Open Access 2026-10-22

Interactive Music, Live Electronics

Andrea Szigetvári, DLA and Ádám Siska

Open Access 2026-10-22

© Andrea Szigetvári, DLA and Ádám Siska

Creative Commons NonCommercial-NoDerivs 3.0 (CC BY-NC-ND 3.0)
This work can be reproduced, circulated, published and performed for non-commercial purposes without restriction by indicating the author's name, but it cannot be modified.

Reader: Stewart Collinson

ISBN 978 963 279 216 3

Prepared under the editorship of Typotex Kiadó (http://www.typotex.hu)
Responsible manager: Zsuzsa Votisky

Open Access 2026-10-22

Contents

0 Foreword **1**

1 Introduction to Live Electronics **3**

2 Fixed-Media and Interactive Systems **5**
2.1 Studio Tools 5
2.1.1 Sequencers 5
2.1.2 DAWs and Samplers 6
2.1.3 Integrated Systems 6
2.1.4 Composing in the Studio 6
2.2 Interactive Systems 7
2.2.1 Roles of Interactive Systems 7
2.2.2 Composition and Performance Issues 8

3 Description of Live Electronic Pieces **10**
3.1 Karlheinz Stockhausen: Mantra (1970) 10
3.1.1 Gestures Controlling the System and Their Interpretation 11
3.1.2 Sound Processing Unit 11
3.1.3 Mapping 12
3.1.4 The Role of Electronics 12
3.2 Andrea Szigetvári: CT (2010) 13
3.2.1 Gestures Controlling the System 13
3.2.2 Gesture Capture and Analysis Strategies 14
3.2.3 Sound Synthesis and Processing Unit 16
3.2.4 Mapping 16

4 Listing and Description of Different Types of Sensors **19**
4.1 Defining Controllers 19
4.2 Augmented Instruments 20
4.3 Instrument-like Controllers 21
4.4 Instrument-inspired Controllers 23
4.5 Alternate Controllers 24

5 Structure and Functioning of Integra Live **27**
5.1 History 27
5.2 The Concept of Integra Live 28
5.2.1 Structure of the Arrange View 29

Open Access 2026-10-22

5.2.2 Structure of the Live View . . . 32
5.3 Examples . . . 33
5.4 Exercises . . . 36

6 Structure and Functioning of Max 37
6.1 History . . . 37
6.2 The Concept of Max . . . 38
6.2.1 Data Types . . . 39
6.2.2 Structuring Patches . . . 40
6.2.3 Common Interface Elements and Object Categories . . . 40
6.2.4 Mapping and Signal Routing . . . 42
6.3 Examples . . . 43
6.3.1 Basic Operations . . . 43
6.3.2 Embedding . . . 44
6.3.3 Signal Routing . . . 45
6.3.4 Data Mapping . . . 46
6.3.5 Templates . . . 47
6.4 Exercises . . . 49

7 OSC — An Advanced Protocol in Real-Time Communication 50
7.1 Theoretical Background . . . 50
7.1.1 The OSC Protocol . . . 50
7.1.2 IP-based Networks . . . 51
7.2 Examples . . . 52
7.3 Exercises . . . 53

8 Pitch, Spectrum and Onset Detection 55
8.1 Theoretical Background . . . 55
8.1.1 Signal and Control Data . . . 55
8.1.2 Detecting Low-Level Parameters . . . 56
8.1.3 Detecting High-Level Parameters . . . 56
8.2 Examples . . . 57
8.2.1 Analysis in Integra Live . . . 57
8.2.2 Analysis in Max . . . 58
8.3 Exercises . . . 61

9 Capturing Sounds, Soundfile Playback and Live Sampling. Looping, Transposition and Time Stretch 62
9.1 Theoretical Background . . . 62
9.1.1 Methods of Sampling and Playback . . . 62
9.1.2 Transposition and Time Stretch . . . 63
9.2 Examples . . . 64
9.2.1 Sampling in Integra Live . . . 64
9.2.2 Sampling in Max . . . 65
9.3 Exercises . . . 67

Open Access 2026-10-22

10 Traditional Modulation Techniques 70
10.1 Theoretical Background . . . 70
10.1.1 Amplitude and Ring Modulation . . . 70
10.1.2 Frequency Modulation . . . 71
10.1.3 The Timbre of Modulated Sounds . . . 72
10.2 Examples . . . 73
10.2.1 Ring Modulation in Integra Live . . . 73
10.2.2 Modulation in Max . . . 74
10.3 Exercises . . . 75

11 Granular Synthesis 78
11.1 Theoretical Background . . . 78
11.2 Examples . . . 79
11.2.1 Granular Synthesis in Integra Live . . . 79
11.2.2 Granular Synthesis in Max . . . 80
11.3 Exercises . . . 82

12 Sound Processing by Filtering and Distortion 84
12.1 Theoretical Background . . . 84
12.1.1 Filtering . . . 84
12.1.2 Distortion . . . 85
12.2 Examples . . . 87
12.2.1 Filtering and Distorting in Integra Live . . . 87
12.2.2 Filtering and Distorting in Max . . . 87
12.3 Exercises . . . 92

13 Sound Processing by Delay, Reverb and Material Simulation 95
13.1 Theoretical Background . . . 95
13.1.1 Delay . . . 95
13.1.2 Reverberation . . . 95
13.1.3 Material Simulation . . . 96
13.2 Examples . . . 97
13.2.1 Material Simulation in Integra Live . . . 97
13.2.2 Delay and Reverb in Max . . . 97
13.3 Exercises . . . 98

14 Panning, Surround and Multichannel Sound Projection 100
14.1 Theoretical Background . . . 100
14.1.1 Simple Stereo and Quadro Panning . . . 100
14.1.2 Simple Surround Systems . . . 101
14.1.3 Multichannel Projection . . . 101
14.1.4 Musical Importance . . . 101
14.2 Examples . . . 102
14.2.1 Spatialisation in Integra Live . . . 102
14.2.2 Spatialisation in Max . . . 103
14.3 Exercises . . . 105

Open Access 2026-10-22

15 Mapping Performance Gestures **106**
15.1 Theoretical Background 106
15.1.1 Mapping 106
15.1.2 Scaling 107
15.2 Examples 107
15.2.1 Mappings in Integra Live 107
15.2.2 Mappings in Max 109
15.3 Exercises 111

Bibliography **115**

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

Chapter 0

Foreword

This syllabus approaches the realm of real-time electronic music by presenting some of its most important concepts and their realisations on two different software platforms: *Integra Live* and *Max*.
Both applications are built on the concept of *patching*, or *graphical programming*. Patches contain small program modules — called *unit generators* — which can be connected together. Unit generators are 'black boxes' whose internal logic is hidden from the musicians, performing some well-defined, characteristic action. The behaviour of the patch as a whole depends on the choice of unit generators and the connections that the musician creates between them.
The visual design of Integra Live resembles a more traditional multitrack sequencer, with multiple tracks and a timeline. The main novelty of the software is that, instead of containing pre-recorded samples, the tracks contain patches. In other words, rather than holding actual *sounds*, each track describes *processes* to make sounds, extending the philosophy of the traditional sequencer towards live, interactive music performance.
Max, on the other hand, offers no more than a blank canvas to its user. Although this approach requires a more computer-oriented way of thinking, it does not put any borders in the way of the musicians' imagination.
Chapters 1–4 present the basic aspects of live electronic music, pointing out the typical differences (in terms of aesthetics, possibilities and technical solutions) between tape music and live performance. To illustrate these, analyses of two pieces involving live electronics is presented. A detailed description of sensors used by performers of live electronic music is also included.
Chapters 5–6 introduce the two software environements used in this syllabus, Integra Live and Max, respectively.
Chapter 7 describes protocols that allow different pieces of software and hardware (including sensors) to communicate with each other in a standardised way.
Chapter 8 defines and presents different methods to capture low- and high-level musical descriptors in real-time. These are essential in order to let the computer react to live music in an interactive way.
Chapters 9–11 describe different methods of creating sounds. Some of these are based on synthesis (the creation of a signal based on a priori principles), others on capturing existing sounds, while others mix these two approaches.

Open Access 2026-10-22

Chapters 12–13 contain techniques of altering our sound samples by different sound processing methods.
Chapter 14 is centred on the issues of sound projection.
Chapter 15 introduces the quite complex problematics of mapping, focusing on the questions of transforming raw sensor data into musically meaningful information.
Every Chapter starts with an introduction, describing the purpose, basic behaviour and possible parameters (and their meanings) of the methods presented. This is followed by examples realised in Integra Live and Max (some Chapters do not contain Integra Live examples). The purpose of these examples is demonstration, emphasizing simple control and transparency; indeed, they cannot (and are not intended to) compete with commercially available professional tools. Moreover, in some cases, they deliberately allow some settings and possibilities that professional tools normally forbid by design. Our goal in this was a better illustration of the nature of some techniques through the exploration of their extreme parameters and settings.
The syllabus contains a number of excercises, to facilitate understanding through practice of the principles and methods explored.
To run the example projects for Integra Live, Integra Live needs to be downloaded and installed (available freely from `http://www.integralive.org` under the GPL license). Software written in Max were converted into standalone applications — there is no need to have a copy of Max in order to open these files.

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

Chapter 1

Introduction to Live Electronics

The first electroacoustic works — not considering pieces using electro-mechanic instruments, like the ondes Martenot — were realized for fixed media (in most cases, magnetic tape). Performing such music consists of playing back the recordings, the only human interaction being the act of starting the sound flow.

The reason behind this practice was purely practical: the realization of these works required the use of devices, which at the time were available only in a few 'experimental' studios. In addition, the time scale of preparing the sound materials was many orders of magnitude above the time scale of the actual pieces. This made it impossible to create these works in real-time. As an example, the realization of Stockhausen's *Kontakte*, a key piece in the history of early electroacoustic music, took almost two years between 1958 and 1960.

Preparing music in a studio has its own benefits. Besides avoiding the inevitable errors of a live performance, the composer is able to control the details of the music to a degree that is not achievable in a real-time situation: in a studio, it is possible to listen each sound as many times as needed, while experimenting with the different settings of the sound generator and manipulating devices. The composer can compare the results with earlier ones after each modification and make tiny adjustments to the settings when needed. These are perhaps the key reasons why many composers, even today, choose to create pieces for fixed media. However, tape music has raised a number of issues since its birth.

Within a decade of the opening of the first electronic studios, many composers realized the limitations inherent in studio-made music. For most audiences, the presence of a performer on stage is an organic element of the concert situation. Concerts presenting pre-rendered music feature only a set of loudspeakers distributed around the venue. While the soundscape created by such performances may indeed be astonishing, the lack of any human presence on stage challenges the listener's ability to sustain focused attention.

Another issue arises from the questions of interpretation and improvisation, which are, by their nature, 'incompatible' with the concept of fixed media. Moreover, not only is tape-improvisation impossible, but subtle in situ adjustments, which are so natural in traditional music practice — for example, the automatic adaptation of accompanying musicians to a soloist or the tiny adjustment of tuning and metre to the acoustics of the performance hall – are excluded as well. This problem manifests itself when mixing tape music with live instruments, and becomes much more apparent when the music has to co-exist with other art forms, like dance. This latter was perhaps the main reason that pushed composers like John Cage towards methods which could be realized in 'real-time', leading to such early productions as

Open Access 2026-10-22

Variations V (1965), composed for the Cunningham Dance Company.
As a consequence of the above (and many other) arguments, the demand for the possibility of performing electronic music under 'real-time' circumstances increased over time, eventually giving birth to the world of 'live' electronic music.
Although the first works of this genre were already created in the mid-sixties, it was not until the millenium that the idea of live performance in electronic music became widespread[1]. Moreover, most works involving live electronics were themselves experimental pieces, where improvisation played a great role. The lack of proper recordings as well as documentation (both musical and technical) makes the researchers' task very hard; a detailed summary of the history of the first decades of live electronic music is still awaited[2].
What changed the situation during the past (two) decade(s)?
The obvious answer is the speed and capacity of modern computers, which have played an undisputable role in the evolution of live-performed electronic music. Unlike the studio equipment of the eighties, the personal computer offers an affordable solution for many musicians to use as (or create) musical devices which fit their aesthetic needs.
However, this is not the only reason. Computers just do what their users tell them to do: without proper software solutions and knowledge, even super-fast computers are not much help. The development of software paradigms with live performance in mind — either based on pre-existing studio paradigms, or on completely new ideas — have also made a great impact on the genre. These new paradigms allow musicians to discuss their ideas, creating communities with an increased amount of 'shared knowledge'. This knowledge boosts interest in live electronic music (both from the prespective of musicians and researchers) and serves as inspiration for many artists engaging with the genre.

[1] As a landmark: one of the most important conferences on the subject, *New Interfaces for Musical Expression* (NIME) was first held in 2001.

[2] And, as the 'protagonists' of that era — the only sources we could rely on — pass away, such summary might never materialise.

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

Chapter 2

Fixed-Media and Interactive Systems

Since the advent of fast personal computers, the performance practice of electroacoustic music has been in continuous transformation. The emergent drive desire of live interpretation possibilities is forcing tape music to yield it's primacy to new digital interactive systems — also called *Digital Music Instruments* (*DMI*). The field is still in a pioneering, transitional state, as it is difficult to compete with the quality of both past and present acousmatic and computer music, created using non-real-time techniques. A key reason behind this is that the compositional process itself is inseparable from its realisation, allowing the careful creation of the fixed, final sound material of the piece. Therefore, developers of DMIs have to solve (at least) two essential problems:

- Invent strategies that substitute (recreate) the non real-time studio situation, where sonorities are the result of numerous sequential (consecutive, successive) procedures.
- Develop appropriate ways of gestural control to shape the complex variables of the musical process.

2.1 Studio Tools

To better understand the main issues of interactivity, we first present the most important devices involved in studio-based (offline) composition. These solutions have been constantly evolving during the past sixty years, and development is still active in the field, to say the least. Nonetheless, the past decades have already given us enough experience to categorise the most important instruments of non-real-time music editing.

2.1.1 Sequencers

A *sequencer* allows recording, editing and play-back of musical *data*. It is important to emphasize that sequencers deal with data instead of the actual sound, thus, they represent an abstract, hierarchical conceptualisation of musical information. In contrast to sound editors, a sequencer will not offer the possibility of editing the signal. Instead, the information is represented in a symbolic way. This could be a musical score, a piano roll, or any other symbolic representation. However, in contrast to traditional scores, a sequencer may display (as well as capture and/or play back) information that is not available in a 'traditional' score (for example exact amplitude envelopes, generated by MIDI controllers). As it is clear from

Open Access 2026-10-22

the above, a sequencer usually contains a minor score engraver as well as the capabilities for creating and editing complete envelopes.

2.1.2 DAWs and Samplers

The *DAW* (Digital Audio Workstation) is similar to the sequencer in that it has storage, editing and play-back options. However, in contrast to a sequencer, a DAW always uses (pre-)recorded sound as a source. Thus, it is not the abstract, parametric representation of the sound that is edited, but the sonic material itself. To achieve this, a DAW usually includes a large number of plug-ins — each one capable of modifying the original sound in many different ways.
Although it was originally a separate tool, the *sampler* is, by now, an integral part of any DAW. A sampler stores (or, in many cases, also records) samples of original sound sources, often in high quality. Then, fed by control data arriving from (MIDI) controllers (or a sequencer), it chooses the actual sound sample to be played back, adjusting its timing, pitch, dynamics and other parameters according to the actual control values.

2.1.3 Integrated Systems

An *integrated system* is nothing more than a tool containing all listed devices. It contains a complete built-in sequencer, a DAW and a sampler. These integrated systems are usually multi-track, each track having its own *inserts* (plug-ins modifying the sounds of the individual tracks), notation tools that ease the creation/editing of control data in addition to actual sounds as well as extensive sample and rhythm libraries offering a broad choice of options for additional sounds. Some more advanced integrated systems offer video syncing capabilities too, allowing the creation of complex audiovisual content.

2.1.4 Composing in the Studio

As outlined in Chapter 1, the non-real-time composing method consists of a careful iterative process of listening, modifying and sequencing samples.
Each segment of the sound can be listened to many times, allowing the composer to concentrate on minute details of the music. According to the aesthetic needs of the piece, one can apply any combination of sound processing tools to alter the original samples. By listening to the result after each step and comparing it to the samples already created as well as the 'ideal' goal present in the imagination of the composer, one can reach any desired sound. After classifying these, one can 'build' the piece by structuring these samples in an appropriate way.
Of course, the steps outlined above are interchangeable to some extent. In many cases, composers create the temporal structure of the samples in parallel with the creation of the samples themselves. It is also possible to take abstract decisions regarding the structure in advance (based, for example, on different aesthetic or mathematical principles), long before producing the first actual sample of music.
In other words, the 'studio situation' — i.e. the opportunity of listening to and altering the samples again and again within isolated, noiseless working conditions — allows composers to create very detailed and elaborated soundscapes.

Open Access 2026-10-22

2.2 Interactive Systems

The offline approach, however efficient in terms of sound quality and controllability of the music, does not allow interaction with the computer, which (should be) a key element of any live performance. Hence, there has been an increasing demand for 'raising' our electronic tools to a higher musical level since the earliest days of electroacoustic music, implementing solutions that allow our devices to react to their musical environment in a (half-)automated way. Such tools are called *interactive systems*.
These systems are still highly individual: there are no general tools providing ready-made environments or instruments for composition. The composer/developer has to conceive their instrument from scratch, defining the musical potential of the piece to a greater extent. An established theory is also lacking: the available bibliography consists of continuously evolving reports on the actual state of developments. However, these only document some aspects of the field — mainly the tools which gradually become standardized, and can be used as building blocks for consecutive works. Reviewing the documented works, the variety of solutions is striking.

