- To be able to program and use algorithms based on the Fourier transform that use bin manipulations to process sounds
- To be able to program and use sound processing algorithms based on the techniques of analysis and resynthesis, in particular those of the phase vocoder
- To be able to program and use algorithms for sound processing by means of convolution, including cross-synthesis, convolution reverb and convolution with microsound techniques

Competence
- To be able to create a brief study based on the use of convolution and the phase vocoder

ACTIVITIES
- Constructing and modifying algorithms

SUPPORTING MATERIALS
- List of Max objects - List of attributes, arguments and messages for specific Max objects - List of Gen operators

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

13.1 THE VOCODER

Let's begin our discussion by seeing how to create an encoder. First, we need a filter bank, i.e., the `fffb~` (fast fixed filter bank) object, which we discussed in section 3.7P of the first volume. This object normally has as many outlets as filters in the bank, so for the sake of simplicity, we will use the multichannel version `mcs.fffb~`, which combines all the filters into a single multichannel output (see figure 13.1).

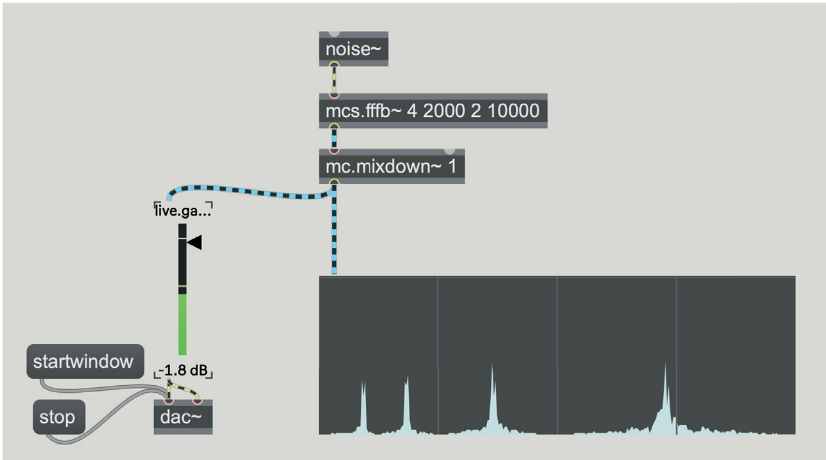


Fig. 13.1 The `mcs.fffb~` object

The object has four arguments: the first indicates the number of filters in the bank; the second, the frequency of the first filter; the third, a multiplication factor for the subsequent filters (in the case illustrated above, the multiplication factor is 2, which means that each filter will have twice the frequency of the previous one, or in other words they will progress in octaves); and the fourth argument represents a Q factor applied to all the filters in the bank.
As we learned in section 13.1T, we need to know the amplitude level output by each filter; and we can get this information by using the **`avg~`** object. This object receives a signal and a series of bangs. For each bang received, the object reports the average (absolute) value of the input signal (in the form of a Max number), calculated since the previous bang (see figure 13.2).

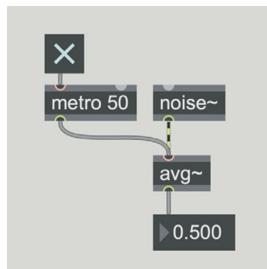


Fig. 13.2 The `avg~` object

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

In the patch **13_01_encoder.maxpat** (figure 13.3), we use these objects to make an encoder.