2.2.1 Roles of Interactive Systems

Authors writing comprehensive summaries conceptualize the workflow of interactive computer music systems in different ways. Table 2.1 outlines the main aspects of three different approaches.

Table 2.1: The phases of interactive processing proposed by three different authors. One can observe that, as time progresses, the workflow becomes more exact and pragmatic (thanks to cumulative experience).

	Rowe (1993)	Winkler (2001)	Miranda (2006)
1.	Sensing	Human input, instruments	Gestures to control the system
2.	Processing	Computer listening	Gesture capture strategies
3.	Response	Interpretation	Sound synthesis algorithms
4.		Computer composition	Mapping
5.		Sound generation	Feedback modalities

The stages of the classification proposed by Miranda and the relations between these is depicted in Figure 2.1. The main elements are:

Controlling Gestures. This means both the physical and musical gestures of the performer. When creating an interactive system, the composer/designer needs to define both the set of meaningful motions and the sonic events that may be comprehended by the machine.

Gesture Capture and Analysis. The physical gestures are captured by sensors, while sound is normally recorded by microphones. However, a gesture capture/analysis strategy is far more than just a collection of sensors and microphones: it is the internal logic which converts the raw controller and signal data into parameters that are meaningful in the context of the piece. Sensors are presented in Chapter 4 while some signal analysis strategies are explained in Chapter 8.

Open Access 2026-10-22

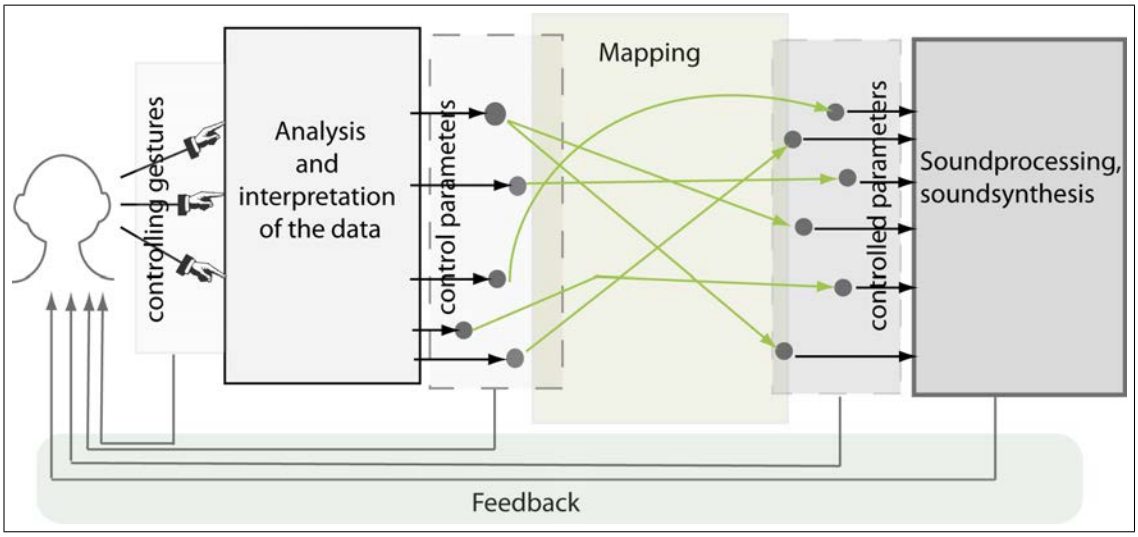


Figure 2.1: On the one hand, the number and type of data structures that control sound processing and synthesis are defined by the gesture capture strategies, the analysis, and the interpretation of control data. On the other hand, the number of (changeable) parameters of the sound-producing algorithms is fixed by the algorithms themselves. Mapping creates the balance between these by specifying which control data belongs to which synthesis parameter, as well as the registers to be controlled. The choice of feedback modalities influences the way how the performer is informed about the results of their gestures.

Sound Synthesis and Processing. This is the 'core' of an interactive system, which produces the sound. The parameters controlling the sound creation — although scaled or modified — are derived from the control values arriving from the gesture analysis subsystem.

Mapping. Normally we use a set of controllers whose values are routed in different ways to the parameters of sound synthesis. Mapping (presented in Chapter 15) denotes the way the parameters arriving from the controllers are converted into values that are meaningful for the sound generation engine.

Feedback. To have real, meaningful interaction, it is essential to give different levels of feedback to the performer simultaneously. The most obvious, fundamental form of feedback is the sound itself; however, as in the case of 'classic' chamber music (where performers have the ability to, for example, look at each other), other modes are also very important to facilitate the 'coherence' between the performer and the machine/instrument. Examples include haptic (touch-based) feedback (normally produced by the sensors themselves) or visual representation of the current internal state of the system.

2.2.2 Composition and Performance Issues

Compared with the tools accessible in a studio, an interactive 'performing machine' has the undisputed ability of performing live, giving musicians the ability to interpret the same piece in many different ways, as with any instrumental work. In this respect, interactive systems

Open Access 2026-10-22

are much better suited for combining with 'ordinary' instruments than tape-based music. Improvisation, a realm completely inaccessible to fixed media, is made possible by these devices.
Conversely, it is much harder to achieve the same sound quality and richness with these devices compared to standard studio tools. While the offline/studio method is built on the slow process of listening to, manipulation of, editing, layering and sequencing of *actual* samples, an interactive system is a generic device where, during the design process, there is no way to test *every* possible sound generated by the instrument. Simply, the challenge for the musician wanting to work in a live, interactive, improvisational way is to begin to understand what the possibilities of real-time sound synthesis/manipulation are, given (con)temporary computational limitations.

Open Access 2026-10-22

Chapter 3

Description of Live Electronic Pieces

In what follows, we examine two compositions, based on the categorisation proposed by Miranda (see Chapter 2). One of these pieces (Mantra, by Karlheinz Stockhausen) was created during the very early days, while the other (CT, by Andrea Szigetvári) is a very recent piece within the genre of interactive electro(acoustic) music. Our choice of pieces illustrates how the possibilities (both musical and technical) have evolved during the past forty years.

3.1 Karlheinz Stockhausen: Mantra (1970)

Mantra, a 70 minute long piece for two pianos, antique cymbals, wood blocks and live electronics was composed by *Karlheinz Stockhausen* (1928–2007) as his Werk Nr. 32 during the Summer of 1970 for the *Donaueschinger Musiktage* of the same year. It has a distinguished place within the composer's œuvre for multiple reasons. On the one hand, it is one of his first pieces using *formula technique* — a composition method derived by Stockhausen himself from the serial approach —, a principle which served as his main compositional method during the next 30 years, giving birth to compositions like *Inori*, *Sirius* or the 29 hour long opera *Licht*. On the other hand, it was also one of his first determinate works (i.e. where the score is completely fixed) to include live electronics, and generally, his first determinate work after a long period of indeterminate compositions.

The setup requires two pianists and a sound technician. Each pianist performs with a grand piano, 12 antique cymbals, a wood block, and a special analog modulator (called 'MODUL 69 B'), designed by the composer. Additionally, Performer I also operates a short wave radio receiver. The sound of the pianos, picked up by microphones, is routed through the modulator as well as being sent directly to the mixing desk. The combination of unprocessed and modulated piano sounds is amplified by four loudspeakers set at the back of the stage. The role of the technician is the active supervision of this mix, according to general rules set by the score.

The most comprehensible effect of MODUL 69 B is the built-in ring modulation, which is applied to the piano sound most of the time. The details of ring modulation are explained in Chapter 10. Here, we just refer to the results.

Ring modulation consists of the modulation of the amplitude of an incoming signal (called *carrier*) with a sine wave (called *modulator*). When the modulator frequency is very low (below approx. 15 Hz), the modulation adds an artificial tremolo to the timbre. With higher

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

frequencies, two sidebands are generated per each spectral component. Depending on the ratio of the carrier and frequency, the resulting timbre may either become harmonic or inharmonic.
In the subsequent subsections, we examine the role of the ring modulation within the piece.

3.1.1 Gestures Controlling the System and Their Interpretation

When Stockhausen composed the piece, the choice of live controllers was very limited: the main controller of the MODUL 69 B is a simple dial (with a single output value), which controls the modulator frequency of the device. This controller allows two basic gestures:

- Rotating the dial (the output value changes continuously).
- Leaving the dial at its current position (the output value remains unaltered).

Of course, one may combine the two gestures above to create more complex ones, e.g. it is possible to rotate the dial back and forth within a range.
The dial has twelve pre-set positions, which are defined as musical tones. This representation is musically meaningful for the performer, yet it hides all irrelevant technical details from their sight. In most of the cases, the Pianists need to select one of these presets and hold them steady for very long durations. However, there are situations when they need to

- rotate the dial slowly from one preset to the other;
- play with the dial in an improvisatory way (usually, within a range defined by the score);
- vibrate the incoming sound by slowly (but periodically) altering the oscillator frequency;
- add tremolo to the incoming sound by setting the dial to very low frequency values.

The interface is quite intuitive: it is suited both for playing glissandi and for holding a steady note for as long as needed. It is consistent, as tiny changes in the dial position cause smooth and slight changes in the modulating frequency. Yet it is very simple as well: since the dials hold their position when left unattended, they require minimal supervision from both performers. Moreover, the music is composed in such a way that the performers do not need to carry out difficult instrumental passages at times when they need to control the dial.
The ‘playing technique’ may be learned and mastered by any musician in less than a day.

3.1.2 Sound Processing Unit

According to the score, MODUL 69 B consists of 2 microphone inputs with regulable amplifiers, compressors, filters, a sine-wave generator and a particularly refined ring modulator. Unfortunately, the exact internal setup is not documented (at least, in a publicly available form), which makes any discussion regarding the fine details of the device hopeless. Nevertheless, by analysing the score we may figure out that the modulation range of the ring modulator is between 5 Hz and 6 kHz. A remark by Stockhausen also clarifies that the compressor is needed in order to balance the ring-modulated piano sound, as (specially, for still

Open Access 2026-10-22

sounds) the signal/noise ratio of the ring-modulated sounds differ from that of the original piano timbre.
Without knowing the internals, we can not guess the number of the device's variable parameters. It is evident that the modulator frequency is such a parameter; however, we don't know whether the built-in compressor or the filter operates with fixed values or not. While it might make sense to assume that at least one of these values (the cutoff frequency of a hypothetical high-pass filter, if there was any) would change according to the actual modulator frequency, there is no way to verify such assumptions.

3.1.3 Mapping

In Mantra, the dial controls the frequency of the modulator. There are twelve pre-set positions, labeled with numbers 1 to 12, corresponding to different tones of a twelve-tone row and its inverted form. However, instead of referring to these numbers, the score indicates the actual presets to use with the pitches corresponding to the particular positions.
We have no information about the particular mapping between the physical position of the dial and the corresponding frequency. However, the score suggests that the link between the position of the dial and the actual frequency is, to say the least, monotonic. Moreover, it is likely that it follows a logarithmic scaling in frequency (linear scaling in pitch), at least in the frequency region where the rows are defined ($G_3^\flat$ to $G_4^\sharp$ for Performer I and $B_2^\flat$ to C_4 for Performer II).
The two basic motions, explained in Section 3.1.1, are mapped to the modulator frequency as follows:

- Changing the position of the dial causes glissandi.
- Keeping the dial position constant creates a steady modulation at a particular frequency.

3.1.4 The Role of Electronics

As we have said, together with the idea of representing the modulating frequencies as musical tones instead of numerical frequency values, the device is very intuitive. On the other hand, we may realise that the overall possibilities of MODUL 69 B (particularly, the expressivity allowed by the device) is very limited. We may obtain the following musical results:

- Artificial tremolo, by choosing a very low modulating frequency.
- (Timbral) vibrato, by modulating (manually) the dial around a central value.
- (Spectral) glissandi, by rotating the dial.
- Spectral alteration of the piano sound.

The timbral modifications introduced into the piano sound are of key importance in the piece, from both a conceptual and a musical point of view. The 'conceptual role' of the modulator frequencies is to mark the sections of the piece, which are defined by a main formula of thirteen tones (built on the previously mentioned twelve-tone row). Since the types of timbral alterations (e.g. harmonic, transposed, inharmonic etc.) of the piano sound are related to the

Open Access 2026-10-22

ratio of the base pitch of the sound and the modulator frequency, having a fixed modulator frequency organizes the twelve tones into a hierarchy — defined by their relation to this modulator —, creating an abstract 'tonality'. This is different from the classical tonality, though, since octaves do not play the same role as they do in classical harmony: the results of ring modulation *do* change if we add an octave to an initial interval. Nevertheless, by computing — for each possible combination appearing in the piece — the timbral changes that ring modulation introduces, we could find the 'centres of gravity' of each section within the piece, underlying the role that ring modulation plays in Mantra.

3.2 Andrea Szigetvári: CT (2010)

CT is an audiovisual piece realised purely in Max and Jitter, applying Zsolt Gyenes' 100 second computer tomograph animation as a visual source. Computer tomography (CT) is a medical imaging technology: normally, the scanned 'object' is the human body, for the purposes of diagnosis. To create an 'experimental' scan, metal wires were placed in the tomograph. The frames of slices of the metal wires were photographed and animated, creating unexpected results in the form of abstract moving images. The animation does not have any particular meaning, it works like a Rorschach test, where the expressive qualities, meanings are added by the viewing subjects. The music serves here as a tool to particularize the expressivity of the visual system. Different sonic interpretations of the same visual gestures are produced by an interactive music system, in which parameters are modified in real time by the performer.
Figure 3.1 shows how the cross-sections of the metal wire form the frames of a video animation.
Figure 3.2 displays every 5th frame of a 4.2 sec long sequence of the final video. There are shiny sparks travelling in space, inhabiting different orbits, turning around, and changing their size and shape. The number of the sparks, the speed and direction of their movements varies in time.

3.2.1 Gestures Controlling the System

The music is controlled by the animated tomograph images. These describe five orbits, which are assigned to five musical parts. Figure 3.3 displays the activity on these orbits. This 'score' helps to indentify sections of solos, duos, trios, quartets and quintets.
The realising software can play the 2 394 frames at different speeds:

- From the beginning to the end (original structure).
- Manually advancing frames in both directions.
- Using loops with controllable starting points, lengths and directions.
- Using random playback with controllable range and offset.

The parameters of the video playback are manipulated by the performer with a MIDI controller, which contains sliders and knobs.
As the sound sequences are controlled by the movements of the sparks of the CT animation, the gestures of the performer influence the music in an indirect way. Depending on how

Open Access 2026-10-22

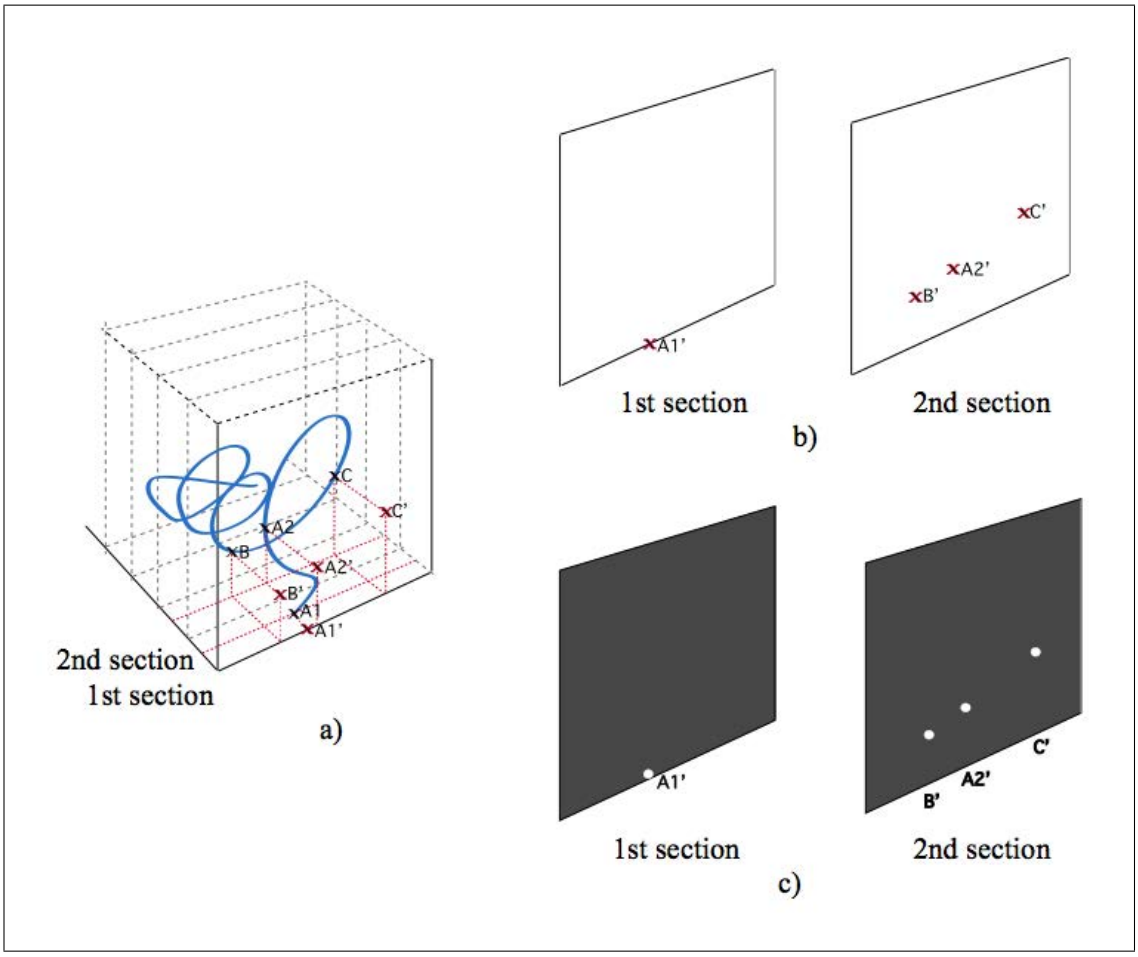


Figure 3.1: Process of the CT scan of metal wires: a) *Position of the wire in the computer tomograph. The red lines indicate 2 intersections.* b) *2 frames created by the intersection: 1st section contains one, the 2nd section contains three 'sparks'.* c) *The final results are bright shining sparks projected on a black backgound.*

the video is played back, it is possible to control those movements either directly (frame by frame) or by assigning general parameters to loop or random playback.