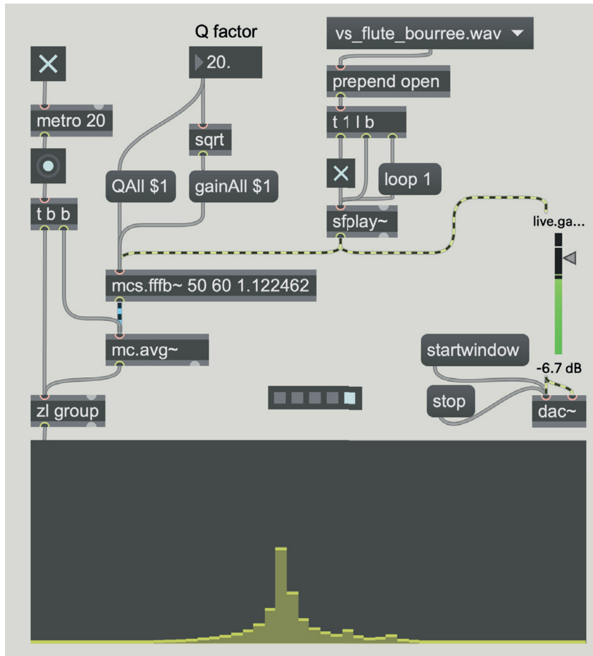


Fig. 13.3 The patch **13_01_encoder.maxpat**

An audio signal is sent to the **mcs.fffb~**object. This object contains a bank of 50 filters; the frequency of the first filter is 60 Hz, and the ratio between the filters is 1.122462 – i.e.,$\sqrt[6]{2}$, which is an increment of one whole step between filters. We have, in other words, a sixth-octave filter bank (each octave is divided into six parts). The fiftieth and last filter will have a frequency of approximately $60 * (\sqrt[6]{2})^{49} = 17{,}241$ Hz.
Notice that we raise the tone ratio to the power of 49 and not 50; this is because the multiplication series starts with the second filter: we could say that the first filter has a frequency of $60 *(\sqrt[6]{2})^0 = 60$ Hz.
With the number box at the top, we control the Q factor (message "QAll") and the gain (message "gainAll") of all the filters; the gain is set to the square root of the Q factor to compensate for the energy loss that a high Q entails.[1]
The multichannel signal generated by **mcs.fffb~** is sent to the **mc.avg~** object, which receives a bang every 20 milliseconds from a **metro**. For each bang, **mc.avg~** generates the 50 values corresponding to the 50 channels; these values are sent to a [**zl** group] object, which groups them into a list and sends them to the **multislider** object at the bottom. What is displayed in the **multislider** is in effect the spectral profile of the sound sent to the filter bank.

[1] See section 3.3P of the first volume.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Note that when the Q factor is very high, the bandpass filters of `mcs.fffb~` tend to produce resonances that fade away slowly, as we know, and this slows down the movements of the spectral profile displayed in the `multislider`. We can use the Q factor, therefore, to adjust the spectral profile's rate of change. With this in mind, compare the first preset, which sets the Q-factor to 20, with the second, which sets it to 1000.

With a decoder, we can use the spectral profile obtained to shape a second broad-spectrum sound. Let's look at a full example in the patch **13_02_vocoder.maxpat** (figure 13.4).