3.2.2 Gesture Capture and Analysis Strategies

Each frame is represented as a set of x and y coordinates, representing the positions of the (active) sparks. An algorithm — calculating the speed of the sparks between any consecutive frames — guarantees that any combination of frames would produce a speed value, depending on how far the positions of the sparks are from each other on a given orbit.

We obtain 3 parameters per each orbit after analyzing the frames and computing the speeds. Thus, the music can be conrolled by fifteen parameters all together: $x_1, y_1, v_1, x_2, y_2, v_2, x_3, y_3, v_3, x_4, y_4, v_4, x_5, y_5, v_5$.

Open Access 2026-10-22

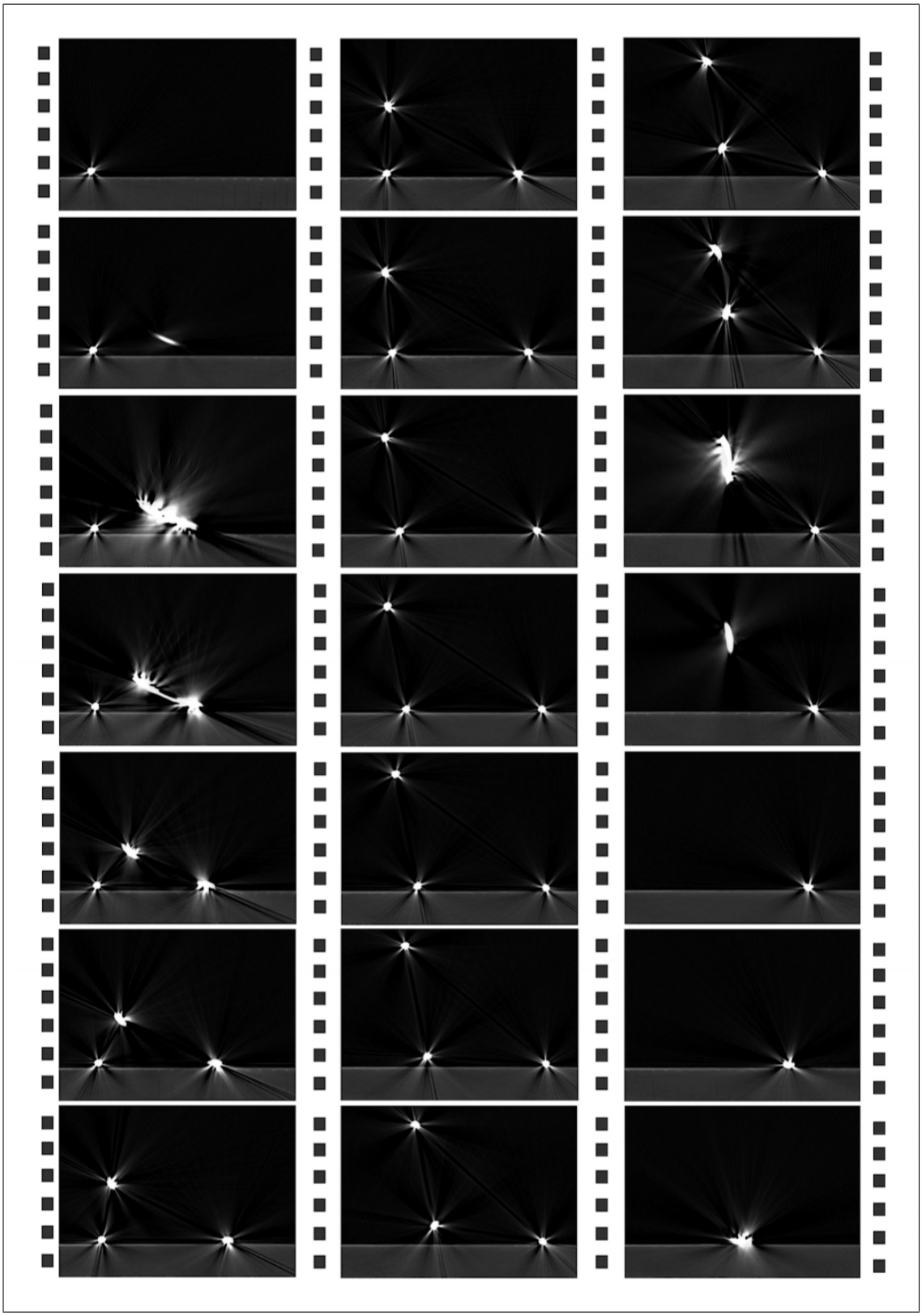


Figure 3.2: Slices of the original video of 2 394 frames (100 sec).

Open Access 2026-10-22

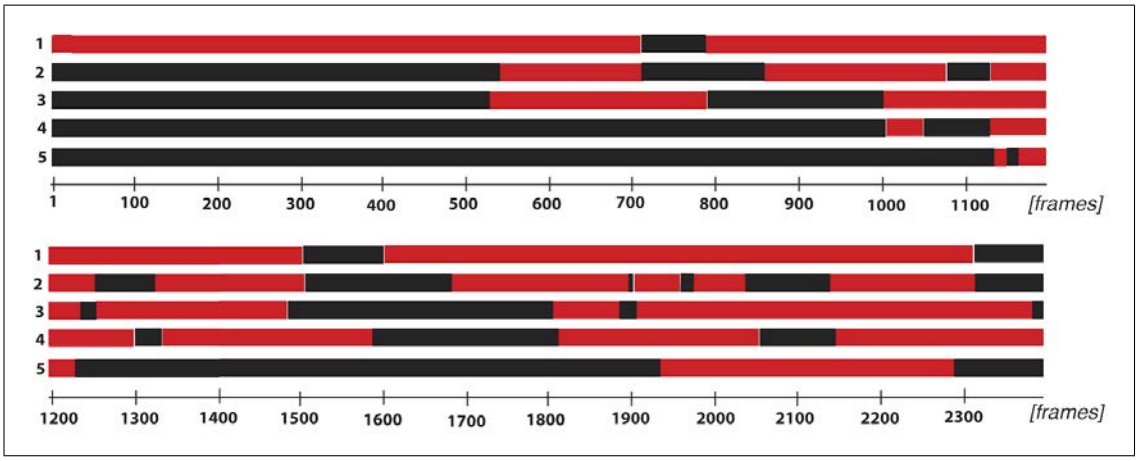


Figure 3.3: The score of CT, derived from the five identified orbits. These are displayed as horizontal lines, the one below the other. For each line, those frames where the respective orbit is active are marked red; inactive parts are coloured black.

3.2.3 Sound Synthesis and Processing Unit

Each orbit is linked to a different electroacoustic voice, 'sonifying' the movements of the sparks. As a result, the piece consists of five discrete (unfused) electroacoustic parts, imitating the feeling of divergent motions. The percept of different characteristics is explored by feeding similar videos to different timbre spaces.
There are 43 parameters assigned to a single voice, depicted in Figure 3.4. Together with the parameters controlling the volumes of the five voices, the low-pass filter used after their mix and the reverb (7 parameters), $43 \times 5 + 5 + 1 + 7 = 228$ parameters are available to control the sound generating engine.

3.2.4 Mapping

For each voice, the data received from the analysis of the video frames is mapped individually to the parameters of the sound generating engine with the help of a matrix.
As there are fewer parameters arriving from the video (15) than the total number of parameters of the sound engine (228), only a few selected musical parameters are controlled by the movements of the animation. Some parameters are kept constant, only for creating source sounds, subject to further dynamic changes. The list of dynamically changeable parameters is shown on Figure 3.5. Mapping is then reduced to the following two tasks:

1. Assigning control data to dynamically controlled parameters.
2. Defining the range of these parameters.

Figure 3.6 depicts the matrix assigned to the first voice. This matrix allows a huge number of variations: as each connection can be turned on and off individually, the connections themselves (not to mention the different scalings) offer $2^{(19\times4)} \approx 75\,558$ trillion combination possibilities. The different musical motifs of the piece are created by letting the movements of the video explore different routes through the timbre space, defined by the presets of the matrices.

Open Access 2026-10-22

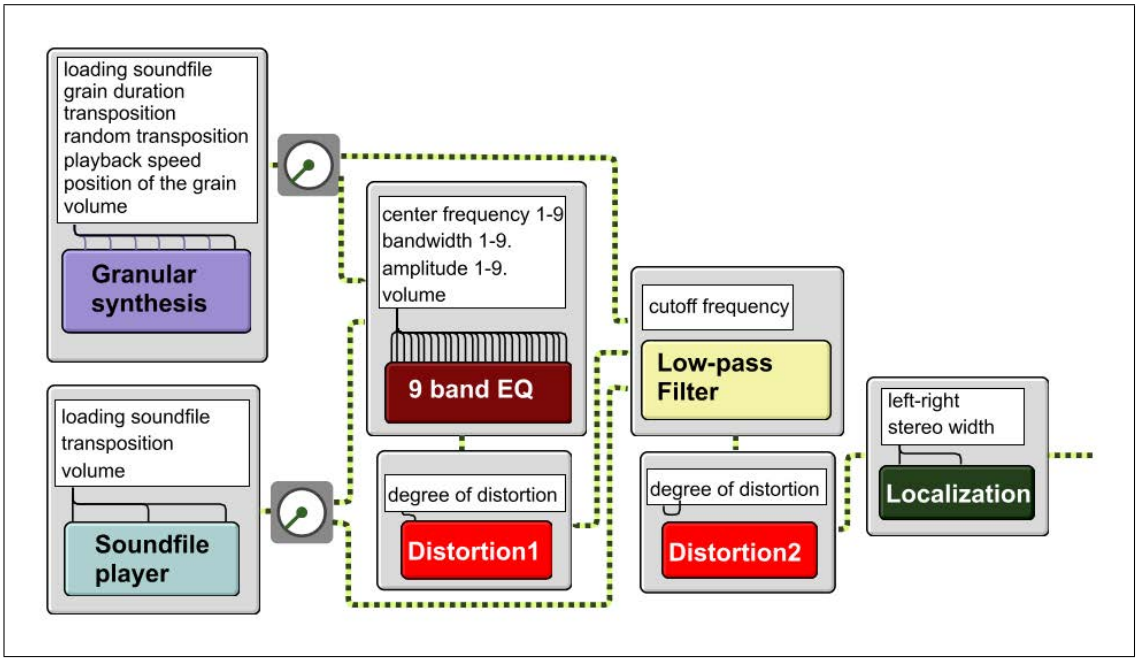


Figure 3.4: The structure of the sound engine of a single voice, showing the avaliable synthesis and processing techniques (soundfile playback, granular synthesis, subtractive synthesis and distortion), together with their controllable parameters and the possible routes between the sound generating units.

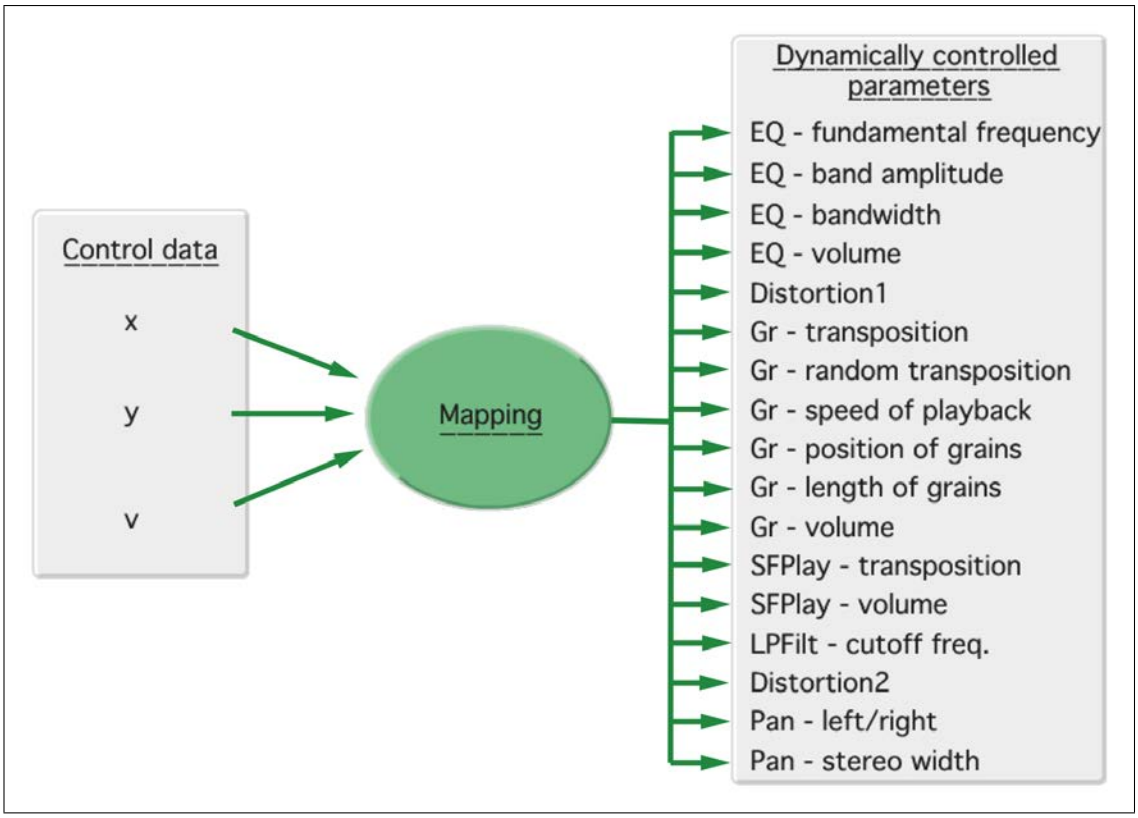


Figure 3.5: The parameters of the sound engine which are exposed to the dynamic control system based on the motion of the sparks.

Open Access 2026-10-22

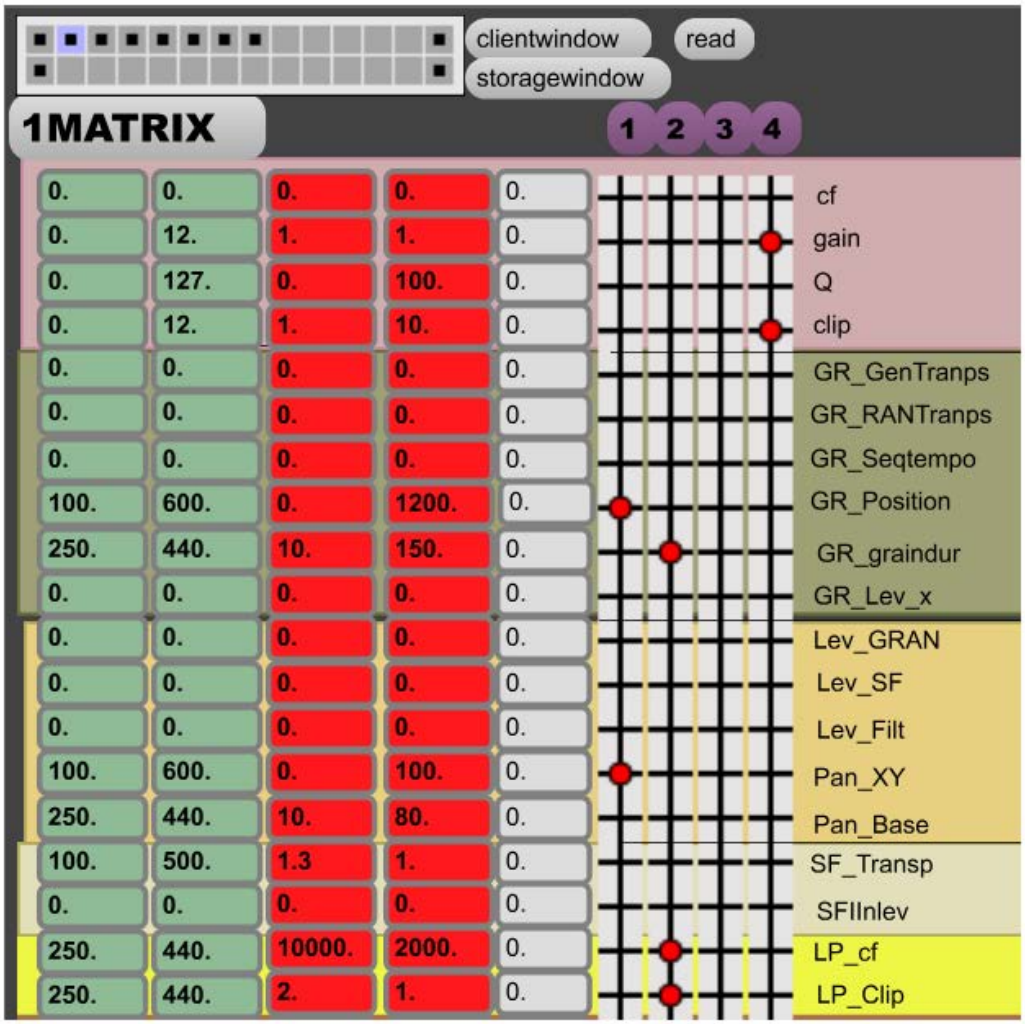


Figure 3.6: The matrix controlling the mappings within a single voice. The red dots specify the synthesis parameters that are assigned to the individual control data in a certain preset. The green number boxes set the ranges of the incoming control data, while the red ones define the allowed ranges for the controlled parameters.

Open Access 2026-10-22

Chapter 4

Listing and Description of Different Types of Sensors

A controller is a device for transforming the performer's gesture into abstract machine language (in most cases, a sequence of numbers or a non-stationary electronic signal). In a live scenario, this information may either be used to control the generation, the processing or the projection of sounds (or any simultaneous combination of these). However, the data arriving from a controller is not enough per se to master these processes, as the information is normally structured in a different way than the expected parameter structure of the controlled devices. Hence, proper *mappings* have to be defined between the information arriving from the controllers and the parameters of the different generating, processing and projecting tools. These mappings — and the musical implications of using controllers — are explored in Chapter 15. Here, we concentrate on presenting the most commonly available controllers.

4.1 Defining Controllers

Historically, controllers were inherent parts of the instrument: there is no way to break an instrument into a set of controllers, synthesis/effects, and mappings between these, as the mechanical controllers are also parts of the synthesis/effect engine. In live electronic music, it is just the opposite: these independently defined entities must be merged into an instrument, giving us the freedom to use literally any human-controlled device[1] to control sonic parameters. This freedom has a price, though. When 'breaking up' the instrument paradigm into these categories, we have to re-create some tasks that were automatically present in the 'composite' paradigm. Thus, in order to build a controller, one needs to define at least the following three aspects:

Communication channel. What is the means by which the controller communicates? Does it interact with the performer through physical connection? Through motion sensing? By eye-contact? By speech recognition?

Detection. How does the device detect the above means of communication in a technical sense? Does it include buttons? Sliders? Cameras? Electrodes?

[1] Or even human-independent, like a thermometer.

Open Access 2026-10-22

Feedback. Does the device offer any direct feedback to the performer? If yes, how is this feedback delivered?

In what follows, we present a few controllers in respect of the above issues, following a categorisation based on the conceptual distance of a controller from already existing instruments[2]. It must be emphasized that the borderlines between the following categories are by far not strict, though.

4.2 Augmented Instruments

This family consists of traditional instruments equipped with sensors. As they are built on existing instruments, the communication channel and the feedback given to the performer are defined by the base instrument. Also, even though augmented with sensors providing control data, the instrument still produces it's original sound.
A commercial product is Yamaha's *Disklavier* (depicted in Figure 4.1), a real piano equipped with sensors able to record the keyboard and pedal input from the pianist. Moreover, disklaviers are equipped with appropriate tools in order to reproduce the acquired performance data. As a result, they are not only used by live electronic musicians, but also serve as tools to record, preserve and reproduce actual *performances* (rather than the *sound of the performances*) of pianists.

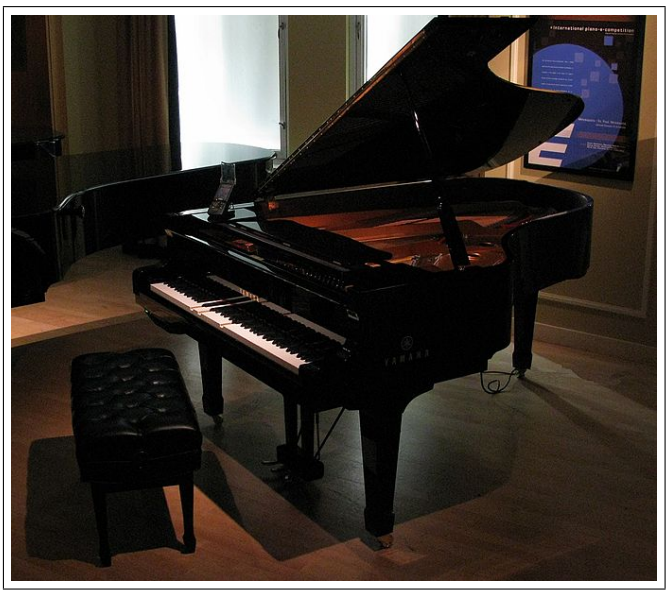


Figure 4.1: Yamaha Disklavier.

Different research centres have delivered a great amount of augmented instruments during the past 30 years. *MIDI Flute*, developed by IRCAM, adds sensors to the flute keys to identify the exact fingerings of the performed notes. This approach was expanded by several researchers, by adding inclination detectors or wind sensors. The Hyperinstruments (*Hyperviolin*, *Hyperviola*, *Hyperchello*) of MIT Media Lab combine bow motion and finger

[2]Based on E. R. Miranda and M. M. Wanderley, *New Digital Musical Instruments: Control and Interaction Beyond the Keyboard* (Middleton: A–R Editions, 2006).

Open Access 2026-10-22

placement detection with spectral analysis of the recorded sound to provide detailed real-time performance parameters. Augmented trumpets of Princeton Media Lab include, above key pressure detectors, air-pressure sensors (microphones) built into the mouthpiece of the instrument.
Hybrids form an interesting subcategory. These are instruments that, although based on traditional instruments equipped with sensors, they no longer produce their sound in the way the original instrument does. An example is the *Serpentine Bassoon* (see Figure 4.2), which is based on a classic *serpent* and is played with a conventional bassoon reed. However, its sound is produced electronically: it has a built-in microphone recording the acoustic sound produced by the instrument, which – after being processed by a computer – is delivered to the audience by means of electronic amplification.

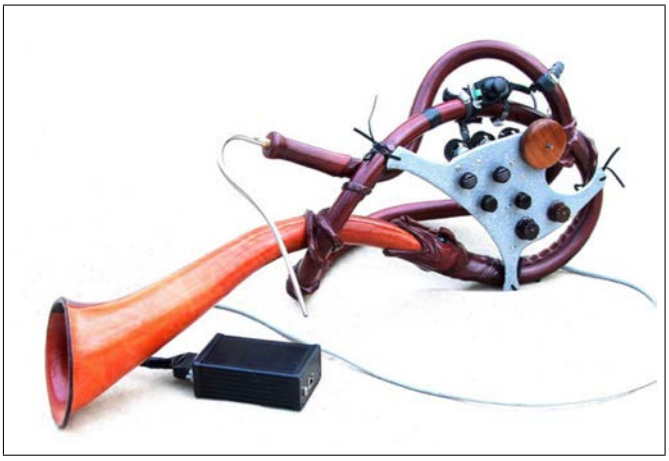


Figure 4.2: Serpentine Bassoon, a hybrid instrument inspired by the serpent. Designed and built by Garry Greenwood and Joanne Cannon.

4.3 Instrument-like Controllers

These devices may be imagined as if we took the 'control mechanism' out of an instrument, and possibly extending the performance possibilities (like adding additional octaves to the playable range). The reason for creating such controllers is twofold. Firstly, as most trained musicians play on acoustic instruments, a great number of potential performers may be reached with controllers that imitate the behaviour of the instruments they have already learned. Secondly, these performers already know how to master their instruments, bringing the opportunity of using advanced playing techniques in live electronic music.
Based on the construction, it is obvious that their communication channel is practically identical to the respective instrument's original set of gestures. However, the direct feedback of the controller normally lacks some of the feedback mechanisms present in the original instrument. This may be due to the lack of body vibrations (which is caused by the lack of direct sound emission) or by differences in the underlying mechanical solutions (like the different mechanisms implemented by a real piano and an electronic keyboard).
The by far most widespread such controller is the (piano-like) keyboard (see Figure 4.3). To highlight its importance, one should consider that early modular synthesizers – like the ones manufactured by Moog or ARP – were driven by keyboard controllers, or that the MIDI

Open Access 2026-10-22

standard is definitely keyboard-biased[3]. Modern keyboard controllers are normally velocity-sensitive, often pressure-sensitive as well (i.e. they may also detect the force applied when holding a key, also known as aftertouch).

Figure 4.3: MIDI master keyboards of different octave ranges. In fact, most master keyboards also contain a set of built-in alternate controllers, like dials and sliders.

Commercially available wind instrument controllers also exist, like the *Akai EWI* or the *Yamaha WX* series. This latter simulates a saxophone by measuring both the fingering and the air blowing pressure. Also, one may convert the WX5 into a recorder controller by changing its mouthpiece. Electronic string controllers are popular, too (see Figure 4.4).

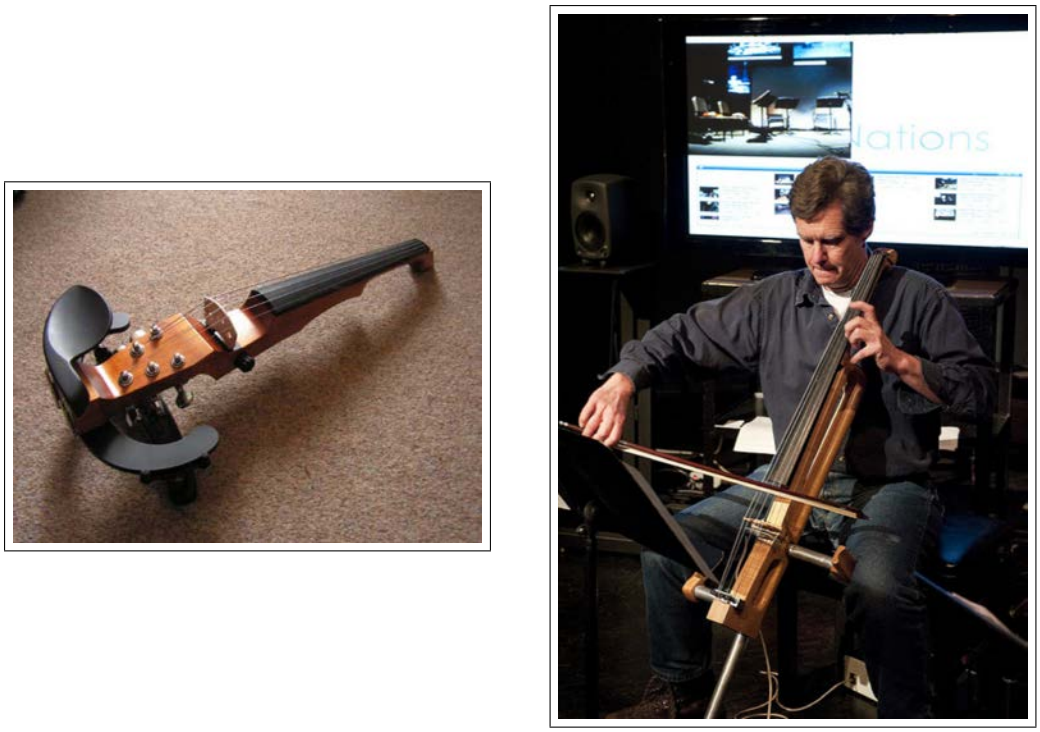


Figure 4.4: Electronic string controllers: violin (left) and chello (right).

[3] And has constantly been criticised for this by some researchers since its creation.

Open Access 2026-10-22

An exotic example of instrument-like controllers is the *Moog Etherwave Theremin* (see Figure 4.5). This controller resembles the interface of the original theremin, one of the first electronic instruments, invented by Lev Sergeyevich Termen in 1928. The theremin has two antennae to control the pitch and the loudness of the sound by sensing the distance between the antennae and the performer's hands. Moog's controller has the same two antennae, and (apart of being able to produce its own sound as well) transmits the captured distances as MIDI data.

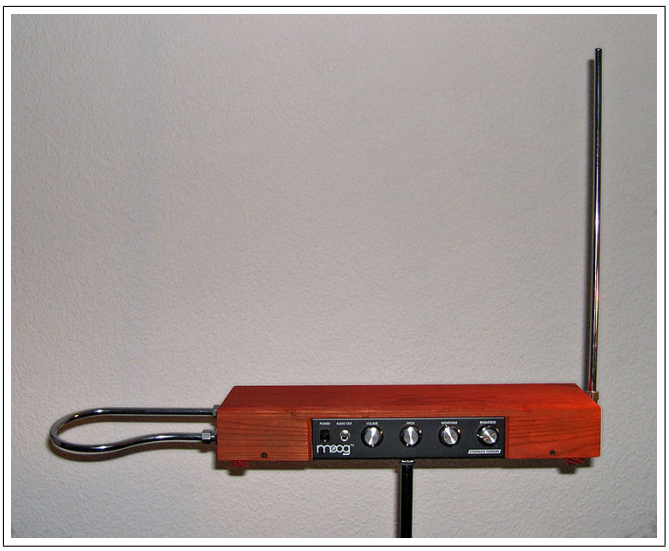


Figure 4.5: The Moog Etherwave Theremin.

4.4 Instrument-inspired Controllers

As a further step, it is possible to derive controllers which do not reproduce the controls of existing instruments, but just take inspiration from them. In some cases, they stay quite close to classic instrument concepts, and in others, the resemblance to any known instrument may seem almost purely coincidental.

The reason why one may call the controllers in this category 'instrument-inspired' is that the set of gestures that they understand (i.e. their communication channel) shares elements with the 'gestural vocabulary' of a real (family of) instrument(s). The more these two vocabularies overlap, the closer the controller is to the control surface of that real instrument.

The way of giving direct feedback as well as the detection principles depend largely on the actual 'extrapolation' of the instrument. Laser harps (depicted in Figure 4.6), of which many have been developed independently among the past decades, require the performer to touch the laser beams – the virtual strings of the harp –, imitating the way real harps are 'controlled'. Some of these instruments may also give visual feedback, as the laser beam is 'cut' when the performer touches it. Extended keyboard controllers may deliver the 'classic keyboard-like' feedback, however, they may be equipped with additional detection systems, not present on classic instruments – like additional control surfaces allowing glissandi. There are also wind controllers with structures very close to instrument-like wind controllers, which only differ from the latter in the application of non-conventional fingerings. This approach

Open Access 2026-10-22

is of particular interest to instrument builders who show an interest towards the creation of microtonal controllers based on non-conventional scales.

Figure 4.6: Two different laser harp implementations which both send controller values for further processing. The LightHarp (left), developed by Garry Greenwood, uses 32 infrared sensors to detect when a 'virtual string' is being touched. As the 'strings' are infrared, they can not be seen by humans. The laser harp (right), developed by Andrea Szigetvári and János Wieser, operates with coloured laser beams.

4.5 Alternate Controllers

These are the controllers that have nothing in common with mechanical instrument control surfaces. Their means of communication and detection mechanisms vary from controller to controller, which makes it impossible to talk about these in a generic way. However, we may distinguish between *touch controllers* (which require direct physical contact in order to operate), *expanded-range controllers* (which react to gestures performed *within a certain range* from the instrument and may or may not require direct physical contact) and *immersive controllers* (which capture the gestures of the performer *anywhere*, maintaining an unbreakable, always active connection with the performer, without the possibility to 'leave' the control surface). Except for touch controllers, alternate controllers usually do not have any form of haptic feedback, and even touch controllers do not offer much normally in this regard.
The most obvious and 'ancient' touch controller is the *button* (allowing the triggering of an event) and the two-state *switch*, but *sliders*, *dials* and other basic controllers, already familiar from common electronic household devices, also belong to this category together with (legacy) computer controllers like joysticks, trackpads or the mouse. Their gestural vocabulary is somewhat limited compared to more advanced controllers, understanding only one (or some of them, two) dimensional motions. Nevertheless, as they have been used as practically the only way to control electronic sound for quite a few decades, some musicians

Open Access 2026-10-22

gained a deep experience in mastering these tools. Sophisticated controllers, like *The Hands* (see Figure 4.7) combine the ability of playing on keys with built-in motion capturing.

Figure 4.7: The Hands, developed by Michel Waisvisz. An alternate MIDI controller from as early as 1984.

Currently perhaps the most widely used touch controllers are the touch-sensitive 2D screens of tablets, smartphones etc. As these come with highly programmable visual interfaces (and because they are more and more widespread), their usage within the community has constantly been increasing during recent times. Some solutions, like the *Reactable* (see Figure 4.8), build on a combination of touch sensing and image recognition – although its control capabilities are very limited.

Figure 4.8: Reactable, developed at the Pompeu Fabra University. The interface recognises specially designed objects that can be placed on the table and executes the 'patches' designed this way.