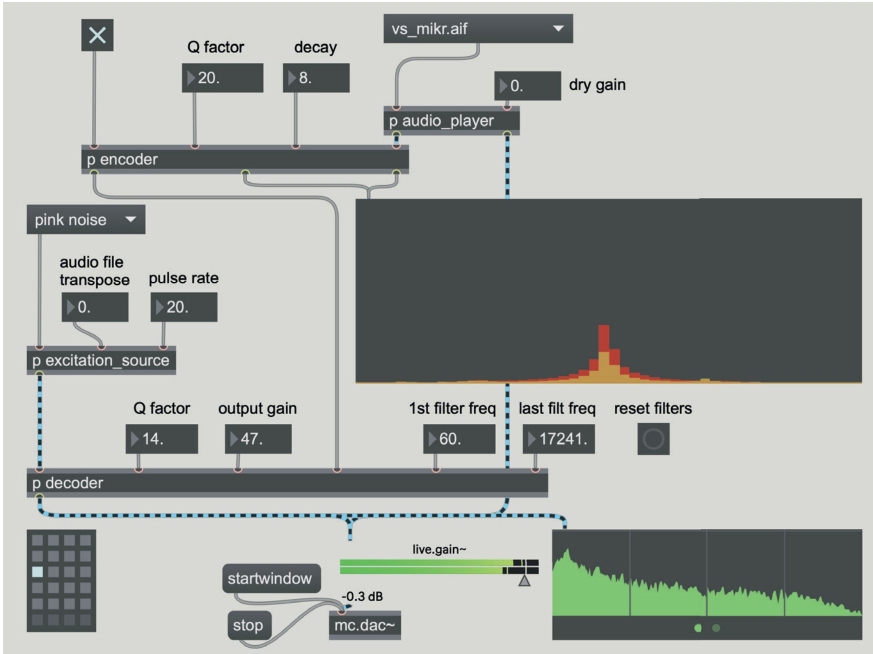


Fig. 13.4 The patch **13_02_vocoder.maxpat**

Before analyzing the contents of the subpatches, let's examine the parameters shown in the figure.
The subpatch "encoder," at the top, contains a modified version of the previous patch; the two numerical parameters are the Q factor and the decay of the envelope followers (one for each filter). The `toggle` controls the internal `metro` object, and if it is turned off, the spectrum remains "frozen." The `umenu` connected to the subpatch "audio_player" allows us to choose the audio file to send to the encoder.
On the left, in the middle of the figure, the subpatch "excitation_source" allows us to select the source used to excite the filter bank; the options are a granulated orchestral sound, a pink noise generator and an impulse generator.
Finally, at the bottom, the subpatch "decoder" contains the filter bank that receives both the signal of the excitation source and the spectral profile from the subpatch "encoder" at the top. We can adjust the Q factor and the gain

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

of this filter bank, and we can also set the frequencies of its first and last filter (thereby modifying them with respect to those of the filter bank of the encoder): inside the subpatch, as we will see, is an algorithm that determines the correct multiplication factor for the filters in the bank. Clicking the button "reset filters" in the lower right returns the filters to the same frequencies as those of the encoder's filter bank.
Listen to all the presets and observe the parameters. Try turning the `toggle` at the top left off and on to "freeze" the spectral profile produced by the encoder. Modify the parameters' values to create new presets.

Now let's look at the contents of the subpatch [`p` encoder], which, as we said, is derived from the patch **13_01_encoder.maxpat**. Double-click to open the subpatch (figure 13.5).

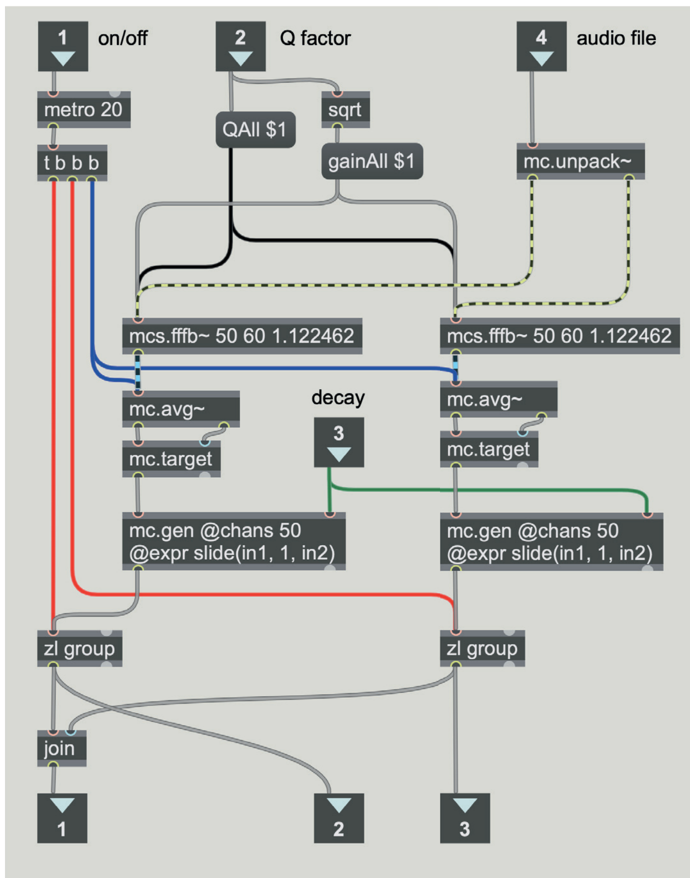


Fig. 13.5 The subpatch [`p` encode]

The audio file to be analyzed – received by the fourth `inlet`, in the upper right corner – is stereo; the two channels are sent to two identical filter banks. As in the previous patch, each bank contains 50 filters that are a whole step apart, and the frequency of the first filter is 60 Hz.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

We make 50 envelope followers by calculating the average amplitudes of the filters with **mc.avg~**, and then sending the resulting values, via **mc.target**, to the **mc.gen** object, which implements the **slide** operator[2] using the attribute @expr. The third **inlet** of the subpatch allows us to set a decay factor for the envelope followers (the "slide-down" parameter of **slide**), which, as you can hear by listening to the presets, adds a reverb tail effect to the sound of the decoder.
The values from the two banks of envelope followers are grouped together and sent to the subpatch containing the decoder. These values are also sent through the second and third outlets to two superimposed **multislider** objects on the right side of the main patch, which display the spectral profiles of the two audio channels.
Let's turn to the subpatch [p decoder], shown in figure 13.6.