Expanded-range controllers include tools with limited motion sensing capabilities, like the *Radio Baton* by Max Matthews (see Figure 4.9), consisting of two batons and a detector plate, implementing 3D tracking of the location of the baton heads – provided that those are located within a reasonable distance from the detector plate. Other examples include commercial sensors like the *Wii Remote* or the gyroscope built into many smartphones. The *Leap Motion*, a recent development of 2013, tracks the motion of all 10 fingers individually, allowing the capture of quite sophisticated 3D hand gestures.
Immersive controllers also vary a great amount. They include ambient sensors (like a thermometer) as well as biometrical sensors (like EEG). However, most commonly they are as-

Open Access 2026-10-22

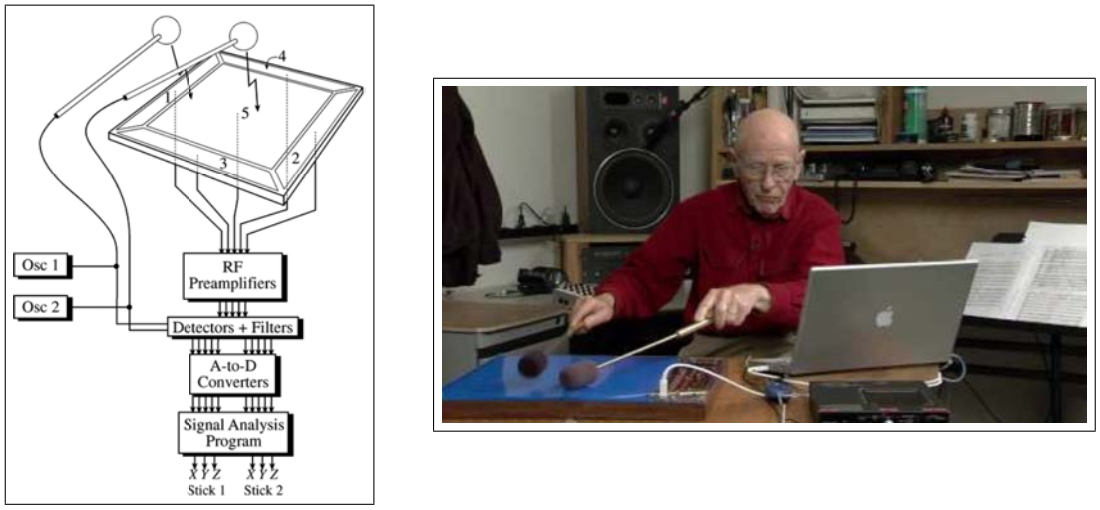


Figure 4.9: The Radio Baton (left) and its designer, Max Matthews (right).

sociated with body suits, data gloves, or full-range location sensing (like video-based motion tracking, as implemented by *Kinect*, depicted in Figure 4.10).

Figure 4.10: Kinect, by Microsoft.

Open Access 2026-10-22

Chapter 5

Structure and Functioning of Integra Live

The most well-known software paradigm in computer-based music is the *integrated system*, a complex environment for the recording, editing and playing back of music and control data, presented in Chapter 2. Although these have proven to be very efficient tools for any musician, their design makes them inappropriate for live electronic scenarios, especially in cases where algorithms or non-conventional mappings play an important role. Improvisation and music following an open-form are also among the art forms that, although not impossible, are very hard to realize with traditional integrated systems.
As the demand for tools suitable for the live performance and improvisation has increased, several software solutions have emerged which have extended the classic paradigm in certain ways. In this Chapter, we present such a software.

5.1 History

There are several reasons behind the development of Integra Live, each one having its own historical background.
During the first few decades of (live) electroacoustic music, the most important theoretical and technological issue was the invention and perfection of novel principles of (and devices for) sound design. The revolution of the past sixty years has given us a rich set of theoretical tools and hardware/software implementations in order to support the creative mind, with each step being marked by a myriad of notable compositions.
As is the case with any emerging human activity, the 'abstract language' describing electroacoustic music has been developing in parallel with these novel tools since the early days; however, the lack of such a generic language made (and perhaps still makes) it impossible to describe these compositions in a device-independent manner. As a consequence, the performability of many great electroacoustic works is tied to the life span of the actual devices and software tools which were once used to render them. This means that most pieces written no more than 20–30 years ago have disappeared from the concert halls, simply because their original equipment and/or software environments are no longer available.
With an increasing number of unperformable works and a decreasing life-span of most devices and software environments, the concern of preserving musical works in an implementation-independent format has gained more and more attention. By now, there are

Open Access 2026-10-22

several initiatives focusing on this issue, which have all emerged within the past ten years. Some of these concentrate on collecting original material (data files, original software and hardware etc.) which can be preserved on a long-term basis, so that the musical works could be performed through these original tools anytime in the future. Others tend to re-create the pieces in a device-independent manner, by preserving the actual algorithms behind the specific technological solutions, allowing future performers to re-implement the music on their own, current devices.

Another motivation behind Integra Live is the way in which computer music software emerged from the early days of this artform. The earliest tools were simple text-based computer languages, allowing a limited set of possibilities for sound creation and manipulation. With the development of more and more new ideas, the complexity of these tools increased as well; an advanced understanding of most professional software available today demands a high level of specialisation and training, often requiring years of practice before being able to master any of these instruments. While this led to the rise of a new professional category within musical performance (the *computer musician*), it also raised the bar for those musicians who, although not considering electroacoustic music their primary means for music-making, were interested in this emerging field.

Integra Live, an initiative of Lamberto Coccioli and the Birmingham Conservatoire, addresses both issues by creating an open framework for electroacoustic composition, performance and preservation. The project, begun in 2005, treats the data, setup and implementation of a piece as separate abstract entities, allowing a high-level conceptual description of (live) electroacoustic performances. This approach allows Integra Live-based pieces to keep their musical content independent from the actual implementations of the processes involved — thus, even if the underlying mechanisms are replaced by new software solutions, the pieces themselves would remain performable. Additionally, the separation of the musical material from the underlying technology, supported by a graphical user interface (*GUI*) that resembles well-known music software paradigms, makes the learning curve of this environment very easy for musicians interested in electroacoustic performance.

5.2 The Concept of Integra Live

The software is built upon a highly modularised framework, in which the functionality of the program is distributed in such a way that the modules form independent entities. This means that even if a module is replaced by another one (having the same functionality, of course), the software would still work. The consequence of having modules with well-defined functions is that the musical content (and even the technical setup) of the different pieces realised with Integra Live can be kept independent from the technical implementation of the modules.

A ‘piece’ consist of three important parts:

- A list of all (abstract) musical devices required by the piece and a full description of their functionality (including possible parameters and their meanings etc.). In practice, this means a full list of the Integra Modules (to be defined later in this Chapter) used by the piece.

- A full description of how these devices are interconnected with each other.

Open Access 2026-10-22

- A timeline, containing the values[1] of the parameters at each moment of the performance.

To achieve the above requirements, Integra Live offers a timeline-based editor, where we can define our devices, their connections and their time-dependent parameters. The following sub-sections describe how this is achieved.
The user interface has two basic views. The *Arrange View*, designed for use during (off-stage) composition, looks similar to the interface of most integrated systems: it implements a timeline, offering a custom number of tracks. However, in contrary to classic systems, these tracks do not directly contain performance data (eg. audio samples or control data). Instead, they contain sections of different performance setups, to be described below. Conversely, the *Live View*, designed for use during the performance itself, displays a reduced number of controls for each track, putting an emphasis on displaying only the elements which are really needed for the performer during a performance. The displayed controls may change between different sections of the performance automatically.

5.2.1 Structure of the Arrange View

The atomic elements of Integra Live are called *Modules*. A Module is a 'musical black box' which has a well-defined behaviour. It may have audio inputs and outputs as well as (graphical) controllers which control their exact functioning. Two Modules, together with their controllers are depicted in Figures 5.1 and 5.2. Examples of Modules include simple ones, like *Tap Delay*, a single-input-single-output Module, where the incoming signal may be delayed (and, optionally, fed back) by a given amount of time. Conversely, Complex Modules which have a great number of controls, like *Stereo Granular Synthesizer*, are not easy to describe in all cases.

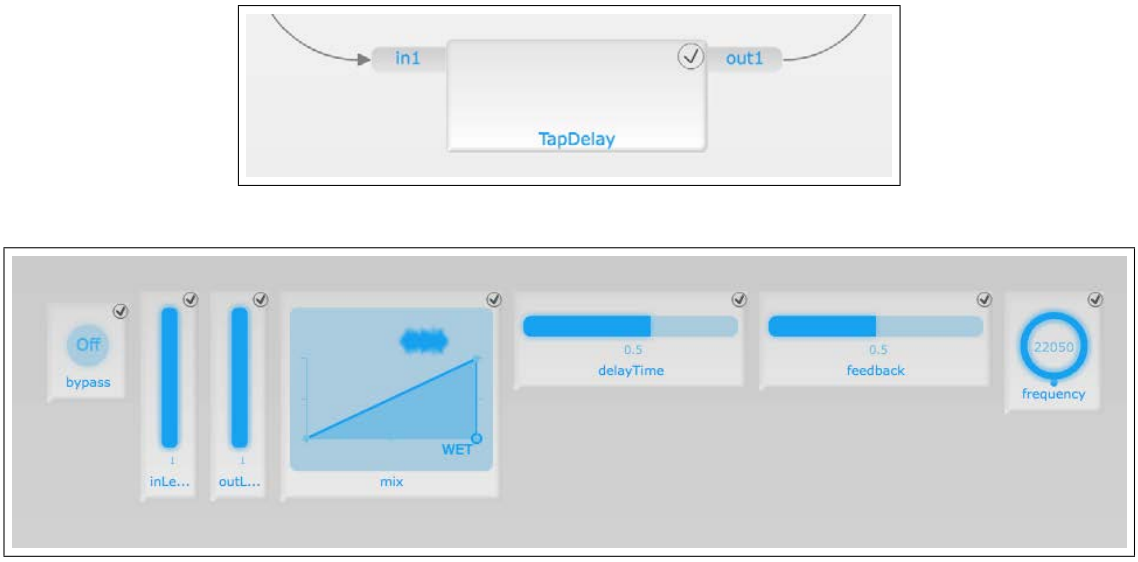


Figure 5.1: A Tap Delay Module and its controls.

As previously mentioned, one of the goals of Integra Live is the separation of (abstract) musical concepts from their implementations. Modules play a key role in achieving this:

[1] In this context, defining a parameter to be 'undefined', 'random', 'user-adjusted' etc. is also a valid 'value'.

Open Access 2026-10-22

the user only knows *what* the Modules do, without having an idea about *how* their task is achieved. As a consequence, one may replace the implementation of a Module with a novel implementation of the same task without affecting the pieces that are built on that very Module[2]; this makes any project realised with Integra Live quite robust against future changes in underlying technologies, separating effectively the life span of a musical piece from the life span of actual software solutions.

Figure 5.2: A Stereo Granular Synthesizer Module and its controls.

Modules may be connected to each other, forming a *Block*, which organizes Modules into logical groups that make musical sense, as depicted in Figure 5.3. An example may be a basic delay Block, where an audio input is connected to a tap delay, which is again connected to an additional effect (eg. limiter, filter etc) and finally, the result is sent to the audio output. Some example Blocks are shipped with Integra Live, including a pitch detector, a soundfile looper and a spectral delay, although the exact number and type of examples may change from version to version. However, the user is free to build any number of custom Blocks.

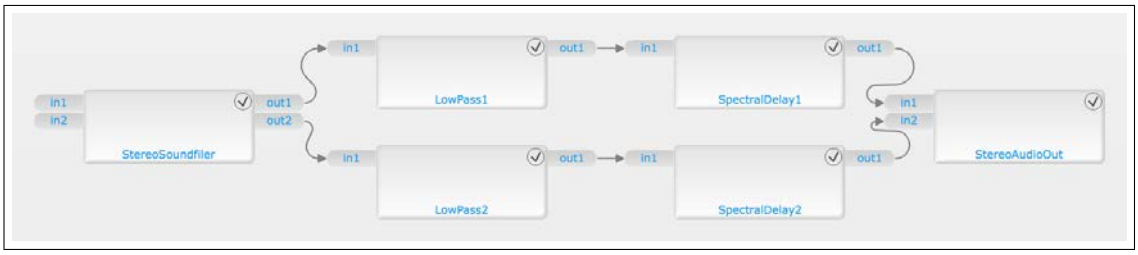


Figure 5.3: A Block containing a spectral delay.

The Blocks are the building elements of the *Tracks*, which allow multiple Blocks to be active at the same time (see Figure 5.4). Now we can see why Integra Live, although its design is inspired by classic integrated systems, is clearly different in its functioning. Instead of containing fixed musical information, like recorded sound, the Tracks in Integra Live contain Blocks of Modules, where only the *setup* of the musical devices is fixed, but not (necessarily) their *musical content*; the musical content depends on the '*interpretation*' of the setup.
Blocks and Tracks are the entities of Integra Live that describe the way the different musical devices (Modules) required by a piece, are interconnected with each other. They allow both a

[2]In fact, there are already a few Modules that have multiple, independent implementations; and, in theory, the same Module may have an unlimited number of parallel implementations within Integra Live.

Open Access 2026-10-22

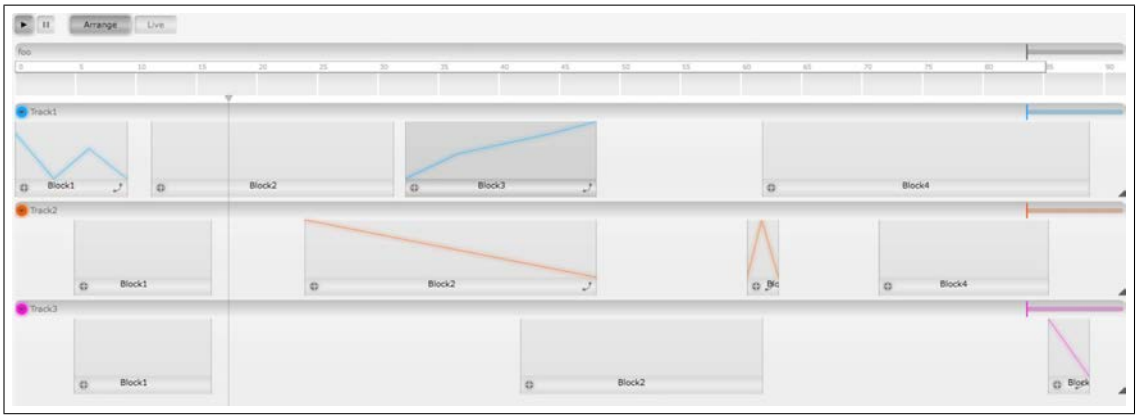


Figure 5.4: An example project containing three simultaneous Tracks. A few Blocks contain controllers controlled by envelopes.

hierarchical and temporal arrangement of the different devices and their jobs and they define the actual connections between these musical devices.

The lower part of the Arrange View contains the Properties Panel, which is different for each distinct context (Modules, Blocks, Tracks). When a particular Module is selected, this panel displays all of its controls (as shown in Figures 5.1 and 5.2), allowing the fine-tuning of their values as well as an option to add or remove them from the Live View (to be discussed below). In the other contexts, it allows to set up *Routings*, define *Scripts*, or edit *Envelopes*.

Routings are the mappings that can be defined between the different Module controls and the external data sources (eg. MIDI keyboard). For example, one can send the values of a specific MIDI slider to the delay time of the Tap Delay Module. Example routings are depicted in Figure 5.5.

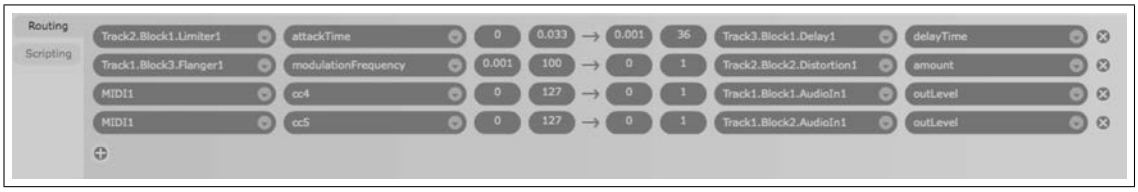


Figure 5.5: Routings between different elements and external (MIDI CC) data sources.

Envelopes can generate automated control data for the Modules. An envelope is always linked to a specific parameter of a specific Module, and defines its value over time. Graphically, envelopes are line segments with multiple breakpoints (see Figure 5.4). The composer will set the exact number and position of these breakpoints by clicking with the mouse; the rest of the contol data is generated by interpolating between these breakpoints. Obviously, if the value of a parameter is controlled by an envelope, it will not respond to any other incoming control value (eg. set by the mouse or by a routing).

The behaviour of scripts is more complex and depends on the actual task that the user-defined script describes. The scripting language of Integra Live is based on *Lua*. Scripting may be very useful in compositions involving aleatorics, algorithms supporting live performance or very complex mappings (eg. neural networks).

Open Access 2026-10-22

5.2.2 Structure of the Live View

The Live View is designed to be used during a live performance. Instead of showing the time boundaries and envelopes of the different Blocks (as is the situation in the Arrange View), it only displays the relevant controls of each Track (see Figure 5.6). As time elapses, some Blocks become active, while others deactivate. Since in Live View only the controllers of the active Blocks are being displayed, the layout changes automatically during a performance: the software will hide every inactive controller and display the activated ones according to the current state of the Blocks.

Figure 5.6: The most relevant controllers of each Track, as shown in the Live View.

Besides being able to see the most relevant performance controllers, one may also navigate through the different *Scenes*. A Scene is a pre-defined section of the timeline, with three possible behaviours. In *Hold* mode, the playhead will be set to pause, while in *Play* and *Loop* mode, it is set to play when the Scene is activated. The difference between the latter two modes is that, once the playhead reaches the end of the Scene, it stops in play mode, whereas it jumps back to the beginning of the Scene in loop mode. Figure 5.7 depicts a few Scenes.

Figure 5.7: Three Scenes in the timeline of Integra Live.

Scenes may be triggered either by mouse interaction or by pressing a key on the keyboard. Since it is up to the user to assign Scenes with keystrokes, the Live View also contains a small keyboard icon showing the actual shortcuts of the different Scenes. This is shown in Figure 5.8.

Open Access 2026-10-22

Figure 5.8: The 'virtual keyboard' of the Live View, showing the shortcuts of Scenes.

Scenes play an important role in the high-level temporal organisation of a piece by allowing control possibilities over the internal time-flowing of a composition.
It must be kept in mind that, if one clicks with the mouse on a gray area of the timeline which contains a Scene, the playback behaviour would automatically be that of the Scene. That is, if the behaviour is play, the playhead would stop at the end of the Scene. Therefore, to force the timeline to advance the playhead in an uninterrupted way, one needs to set the playhead using the small white area below the gray (see Figure 5.7).

5.3 Examples

The project `le_05_01_integra.integra_` file is downloadable using the following link: `http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_05_01_integra.integra_`
The project `le_05_01_integra.integra_`illustrates the above. The project opens in Live View. It contains five pre-defined Scenes, called Start, Loop, Click1, Click2 and End. Press the play button to play the performance.
If we play the entire 'piece', we'll see how the layout changes as different Blocks become active. We also see that, during the different Scenes, some control values are changed automatically, like the panning in the Start Scene or the microphone level in the Scene Click2. In Scenes Click1 and Click2 (which can either be loaded by clicking with the mouse on the respective parts of the timeline or by pressing the respective keyboard keys associated with them), a sound file player is displayed. Here, we should load an arbitrary sound file and start playing it (by clicking on the big trigger to the left side of the waveform display, see Figure 5.9).

Figure 5.9: The sound file player. Click on the 'Load File' *field to load a sound file. Click on the* 'Bang' *to start playing it.*

To see how the use of Scenes may help the performance of a piece following an open or improvisatory form, try experimenting with different sequences of Scenes (for example,

Open Access 2026-10-22

Click1→Click2→Loop→Click2→Start→End). You may activate these Scenes by pressing their keys of the keyboard associated with them (see Figure 5.10).

Figure 5.10: The keyboard shortcuts associated with the Scenes of the demo project.

We may switch to Arrange Mode with the toggle on the top left of the display (see Figure 5.11). Here, two Tracks are displayed, as depicted in Figure 5.12. Track1 contains two Blocks, while Track2 contains a single Block. We can inspect the contents of each Block by double-clicking on them, opening the *Module View*.

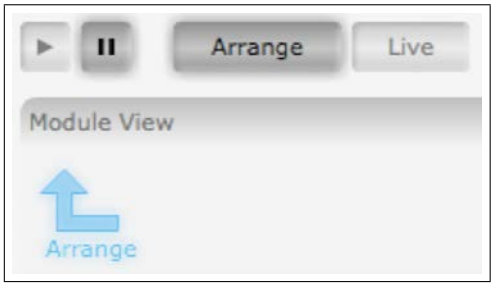


Figure 5.11: The main switch between the Arrange and Live Views. Also depicted are the playback controls (top left) and the button (in blue) that would return us from the Module View to the Block View.

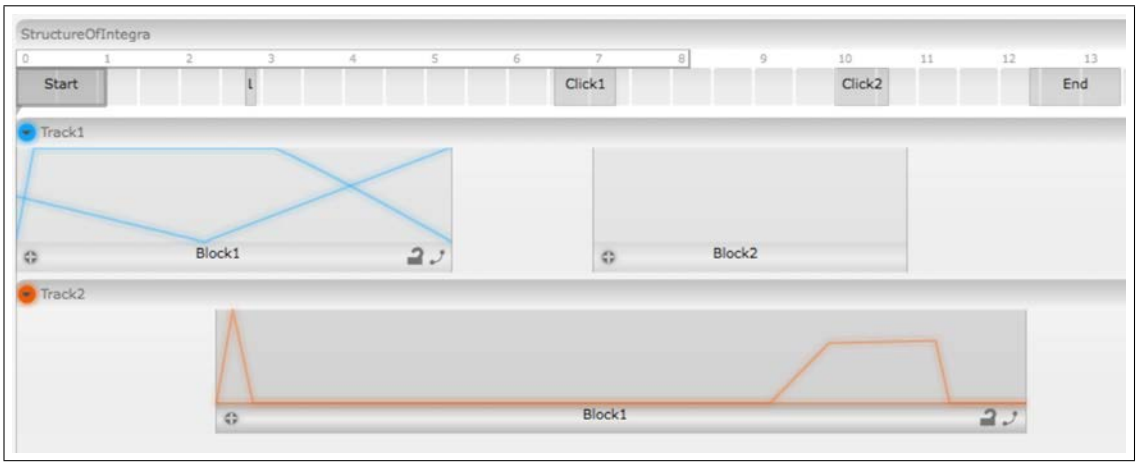


Figure 5.12: The demo project, in Arrange View. We may see all the involved Envelopes and Scenes.

The first Block of Track1 contains three Modules: a *Test Source*, a *Stereo Panner* and a *Stereo Audio Out*. By clicking on them, their controls will be displayed in the Properties

Open Access 2026-10-22

Panel. For example, by clicking on the Test Source, we may realize that the test tone of this patch is a pure sine wave of 220 Hz (see Figure 5.13). After returning to the Arrange View (by clicking on the arrow of the top left corner of the Module View), we may inspect the other two Blocks. The second Block of Track1 contains a very simple file player. Finally, the only Block residing in Track2 is a tape delay line, consisting of two different sources, a low-pass filter and a 'classic' tape delay with feedback options, as depicted in Figure 5.14. The two incoming sources are the microphone and white noise, both of which are connected to a low-pass filter with a very low (approx. 45 Hz) cut-off frequency. This filtered source is sent to the tape delay line, which also has a built-in low pass filter for its feedback circuit. Comparing this to the Live View, we realize that not all of these controllers are displayed. In fact, only the output levels of the two sound sources, the delay time, the feedback factor and the low-pass filter connected to the feedback circuit are shown. During a performance, we can change the feedback values or the delay times, and we can influence the timbre as well by changing the properties of the filter attached to the feedback circuit. However, by hiding the main low-pass filter before the tape delay from the Live View, we can assure that the main character of the sound of this Block remains unchanged.

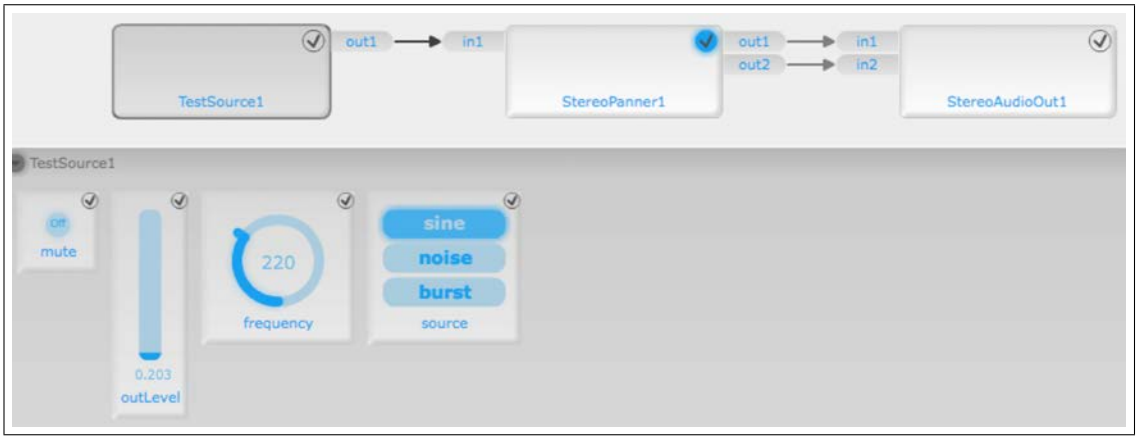


Figure 5.13: The first Block of Track1, under Module View. The Properties Panel displays the controls of the test tone Module.

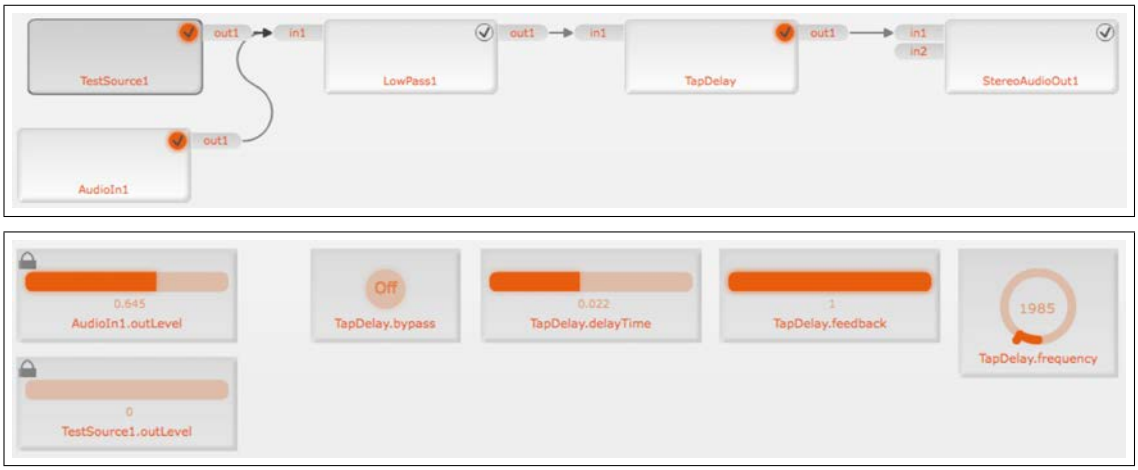


Figure 5.14: The first Block of Track2. Top: Module View. Bottom: Live View.

Open Access 2026-10-22

Finally, we may see in Figure 5.12 that the first Blocks in both Tracks contain several envelopes. By clicking on any of them, the Properties Panel would show the name of the controller that is being controlled by the selected envelope.

5.4 Exercises

1. Routings can be defined within the Properties Panel of the Arrange View, by clicking on the 'Routings' label and the + sign that appears. Define at least three different routings in the `StructureOfIntegra_project`! If you have an external MIDI controller, see how it works by creating routings involving MIDI as well.

2. Add an envelope to the second Block of Track1 in the `StructureOfIntegra_project` to fade in and fade out the sound file that is played back!

3. Create a new Integra Live project by selecting File→New from the main menu! The project will open in the Arrange View. Enter the Module View of the first Block by double-clicking on it. Now, create a simple Block by dragging Modules from the Module Library on the right side of the screen to the main canvas. Connect the audio inputs and outputs of the newly added Modules to each other by click-and-dragging with the mouse. Experiment with the result.

4. In your new project, navigate back to the Arrange View. Add a new Track to the project by double-clicking on the empty part of the main display. Create a new Block in the new Track that overlaps with the Block in the previous Track! Now, enter the Module View of this new Block, and create another patch of Modules.

5. After returning to the Arrange View, create at least three different Scenes by click-and-dragging the mouse over the timeline. You may set the properties of the Scenes (eg. whether their role is hold, play or loop) by clicking on the Scene itself: the properties will be displayed in the Properties Panel.

6. Finally, add some controllers to the Live View. To do this, enter the Module View of each Block, click on the Modules that have controls that you wish to include in the Live View, and select the controls that need to be included in the Live View by clicking on the check icons (displayed on the top right corner) of these controls.

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

Chapter 6

Structure and Functioning of Max

In Chapter 5, we presented music software that allows the user to create live electronic pieces by organising a set of abstract devices into hierarchical and temporal structures. Although this approach is very convenient for quickly building software solutions that support our musical ideas, our possibilities, both musical and technical, are highly constrained by the available Modules in Integra Live. In this Chapter, we present Max, a (Turing-complete[1]) programming environment designed specifically for computer music. Unlike Integra Live, Max does not have any high-level concept about our musical workflow; we need to translate our musical ideas into algorithms, which we then need to build from scratch using the 'language elements' of Max. Although creating music with Max requires a rather algorithmic approach, the benefit of being literally unconstrained by technical limits is what has made Max one of the most popular tools among musicians inclined towards experimental electroacoustic music.

6.1 History

The 'original' Max software was developed by Miller Puckette at IRCAM during the 1980s, called '*Patcher*'. This software realized MIDI and control processing; audio manipulations were done by external devices controlled by Max via MIDI. IRCAM both licensed the software to Opcode (and later to Cycling '74) and they also developed a version called *Max/FTS*, which enabled real-time audio processing on dedicated hardware. A later variant of Max developed by IRCAM was *jMax*, a Java-based multiplatform version of Max.
Today, two main branches of the original software are widely used. One is the commercial variant, developed by Cycling '74; the other is *Pure Data* (abbreviated *PD* or *Pd*), an open-source reimplementation of the software started by Puckette himself in 1996.
During the late 1990s, both platforms were expanded in order to support real-time audio manipulations. This extension was released, for Max[2], in a package called *MSP* ('Max Signal Processing'), by David Zicarelli[3]. A few years later, live video processing packages

[1]This means, loosely speaking, that we can implement any algorithm with the language.

[2]The first published version of PD already included these tools, hence the package doesn't have a separate name in PD.

[3]MSP was originally released as a plug-in for the Opcode version of Max. However, after Opcode lost interest in the further development of the product, Zicarelli founded Cycling '74 as a company solely dedicated to developing Max.

Open Access 2026-10-22

were added, *Jitter* in Max and *GEM* in PD. Although for some time the commercial version of the software ran under the names Max/MSP or Max/MSP/Jitter, the latest version of the program has reverted to the original name 'Max'.
In general, many 'patching' artists exhibit a strong preference towards one environment or the other, generating strong debates within the community. The most common arguments in favour of PD are its portability[4], its openness and the way how it can be embedded in other environments. On the other hand, Max has been favoured for a long time for its more advanced GUI and lately, for some new technologies that are not currently present in PD. These include *Max4Live* (the possibility of embedding Max patches into a popular music production suite, called Ableton Live), *Mira* (which mirrors the interface of any Max patch to iPad) or *Gen* (which adds extended programming capabilities to Max).
Strangely, a considerable amount of people would not consider the fact that PD is a free software as a supporting argument.
In the rest of this chapter, we restrict ourselves to the version of the software maintained and distributed by Cycling '74. However, most of the following description may apply to PD as well.

6.2 The Concept of Max

A textual programming environment demands the programmer to write source code describing the logic of the program using an artificially created language. In contrary, a visual programming environment expects the programmer to create graphical dataflows representing the logic behind the software.
Max is such a graphical programming language. Programs (called *patches* or *patchers*) consist of two main building elements:

Objects (also called *externs* or *externals*) are the building blocks of the program, very similar in their concept to the Modules of Integra Live. Each object has a well-defined behaviour, like multiplying two numbers or generating a square wave. Most objects have internal variables as well, which can be set by sending messages to them. In most cases, their graphical representation is a box, although objects with interactive graphical representations also exist. They accept data through their *inlets* and send out their results through their *outlets*. Inlets are always displayed on the top and outlets are always on the bottom of the object boxes. Objects dealing with audio signals follow a unique naming convention in that the last character of their names is always '∼'.

Patch cords are line segments that connect the boxes with each other, representing the allowed data flow between the objects. A patch cord will always connect the outlet of an object with the inlet of another one. Therefore, data may flow in only one direction through a patch cord.

Max is an event-driven programming language. An *event* is any action that can be detected by the program. Events may be generated by user interaction (e.g. pressing a key on the keyboard, clicking with the mouse, performing on a MIDI-device or using a sensor connected

[4]Currently, Max runs on Mac OS and Windows, while PD also runs on Linux and, as the source code is available, it could be compiled to any platform – in theory.

Open Access 2026-10-22

to the computer etc.) or by event schedulers (e.g. an internal metronome). These events travel through the patch cords and trigger the objects in different ways. Thus, the process of programming in Max consists of creating 'event-handling flows' (or data flows) by placing objects in the patcher and connecting their appropriate inlets and outlets.
When designing a data flow, one has to be mindful of message ordering – in other words, the order in which the different objects react to incoming messages. The most important general rules to remember are as follows:

- If an object outputs data from more than one outlet as a reaction to a single event, it always sends the messages out in a right-to-left order. That is, the *rightmost* outlet is always the first to react, followed by the other outlets.
- Objects having more than one inlet will not be triggered by incoming messages unless they receive a message on their *leftmost* inlet.

Although there are exceptions from these two rules, they are sufficient to determine the exact behaviour of a patch in most cases.

6.2.1 Data Types

Several different data types are allowed to flow on patch cords, and patch cords transmitting different data types have different colouring schemes. These are:

- Ordinary data types (thin black cords), including:

  Numbers. A distinction is made between integer and non-integer numbers (called *floats*).

  Symbols. A symbol is a sequence of characters (like a single word in written language).

  Messages. A sequence containing any number of symbols and numbers whose first element is a symbol (therefore, a symbol can be interpreted as a one-element message). Messages are usually used in order to trigger actions or set internal values of different objects.

  Lists. A sequence containing any number of symbols and numbers whose first element is a number. In fact, most objects do not make any difference between messages and lists and use the collective term *list* for both.

- The *signal* data type (thick black cords with laced yellow lines), transmitting audio signals between objects. In contrast to ordinary data types, where individual events are generated only when a new piece of data arrives, the audio signal flows in an uninterrupted way: as long as Max's *digital signal processing* (DSP) is turned on, the objects dealing with audio signals are constantly active.
- The *matrix* data type (thick black cords with laced green lines), which represents a single frame of video data.

Originally, Max only processed MIDI-like control messages. As the data rate of these messages is at least an order of magnitude below the data rate of digital audio, they are usually

Open Access 2026-10-22

executed at *control rate*[5]. Additionally, since the introduction of audio processing capabilities, a second data flow speed – the *signal rate* – was introduced, i.e. the sample rate of the sound card. Generally, messages at control rate and signal rate cannot be mixed, although there are some objects that will accept both kind of inputs.