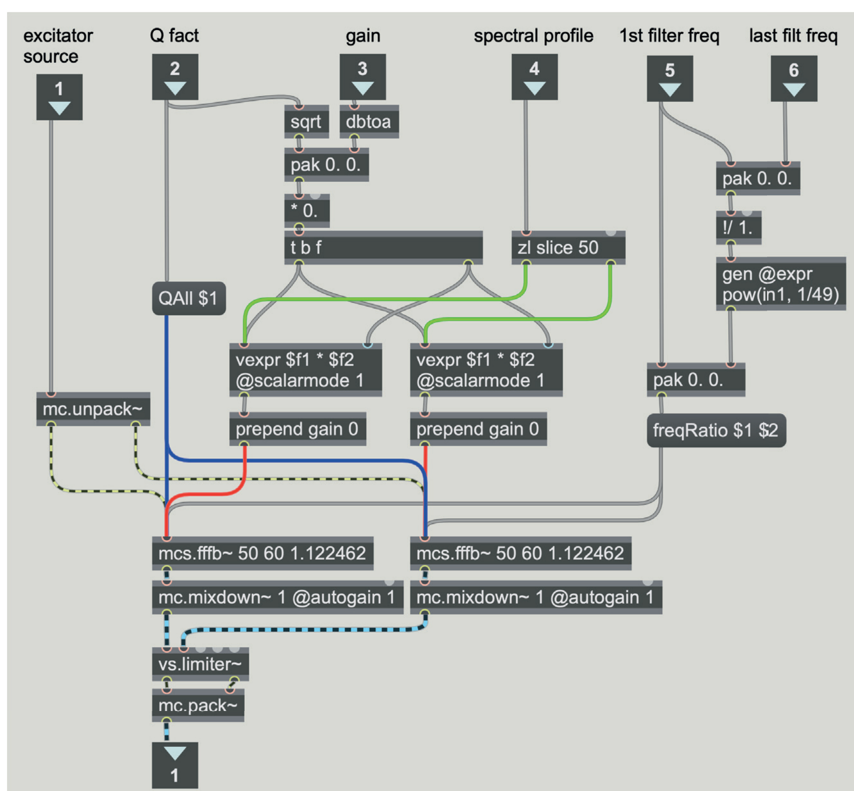


Fig. 13.6 The subpatch [p decoder]

At the bottom are the two **mcs.fffb~** filter banks for the left and right audio channels of the stereo output. These two objects receive all the signals and messages coming from the **inlets**. Let's look at them one by one.

[2] This is, of course, the Gen version of the **slide~** object, which we know from section 7.1P of the second volume (where we discussed envelope followers).

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contempornet 2023 - All rights reserved

The first **inlet** receives the excitation source. The second receives the Q factor. The third receives a gain factor, which is then multiplied by the square root of the Q factor; this last operation, as we have already explained, serves to compensate for the energy loss due to the selectivity of the Q factor. The resulting value is used to rescale the values of the spectral profile received from the encoder (fourth **inlet**).
The last two **inlets** receive the frequency values to be assigned to the first and last filters of the bank. First we calculate the ratio between the frequencies of the last filter and the first filter, using the object [**!**/1.]: this operation gives us the multiplication factor to apply to the frequency of the first filter in order to obtain that of the last filter. Next we calculate the 49th root of this value: this gives us the multiplication factor for the filters; applied 49 times in a row, it will take us from the frequency of the first filter to that of the last filter. This factor is used, together with the frequency of the first filter, as a parameter for the two **mcs.fffb~** objects (message box [freqRatio $1 $2]).