6.2.2 Structuring Patches

It is possible to load other patches into a patcher. In this case the embedded patcher will act as if it was itself an object. There are two different encapsulation models:

- Patchers explicitly designed for inclusion into other patchers are called *abstractions*. The purpose of creating abstractions is the encapsulation of well-defined specific tasks that might be used at several different points of the main program; therefore, creating an abstraction is a matter of *software design*. As with 'real' externals, there are two possibilities for including abstractions in a patch:
  - The abstraction may be represented by a single object box, like any other extern that has no graphical controllers.
  - The original graphical layout of the abstraction can be displayed by loading the abstraction into a special object called `bpatcher`.
- One may also encapsulate a smaller part of a big patcher into a single patch, represented by a single object box. Such 'embedded patchers' are called *subpatches* (or *subpatchers*). The purpose of encapsulating a part of a big patch into a single subpatch is to facillitate the visual organization of the program; thus, this action is a matter of *interface design*.

The difference between subpatches and abstractions is that subpatches only help the visual design of a patch, whereas abstractions truly act as ordinary objects: there can be several instances of the same abstraction within a patch and every instance will behave the same way[6].
To run a patch, one needs to have Max installed on ones computer — or at least the free Max Runtime, which allows the opening and running of patches but prevents their modification. However, it is also possible to convert a patch into a normal, self-contained computer application, on both Windows and Macintosh; in this case, the user needs neither Max nor Max Runtime. The Max examples in this syllabus were first created in Max and then converted into standalone applications. For this reason, when writing about Max examples, we use the terms 'patch', 'program' and 'application' interchangeably.

6.2.3 Common Interface Elements and Object Categories

Most objects in Max will display in an *object box* containing the name of the object and (optionally) its parameters. However, there are a few objects that have their own interface.

[5] In modern implementations of Max this means an execution speed in the magnitude of an event per millisecond.

[6] While it is possible to clone a subpatch – which *looks like* having multiple instances of a subpatcher –, the cloned instances of a subpatch are, in fact, individual subpatches: they can be modified individually, without affecting the others. On the contrary, modifying an abstraction will cause a modification in every instance of that abstraction.

Open Access 2026-10-22

Some of these use this feature to display content to the user in an advanced way (e.g. the `spectroscope~` object, which displays the spectrogram or the sonogram of a sound) while others (like the different sliders, knobs and similar elements) accept (mouse-generated) input from the user. Table 6.1 and Figure 6.1 show the most basic objects accepting user input.

Table 6.1: List of the most common objects for direct user interaction.

Name	Description
`button`	Triggers other messages and processes
`toggle`	Switches between 1 and 0 (on and off)
`number`	Displays and outputs numbers
`slider`	Outputs numbers by moving a slider with the mouse
`dial`	Outputs numbers by moving a knob with the mouse
`kslider`	Outputs MIDI-compatible Pitch and Velocity values by moving the mouse over a piano keyboard
`pictslider`	Like `slider` but in 2 dimensions
`rslider`	Select a range of numbers with the mouse
`multislider`	An array of sliders
`radiogroup`	A check box or radio button group. The behaviour (check box vs. radio button) can be set internally
`umenu`	A drop-down list to select between various options

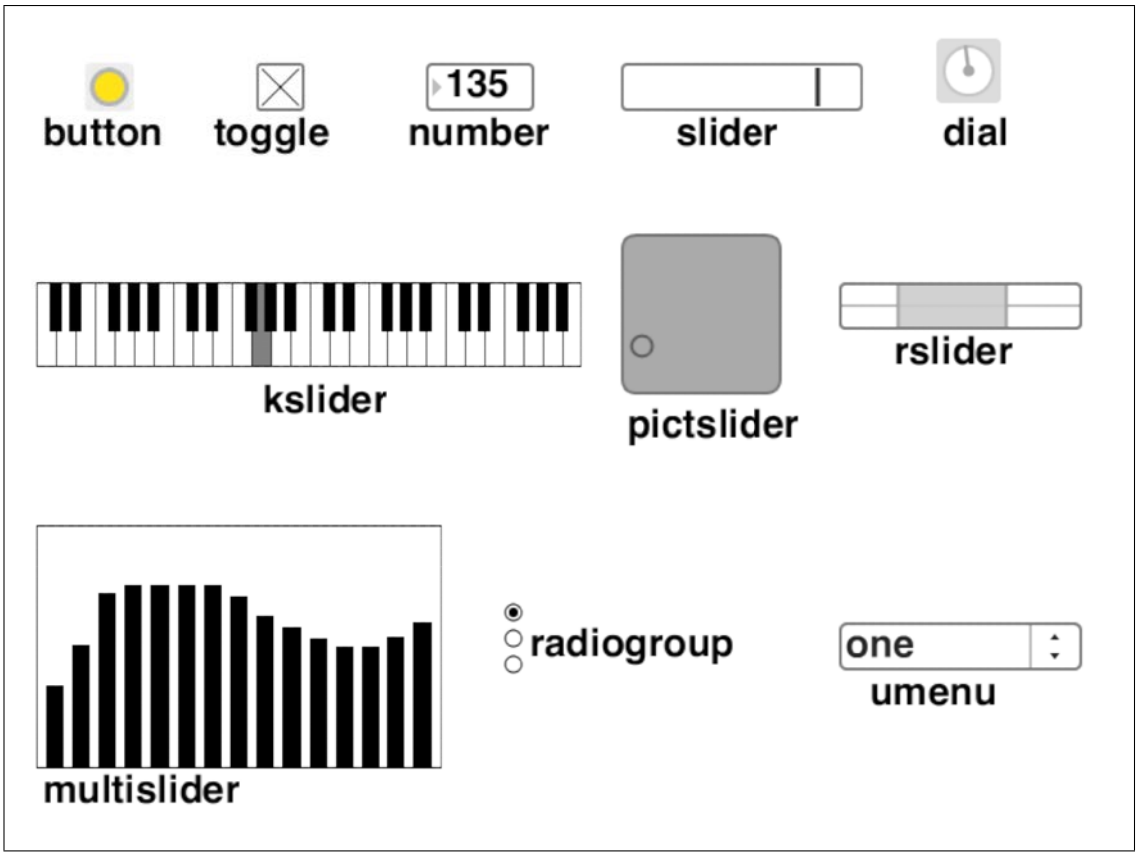


Figure 6.1: Most common objects for direct user interaction.

Open Access 2026-10-22

In addition, in order to create patches, one needs to be aware of the actual objects available as well – i.e. the most important ones. A proper presentation of all objects is far beyond the scope of this text, though. Moreover, as objects are independent dynamic libraries loaded by Max, third-party developers may extend the available palette of objects with their own externals, preventing us from presenting *every* available external.
The built-in externals of Max are organised into several categories by Cycling '74. The most important are:

- Control, Data & Math
- Devices
- Files & System
- Interaction & User Interface
- Lists & Messages
- MIDI, Notes & Sequencing
- Patching
- Timing
- MSP Objects (subcategories: Analysis, Delays, Dynamics, FFT, Filters, Functions, Generators, I/O, Modifiers, Operators, Polyphony, Routing, Sampling, Synthesis, System, UI)
- Jitter Objects (subcategories: Analysis, Audio, Blur/Sharpen, Colorspace, Compositing, Data, Devices, Generators, Lists, Math, Networking, OpenGL, Particles, QuickTime, Spatial, Special FX, Strings, UI, Utilities)
- Max4Live Objects

6.2.4 Mapping and Signal Routing

Max has a number of objects to control signal and data routing. The simplest ones are `gate_` and `gate~`, which send an incoming message or signal to a specified outlet, allowing the splitting of a single data flow into multiple paths. `switch_` and `selector~` (for ordinary data and signals, respectively) do just the opposite: they accept multiple incoming data flows but would let-through only one of them.
For more complex routing tasks, Max has a matrix-like graphical controller, which has versions both for ordinary data (`router`) and signals (`matrix~`). These accept multiple input flows and send out multiple output flows. The exact routing of the inputs to the outputs (allowing the mixing of the inputs with each other as well) is controlled by messages sent to these objects.
There are a few basic objects which implement different scalings. `scale_` and `zmap_` are generic tools which map an input domain to a specified destination. `mtof`, `ftom`, `dbtoa_` and `atodb_` convert between MIDI pitches and frequency values as well as decibel and amplitude values, respectively. Scalings based on more complex mathematical formulae can be

Open Access 2026-10-22

achieved by the expr_object, which evaluates a user-defined mathematical expression. Finally, lookup-table scalings (when each incoming data is mapped to a value defined in a table) can be realised with the table_object.
Defining proper mappings and scalings between the data received by an interactive system and the unit generators interpreting this data is an essential musical task, discussed in details in Chapter 15. It is worth remembering that the above-mentioned objects are the most important ones in Max to implement such mappings, though.

6.3 Examples

LEApp_06_01_is downloadable for Windows and Mac OS X platforms using the following links: http://www.typotex.hu/upload/Szigetvari_interaktivok/_SzigetvariAndrea_LiveElectronics_ENG/06_WIN.zip, http://www.typotex.hu/_upload/Szigetvari_interaktivok/SzigetvariAndrea_LiveElectronics_ENG/06_OSX.zip_
The LEApp_06_01_patch illustrates some of the objects described above. On opening the patch, five subpatches are revealed, called basics, embedding, routing, mapping_and templates, as shown in Figure 6.2. The 'p' before their names indicates that these are neither objects nor abstractions, but subpatchers. In what follows, we present each subpatch in detail.

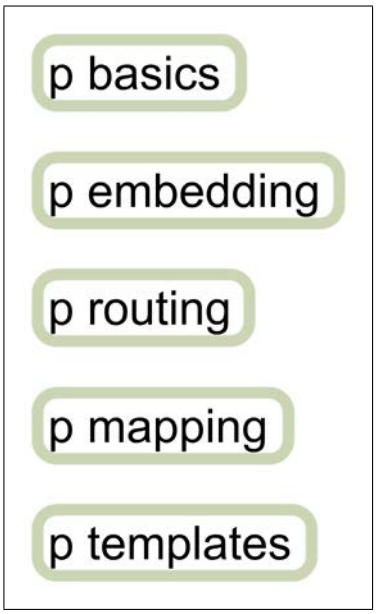


Figure 6.2: The application LEApp_06_01. Subpatches open by clicking on their boxes.

6.3.1 Basic Operations

The subpatch 'basics' (see Figure 6.3) demonstrates simple data flow and basic user input. Example *a)* multiplies two integers. The central object (called '*') has two inlets to receive the two numbers to multiply and has an output for the result, connected to a number box displaying this result and a button that blinks each time an output has been sent out. However,

Open Access 2026-10-22

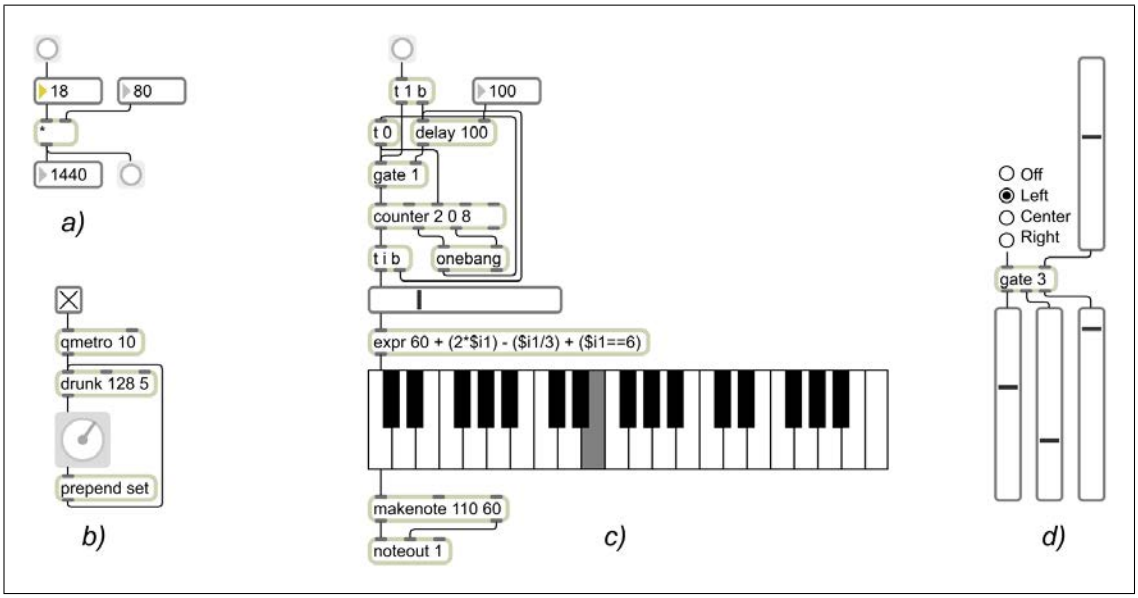


Figure 6.3: The `basics_subpatch`*, containing four tiny examples illustrating the basic behaviour of a few Max objects.*

as mentioned earlier, the object is only triggered when receiving input at its leftmost inlet. To prove this, try changing the number box connected to the right inlet, and then trigger the leftmost inlet by changing the number box connected to that inlet or by pressing the button above it.

Example *b)* demonstrates simple timing and random operations. By turning the toggle on, the `qmetro` object will send out a bang from its outlet every 10 milliseconds, which triggers a random number generator. These random numbers are fed into a dial, which responds to mouse interaction as well.

Example *c)* shows a more complex flow, which can be triggered by a single click on the button on the top. The dataflow would play a C major scale on MIDI. The horizontal slider and the keyboard slider show the current state of the system, but can also be altered by mouse interaction. The speed of the scale can be altered by changing the number box attached to the `delay` object: the value is the time elapsing between two subsequent notes of the scale, in milliseconds.

Example *d)* depicts a very basic message routing setup. The data flow, originating from the upper right slider, may be routed to three different sliders (or can also be turned off) by selecting the proper option from the radio button group.

6.3.2 Embedding

The encapsulation of patchers is demonstrated by the '`embedding`' subpatch, depicted in Figure 6.4.

The subpatcher contains two abstractions (`dikom.el.max.zmap1_` and `dikom.el.max.zmap2`), two subpatches (both called `patcher_zmap`) and a `bpatcher_` containing the abstraction `dikom.el.max.zmap1`. All of these abstractions and subpatchers implement exactly the same functionality as the built-in external called `zmap` (also included in the patch). This object will scale linearly an incoming number. This number is expected

Open Access 2026-10-22

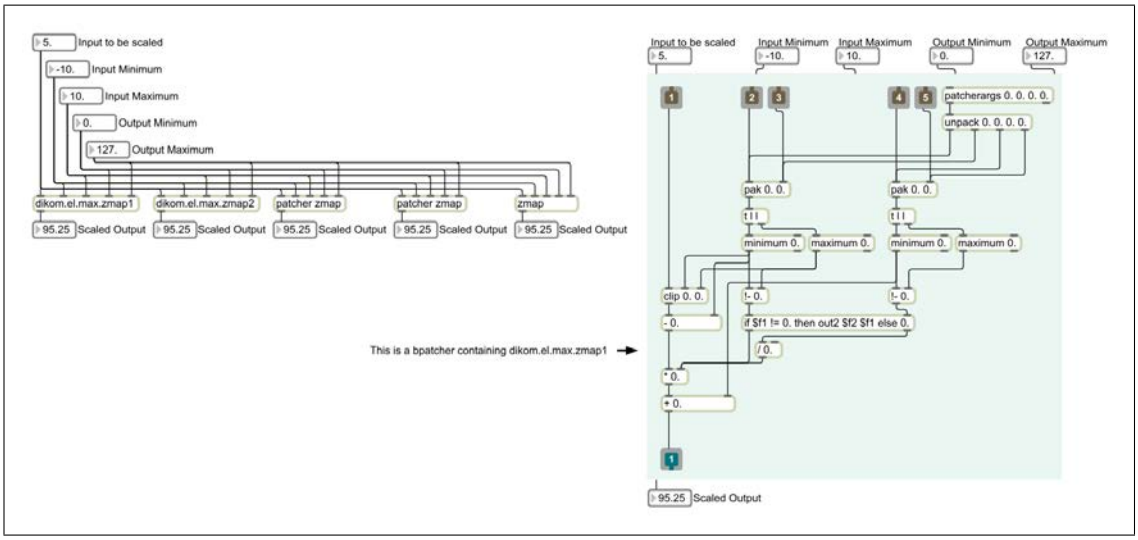


Figure 6.4: The `embedding_subpatch`*, showing different implementations of the same linear scaling method.*

to fall within the domain defined by '*Input Minimum*' and '*Input Maximum*'; in turn, the scaled value will fall in the range defined by '*Output Minimum*' and '*Output Maximum*'.
By changing the settings and the incoming numbers, you can prove that all objects in this subpatch exhibit the same behaviour. However, by inspecting the patches and abstractions, you may realise that they are quite different.
You can see, by double-clicking on the abstractions, that the `dikom.el.max.zmap1` abstraction looks identical to the one in the `bpatcher`. As stated previously, this is the expected behaviour: multiple instances of the same abstractions are always identical, and they only differ in the actual states of their variables. By contrast, the two subpatches differ from each other, although their names are identical.
If you look carefully at the abstraction `dikom.el.max.zmap1`, you will realise that it contains two identical sections. In such situations, it is desirable to create another abstraction for this recurring task: this ensures that, if you need to modify the behaviour of these code sections, you need only modify a single abstraction instead of having to modify the identical sections of code multiple times. `dikom.el.max.zmap2` is an abstraction which was designed keeping this in mind: after opening the abstraction by double-clicking on it, you will see that the identical sections of `dikom.el.max.zmap1` were encapsulated into an abstraction called `dikom.el.max.abstraction`. As is demonstrated by this very simple example, the level of embedding patchers and abstractions is not limited by Max in any form.