ACTIVITIES

- Change the number of filters in the encoder and decoder banks (they must be the same, of course), and modify the subpatches that employ them accordingly. Listen carefully to the results produced by increasing or decreasing the number of filters (a smaller number of filters does not necessarily sound "worse").
- Add a "gate" immediately after the **mc.avg~** objects in the subpatch [**p** encoder] (figure 13.5), so that amplitude values below a certain threshold are set to zero.
- Simulate a "brickwall filter" in the decoder by setting to 0 the amplitudes of the first filters (highpass brickwall) and/or those of the last filters (lowpass brickwall). You will of course need two parameters to indicate the first filter and the last filter that will actually be used (the filter cutoffs).
- Simulate "comb filtering" by zeroing one filter out of two, two filters out of three, etc.

The decoder in the previous patch works by subtractive synthesis: we take a sound with a spectrum rich in components (a noise, a full orchestra, etc.) and we shape it using filters.
Is an additive synthesis decoder possible? Certainly: all we need to do is to replace the filter bank with an oscillator bank – which is just what we do in the patch **13_03_additive_vocoder.maxpat** (figure 13.7).

(...)

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

other sections in this chapter:

13.2 THE FOURIER TRANSFORM

13.3 SIGNAL PROCESSING IN THE FREQUENCY DOMAIN: THE PHASE VOCODER
Other spectral filters
Noise reduction whit the FFT
The vectral~ object
Feedback delay
Phase manipulation
Fft vocoder

13.4 TIME STRETCHING AND PITCH SHIFTING WITH PHASE VOCODER

13.5 CONVOLUTION AND CROSS-SYNTHESIS

13.6 CONVOLUTION REVERB

ACTIVITIES
- **Analyzing algorithms**
- **Completing algorithms**
- **Substituting parts of algorithms**
- **Correcting algorithms**

TESTING
- **Integrated cross-functional project: reverse engineering**

SUPPORTING MATERIALS
- **List of Max objects - List of messages, arguments, and attributes for specific Max objects - List of GEN operators**

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Interlude G
JITTER FOR AUDIO

IG.1 INTRODUCTION TO JITTER
IG.2 NUMERICAL OPERATIONS WITH MATRICES
IG.3 DISPLAYING AUDIO SIGNALS IN JITTER
IG.4 PROCESSING AUDIO SIGNALS USING MATRICES
IG.5 THE JIT.EXPR OBJECT
IG.6 THE JIT.BFG OBJECT
IG.7 JIT.GEN
IG.8 THE FOURIER TRANSFORM AND THE JIT.FFT OBJECT

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

LEARNING AGENDA

PREREQUISITES FOR THIS CHAPTER
- Contents of volumes I and II, Interlude F, Chapters 10, 11, 12 and 13 (theory and practice)

OBJECTIVES
Skills
- To be able to program and use basic algorithms for managing and processing images and videos in Jitter
- To be able to program and use algorithms for sound processing using matrices in Jitter
- To be able to program and use granulation, phase vocoder and wave terrain synthesis algorithms in Jitter
- To be able to program and use algorithms for cross-synthesis between two sampled sounds using offline convolution implemented in Jitter

Competence
- To be able to create a brief study based on the use of Jitter

ACTIVITIES
- Constructing and modifying algorithms in Jitter

SUPPORTING MATERIALS
- List of Jitter objects – List of attributes and messages for specific Jitter objects - Glossary

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

IG.1 INTRODUCTION TO JITTER

Jitter is a Max extension designed (among other things) to process matrices of data:[1] particularly images and videos..
In this Interlude, after a first introduction to image and video management in Jitter – which will allow you to get acquainted with the environment and its characteristics – we will be dealing primarily with the use of Jitter to create matrices and data sets that we can utilize to control, display, process and generate audio signals.
To start with, we need to understand what is meant by the term matrix. If you look up the definition in a computer science textbook, you will most likely find that a matrix is a two-dimensional array. As we know from the first volume, an array[2] is an ordered set of elements of the same type (for example, integer values, floating-point values, etc.) that can be identified by an index number.
We have dealt with arrays on numerous occasions in the previous chapters. The **buffer~**, **table**, and **multislider** objects, for instance, are all containers of ordered sets of elements of the same type that can be identified with an index number: in other words, they are array containers.
A two-dimensional matrix, on the other hand, is an ordered set of elements of the same type that can be represented as a table divided into rows and columns. In this type of matrix, each element is identified by two index numbers: one for the row in which the element is located, and one for the column. In Jitter, matrices can have from 1 to 32 dimensions, and as you might expect, each element of a matrix with n dimensions is identified by *n* index numbers.
We have already dealt with two-dimensional matrices as well: when we used the object called – precisely – **matrix~**.

[1] For the sake of completeness, we should point out that Jitter can work not only with matrices but also with textures – i.e., data sets that are processed, using appropriate algorithms, by the computer's graphics processor, instead of by the CPU. In this chapter, we will be dealing exclusively with matrices.
[2] An array can also be called a "vector." We will be using this term – but with a different meaning – in section IG.7.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Let's look back at the patch 11_18_DX4.maxpat (section 11.2P). Figure IG.1 shows part of its contents.

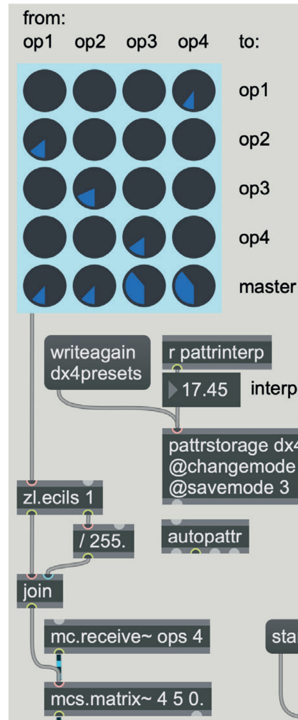


Fig. IG.1 The two-dimensional matrix of the patch 11_18_DX4.maxpat

Here we have a two-dimensional matrix with 4 columns and 5 rows, hence containing 20 (4x5) elements; and by using the dials of the `matrixctrl` interface object, we can set the value of each element. These values, as we know, represent the rescaling factor of the connections between the input and output signals of the `mcs.matrix~` object (the multichannel version of `matrix~`), shown at the bottom of the figure.
Another example of a two-dimensional matrix that we have already discussed is the wave terrain (see section 11.5).

Finally, let's see how Jitter matrices work: open the file **IG_01_jitter_matrix.maxpat** (figure IG.2).
The image contains five mini-patches that show some of the basic features of Jitter. Notice that there are several objects whose names begin with the prefix "jit": this prefix identifies Jitter objects (just as the prefix "mc" identifies multichannel objects and the tilde identifies MSP objects).
Before analyzing the five mini-patches, it is important to point out that in a Jitter matrix – as opposed to the matrices mentioned above – each element (or cell), can contain multiple values. Thus we can have, for example, a two-dimensional matrix with 2x2 cells, each containing 3 values, for a total of 2 * 2 * 3 = 12 values. The number of values per cell must be the same for the entire matrix.

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

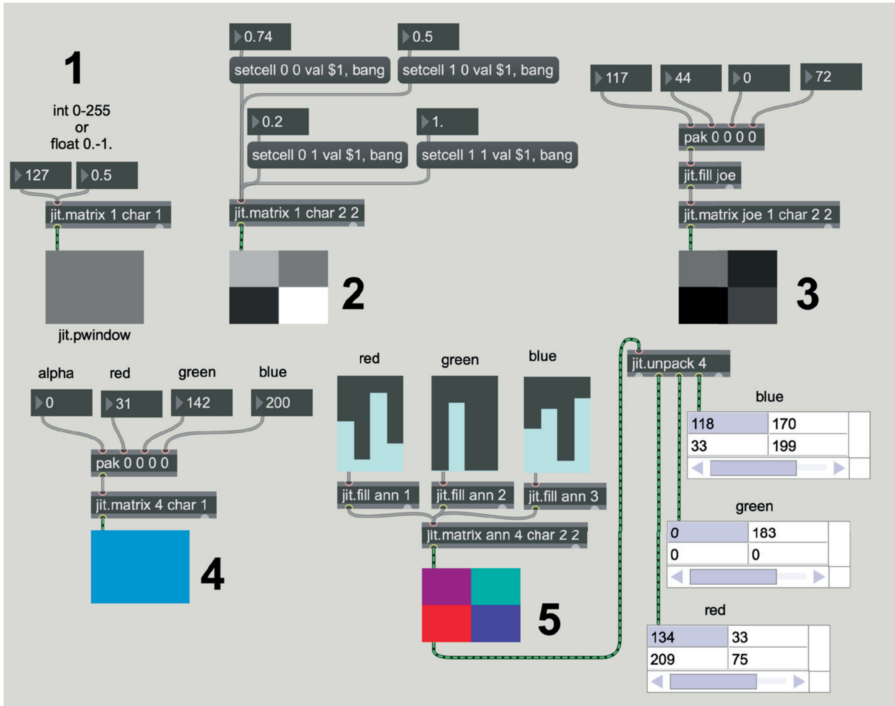


Fig. IG.2 The file **IG_01_jitter_matrix.maxpat**

The **`jit.matrix`** object creates a matrix – that is, a memory space containing data – and is therefore the most fundamental element of the library.
In the first mini-patch, this object has three arguments. The first indicates the number of values for each cell of the matrix (these values are also referred to as the matrix planes); thus our matrix has only one value per cell. The second argument determines the type of value; the type "char" indicates an integer number between 0 and 255 (i.e., a number that can be represented in 8 bits; see section 5.2T of the second volume). After that, there are as many arguments as there are dimensions in the matrix; and each one specifies the size of its corresponding dimension. In this case, there is only one argument, and it has a value of 1: we have, therefore, a one-dimensional matrix with only one cell. This is the smallest possible matrix, consisting of a single element.
The graphical object below the `jit.matrix` is named **`jit.pwindow`**; we have already seen it in several patches in section 11.5P. This object allows us to display the contents of a matrix in graphical form: to be precise, when, as in this case, the matrix contains only one plane (that is, only one value per cell), the values are displayed as shades of gray.
Now let's talk about the values. As we just said, the "char" type corresponds to an 8-bit integer between 0 and 255. If we change the value of the integer number box in the upper left part of the patch, we can see that the color of `jit.pwindow` changes gradually from black (value 0) to white (value 255).
A Jitter convention specifies that char values can also be represented as floating-point numbers between 0 and 1: try changing the value of the

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

floating-point number box connected to the `jit.matrix` and verify that in this case as well the color of `jit.pwindow` passes through the different shades of gray.
Notice that the connection between `jit.matrix` and `jit.pwindow` is made by a green cable with horizontal black stripes: this new pattern tells us that the data transmitted consists of neither simple numerical values nor signals; what is transmitted, in fact, is the name of the matrix created by `jit.matrix`. The receiving Jitter object is able to use the name of the matrix to access its contents. If you want to see the name of the matrix, connect a `print` object to `jit.matrix`'s outlet and then send a new numerical value. The Max Console will display "jit_matrix" and a multi-digit number preceded by a "u": this is the format of the names automatically assigned to matrices as they are created; we can also give matrices more meaningful names, as we will see in a moment.

In the second mini-patch, we have a two-dimensional matrix of 2x2 cells; once again, the matrix has only one plane (or value per cell) and the type is char. This time, if we want to modify the values, we must specify the target cell: we therefore use the message "setcell" followed by the cell index (in other words, the values of the coordinates, starting from 0), and the argument "val" followed by the value we want to set. After setting the value, it is necessary to send a bang to the `jit.matrix` object so that it will communicate the matrix name to the connected objects: this is why the message box, after the message "setcell" with its arguments, contains the message "bang."
If you look at the coordinate values in the four message boxes, you will see they are [0, 0] [1, 0] [0, 1] [1, 1], in that order. The first coordinate is the column number (i.e., the x coordinate), and the second coordinate is the row number (i.e., the y coordinate). Change the values of the four number boxes and observe how the color of the corresponding cell changes accordingly. Notice that the [0, 0] coordinate corresponds to the upper left corner.

We can also send a list of values to the matrix using the object **`jit.fill`** (mini-patch number 3). This object needs to know the name of the matrix to fill. We can name a matrix by entering the name as the first argument of `jit.matrix`: in the third mini-patch, you can see that the matrix is named "joe." This name allows other objects to work on the same matrix, just as the name of the `buffer~` object allows its contents to be shared. It is also possible (exactly as with `buffer~`) for there to be several `jit.matrix` objects with the same name, all referring to the same matrix.

(...)

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

other sections in this chapter:

Images and color manipulation
Crossfading between matrices
Feedback and slide
Dimension interchange, value inversion, rotation
Submatrices, oversampling and interpolation

IG.2 NUMERICAL OPERATIONS WITH MATRICES
Creating a step sequencer

IG.3 DISPLAYING AUDIO SIGNALS IN JITTER

IG.4 PROCESSING AUDIO SIGNALS USING MATRICES

IG.5 THE JIT.EXPR OBJECT

IG.6 THE JIT.BFG OBJECT

IG.7 JIT.GEN

IG.8 THE FOURIER TRANSFORM AND THE JIT.FFT OBJECT

ACTIVITIES
- **Analyzing algorithms**
- **Completing algorithms**
- **Substituting parts of algorithms**
- **Correcting algorithms**

TESTING
- **Integrated cross-functional project: reverse engineering**

SUPPORTING MATERIALS
- **List of Jitter objects - List of messages, arguments, and attributes for specific Jitter objects - Glossary**

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved

Alessandro Cipriani • Maurizio Giri

Electronic Music and Sound Design

Theory and Practice with Max 8 • volume 3

Topics

Reverberation and creative uses of reverb - Spatialization with two or more channels – AM, RM, SSB, FM, and PM - Nonlinear distortion - Wave terrain synthesis - Split synthesis - Granular and particle synthesis - Granulation and segmentation of sampled sounds - Vocoder - Analysis and resynthesis - Cross-synthesis - Convolution - Jitter for audio - Gen programming

"There is no shortage of books in the world that seek to demonstrate the erudition of their authors. It is harder, however, to find books that focus on the readers – taking them on a journey that will ultimately change them. The books by Cipriani and Giri belong to this rare category: they are books that *explain*. (…) The third volume of *Electronic Music and Sound Design* is a kaleidoscopic catalog of ideas and applications for analyzing, synthesizing, and transforming signals in a wide variety of ways. (…) Cipriani and Giri succeed in addressing everyone without weakening the theoretical basis and without unnecessary specializations – achieving a masterful balance of comprehensibility, functionality, and breadth." (From the foreword by **Carmine-Emanuele Cella**, Assistant Professor in Music and Technology, CNMAT - University of California, Berkeley).

This is the third volume of an organic educational system that includes an extensive online component consisting of hundreds of interactive sound examples, videos, theory and practice glossaries, tests, programs written in Max, a Max object library created specifically for these volumes, and many practical activities (often with Gen and Jitter).

ALESSANDRO CIPRIANI is the co-author, with R. Bianchini, of *Virtual Sound*, a textbook on *Csound* programming. His compositions have been published by the *Computer Music Journal*, the *International Computer Music Conference*, CNI, etc. He has composed music for the Peking Opera Theater and for films and documentaries in which computer-processed ambient sounds, dialogues and music blend together interchangeably – notably, with the Edison Studio composers' collective, for the silent movies *Battleship Potemkin*, *Inferno* and *Das Cabinet des Dr. Caligari*, published on DVD by Cineteca di Bologna. He has given seminars at many universities (the University of California, the Sibelius Academy in Helsinki, the Moscow Conservatory, DMU-Leicester, etc.). He is a tenured professor of Electroacoustic Music Composition at the Conservatory of Frosinone and a member of the Editorial Board of the journal Organised Sound (Cambridge University Press).

MAURIZIO GIRI is a professor of composition who teaches Max programming techniques at the Conservatory of Frosinone. He has written both instrumental and electroacoustic music. He is currently working on electronic music, and on the application of new technologies to digital sound processing, improvisation and musical composition. He has written software for electroacoustic improvisation and for live electronics. He is the founder of *Amazing Noises*, a software house that develops musical applications and plug-ins for mobile devices and computers, and he also collaborates with *Ableton*, for whom he has published numerous Max for Live devices. He has published Max tutorials in various professional journals. He has been a resident artist in Paris (*Cité Internationale des Arts*) and in Lyon (GRAME). He has collaborated with the Institut Nicod of the *École Normale Supérieure de Paris*, for a project on the philosophy of sound.

ISBN 978-88-99212-24-7

www.contemponet.com

demo of Electronic Music and Sound Design volume 3 by A. Cipriani and M. Giri - © Contemponet 2023 - All rights reserved