6.3.3 Signal Routing

The '`routing`' subpatch (see Figure 6.5) presents a few methods of routing signals. It contains a `selector~` object which will choose between four available sound sources and a `gate~` which will route the selected signal to either the left or the right loudspeaker. The four sources are, from left to right: the input from the first channel of the sound card (to which one may connect an instrument or a microphone – hence its name Analog to Digital Converter, or ADC), a sound file playback and two synthetic sources. The latter two will emit short sounds when the buttons above the respective subpatches (called `source1` and

Open Access 2026-10-22

source2) are pressed. To activate the file playback, you should press the button above the file_subpatch, which will open a dialog, allowing the selection of an arbitrary sound file.

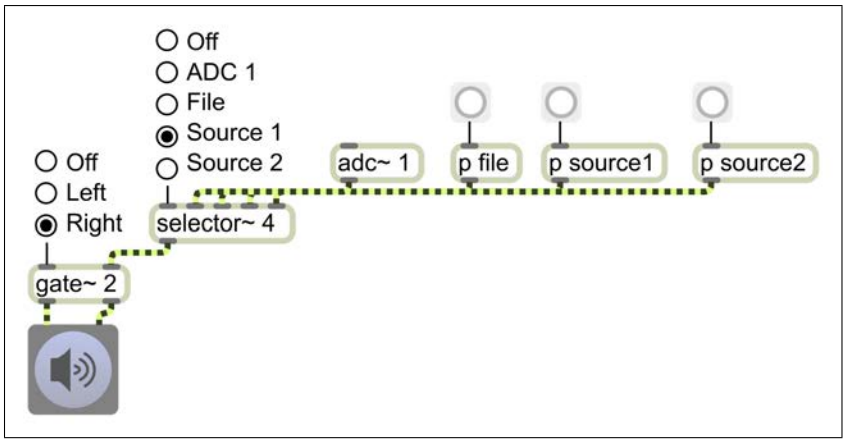


Figure 6.5: The routing_subpatch, containing an Analog-to-Digital-Converter (ADC) input, a file player and two synthetic sounds that may be routed to the left and right loudspeakers.

Note that by default, audio processing in Max is turned off. To turn it on, you should press the loudspeaker icon that you can find in both subpatches. This is also the case for any other Max patch dealing with real-time sound processing.

6.3.4 Data Mapping

The subpatch 'mapping' (see Figure 6.6) shows a few methods of mapping data in different ways.

The upper left part of the patch will react to input coming from a MIDI keyboard. If you don't have a MIDI keyboard, you can emulate it by pressing the keys on the piano slider with the mouse. The example is very simple: it may invert the pitch and/or the velocity of the incoming MIDI data and output the resulting new notes to the internal MIDI synthesizer of the computer. By switching the graphics toggles on the left (right), you may invert the MIDI pitch (velocity) of the incoming data flow.

The upper right part demonstrates simple linear scaling. The slider on the top outputs values between 0 and 127 (the usual range of MIDI control data). The first zmap_will scale these numbers linearly to a range between -1 and 1, and the second zmap_will scale the numbers back to the original 0–127 range. The display on the right shows the scaled values as if they were describing an audio signal.

At the bottom of the patch, a more sophisticated mapping is demonstrated. MIDI data arriving from a MIDI keyboard are displayed on the small piano slider (which can be operated using the mouse as well). The incoming MIDI pitch values are first mapped to their equivalent frequencies by the mtof_(MIDI-to-frequency) object, and are fed into an oscillator programmed inside the subpatch called 'source'. The result is filtered and amplified before being sent to the loudspeakers. The cutoff frequency of the filter and the gain of the amplifier can be set by the respective dials in the patch. However, the dial_object outputs numbers between 0 and 127, which is not a good range for neither cutoff frequency nor gain! Therefore, the values are scaled to their respective ranges. Particularly, the permitted cutoff

Open Access 2026-10-22

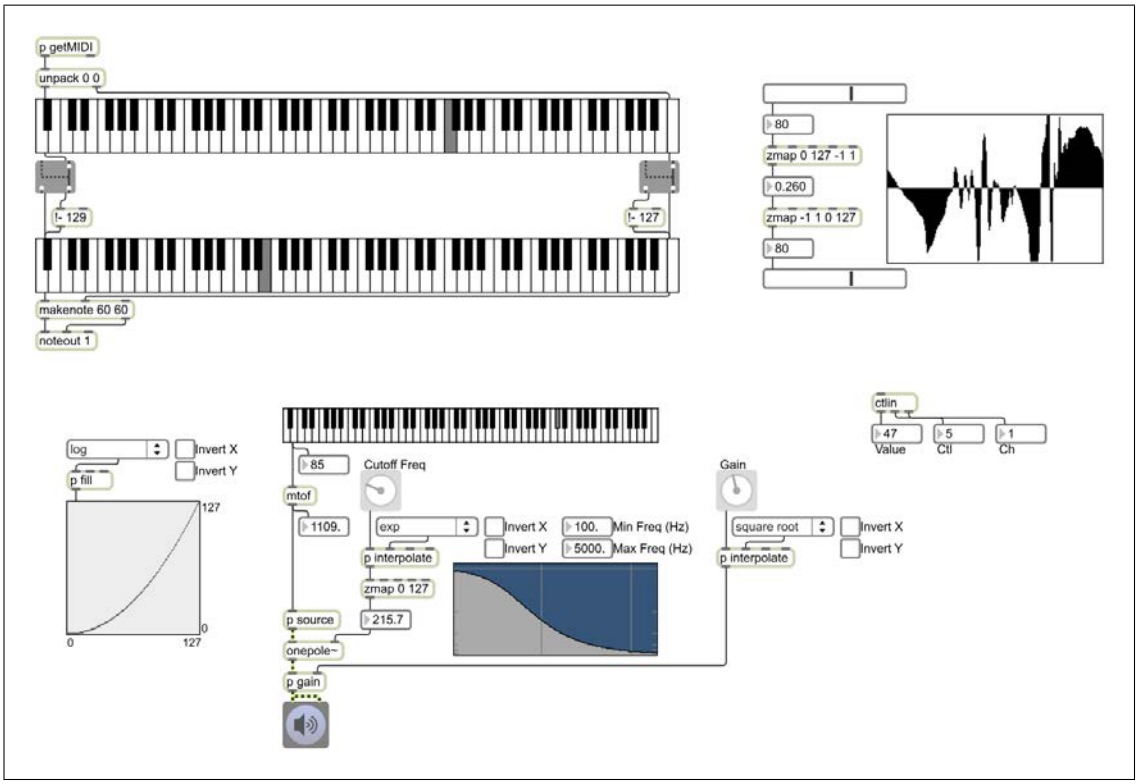


Figure 6.6: The `mapping_subpatch`*, showing several examples of linear and non-linear scalings.*

frequency range can be explicitly set with the number boxes '`Min_Freq`' and '`Max_Freq`' (with default values of 100 Hz and 5 kHz). Note that both values are mapped in a non-linear way. To choose the exact non-linear mapping shape, one may select a basic shape from the two drop-down lists, which one may invert as well, either in respect to their X or their Y axes. There is a small section in the lower left part of the subpatch which displays these mapping shapes graphically.
Don't forget to turn on the audio processing by activating the loudspeaker icon on the bottom of the subpatch if you didn't already do so.

6.3.5 Templates

The last subpatch, called `templates` (depicted in Figure 6.7), presents a few common generic solutions, which will be used during the rest of this syllabus. These include a generic sound source and sound output as well as a panel that displays incoming MIDI data and a button that assigns the incoming MIDI CC number with a specific GUI element. A simple keyboard control is also presented.
Both the sound input and output have a button to turn on or off sound processing and another one to access the settings of the sound card. The generic input has several options:

ADC: External source (either a microphone or a line input).

File: A dropdown list offers different sound files, which are played as infinite loops. It is also possible to load any sound file from the hard disc.

Open Access 2026-10-22

Sine: A pure sine wave whose frequency can be set.

Noise: White or pink noise.

Click: A short impulse is produced whenever the gray button is pressed.

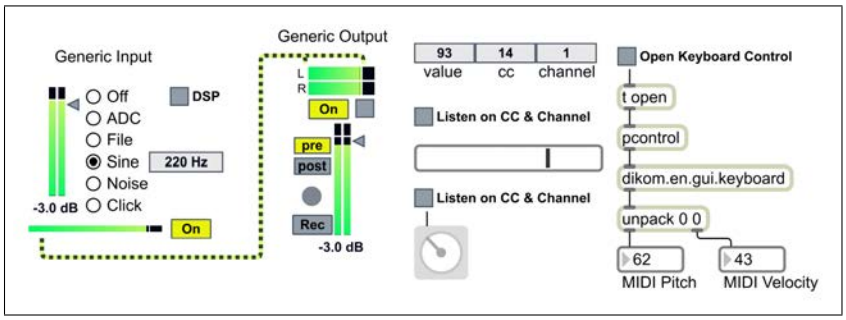


Figure 6.7: The `templates_subpatch`*, showing the different templates used through these chapters.*

The vertical sound level meter shows the loudness of the source *before* the amplifier, while the horizontal level meter shows the loudness *after* the amplifier (therefore, the real loudness of the source). By contrast, the horizontal level meter of the output shows the sound level as it arrives to the output, while the vertical level meter indicates the actual loudness of the outgoing signal. The output panel also has limited sound recording capabilities. The '*pre/post*' buttons let the user choose whether the sound should be captured before or after amplification; by pressing the button above the '*rec*' button, one may set the name and type of the file to create, and by pressing '*rec*', one may start recording. To stop recording, press '*rec*' again.

To access the sound card's settings (also called '*DSP Settings*' in the Max terminology), you need to press either the '*DSP*' button in the generic input or the button next to the *On/Off* controller of the generic output. The DSP window allows you to set such parameters as the sample rate, the audio driver, the vector size or the I/O mappings of the sound card. Please refer to the Max documentation and your sound card's manufacturer to learn about the meaning of these.

The MIDI panel shows incoming control values and CC numbers of MIDI control data as well as the channel from which the messages originate. It is possible to assign a certain MIDI CC with a GUI element by pressing the '*Listen on CC & Channel*' button linked to the specific GUI element.

By pressing the '*Open Keyboard Control*' button, a separate window (depicted in Figure 6.8) will open containing a `kslider`. By clicking on the keys with the mouse, you can imitate the behaviour of a MIDI keyboard. To clear these notes, you can either click on the respective keys again or press the button near '*Clear Hanging Notes*'. You can also convert your computer keyboard into a MIDI keyboard by clicking on '*Enable Computer Keyboard*'. In this case, the four rows of the computer keyboard will act as piano keys in four subsequent octaves, starting from the offset value entered in the '*MIDI Offset*' number box. Of course, you can use a real MIDI keyboard as well to create MIDI note events: to enable this, you need to select your keyboard from the drop-down menu near '*Select MIDI Input*'. In any of these three cases, you will see the generated MIDI note events in the two number boxes of the `templates_subpatch`.

Open Access 2026-10-22

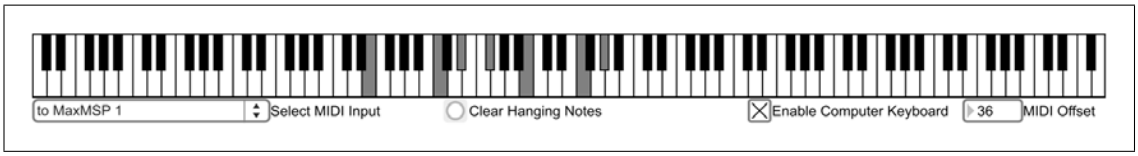


Figure 6.8: Generic keyboard input.

6.4 Exercises

1. Explain how exactly the Example *c)* of the `basics` subpatch works! Refer to the online documentation of the involved objects. Answer the following questions:

   - Why does playback stop after reaching the bottom of the scale?
   - Why does the direction of the playback change? How is the top of the scale defined?
   - Given that `$i1` stands for the incoming (integer) number, `==` tests for equality and `/` is integer division (meaning that the fractional part of the result is dropped), what does `expr 60 + (2*$i1) - ($i1/3) + ($i1==6)` do?

2. Explore the templates in `LEApp_06_01`. Listen to each sound source of the generic audio input template! Create recordings with the generic audio output template (using both the '*pre*' and '*post*' options)! Link different MIDI CC values to the slider in the test patch. Explore the possibilities of the generic keyboard control window. It is very important that you familiarise yourself with these tools, as they are broadly used across the rest of the syllabus.

3. Observe the non-linear mappings of the `mapping` subpatch! Which of the implemented interpolation methods fits well with gain? Which on suits best frequency values?

4. Open the `gain` subpatch in the `mapping` subpatch and explain how it works. Note that the `append` object appends its argument to any incoming message and the `line~` object creates a ramping signal that will reach the amplitude value set by the arriving number. If a list of two numbers arrives, the second item of the list sets the duration of the ramp, in milliseconds. Can you tell why we multiply the incoming gain values by 0.007874?

Open Access 2026-10-22

Chapter 7

OSC — An Advanced Protocol in Real-Time Communication

In the early days of electronic music, devices were assembled as stand-alone instruments. Although a limited amount of modularity was introduced by the first synthesizers, matching the tools manufactured by different brands was far from straightforward. This issue grew with the appearance of polyphonic synthesizers and digital interfaces, leading to the development of the first generic standard of digital electronic music.
Musical Instrument Digital Interface (MIDI), introduced in 1983, describes a protocol (the 'language' used by the devices), a digital interface and a set of connectors, allowing generic communication between electronic music tools implementing the standard. The protocol consists of different messages, followed by its parameters. Generally, these parameters are integers in the range 0–127.
Despite being a very important milestone in the development of electroacoustic music, MIDI has several limitations. Although the resolution of the parameters is adequate for a rough representation, it is unable to catch subtle gestures of performance. Besides, because of the hard-coded range, the definition of proper mappings is not always straightforward.
As devices got faster and exhibited more advanced features, these limitations called for a new standard. The development of the Internet brought a new general-purpose network communication protocol (the Ethernet), and the appearance of USB which, ultimately, replaced most other serial protocols, assisted these demands. *Open Sound Control* (OSC), introduced in 2002, offers a solution much more suitable for modern devices.

7.1 Theoretical Background

7.1.1 The OSC Protocol

Like MIDI, OSC is based on single messages transmitted between devices. However, *OSC Messages* have a more flexible, 'musician-readable' format. These are the different sections of an OSC Message, following their ordering within the message:

Address. Identifies the targeted device. It has an URL-like style, where wildcards are allowed. By using wildcards, a single message may be received by multiple devices. For example, if the OSC-addresses of two synthesizers in a network are `/syn/a` and `/syn/b`, then the wildcard `/syn/*` would address both of these devices.

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

Type Tag. Describes the types of the arguments. There are various data types allowed by the protocol. The standard ones are integers, floating-point numbers, strings and arbitrary binary data (e.g. a pre-recorded buffer, an image etc).

Arguments. The actual data described by the type tags.

These messages may be nested, forming a hierarchy. A nested collection of OSC Messages is called an *OSC Bundle*. In addition, these collections always contain a high-precision Time Tag, allowing for an advanced way of scheduling events or the proper ordering of simultaneously received data.
One of the biggest advantages of OSC is that the protocol is independent of the 'physical layer': it does not specify the physical setup of the connected devices. Because of this, OSC is useful not only when one needs to connect devices remotely (e.g. in two different cities), but also it can serve as the bridge between two different software running on the same computer.

7.1.2 IP-based Networks

Although OSC is not restricted to the network setup, OSC-compliant tools communicate with each other over an IP ('*Internet Protocol*') network in most cases. The most well-known IP network is the Internet, however, institutional and domestic internal networks ('intranets' or 'local area networks') are normally based on IP technology as well.
In an IP network, each device (also called *host*) has a unique address, similar to the unique calling number of each device in a telephone network. An *IP address* currently[1] consists of four numbers in the range 0–255. It is also possible to assign a unique name, called *domain name*, to a particular IP address (e.g. `www.google.com`). By convention, the `localhost` domain name — standing for the `127.0.0.1` address — is a self-reference to the host itself. For example, a computer software talking to the `localhost` is actually communicating with other software running on the same computer.
Since a single host may run several modules simultaneously (just think of a computer running dozens of applications at the same time), the IP address is not enough to route messages from different sources after they reach the host. How would a computer be able to decide whether a newly-arrived network package is part of a web page (and thus, should be processed by the web browser) or is an OSC message that should reach, for example, Max? Therefore, each 'channel of communication' (also called *port*) has a unique identifier, an integer in the range 0–65 535. Processes running on the same computer can use this information to find out who the actual recipient of an incoming data packet is. If IP addresses are analogous to telephone numbers, then port numbers function like phone extensions.

[1] This standard, called IPv4, is currently being actively replaced by IPv6, where an address consists of eight numbers in the range 0–65 535. Get used to it the sooner you can!

Open Access 2026-10-22

7.2 Examples

LEApp_07_01_is downloadable for Windows and Mac OS X platforms using the following links: http://www.typotex.hu/upload/Szigetvari_interaktivok/_SzigetvariAndrea_LiveElectronics_ENG/07_WIN.zip, http://www.typotex.hu/_upload/Szigetvari_interaktivok/SzigetvariAndrea_LiveElectronics_ENG/07_OSX.zip_

To find out the IP addresses of your computer, open the LEApp_07_01_application (see Figure 7.1). The dropdown list on the left shows your network interfaces (i.e. the different networks to which your computer belongs) and the dropdown list on the right shows the address of your computer within the chosen network.

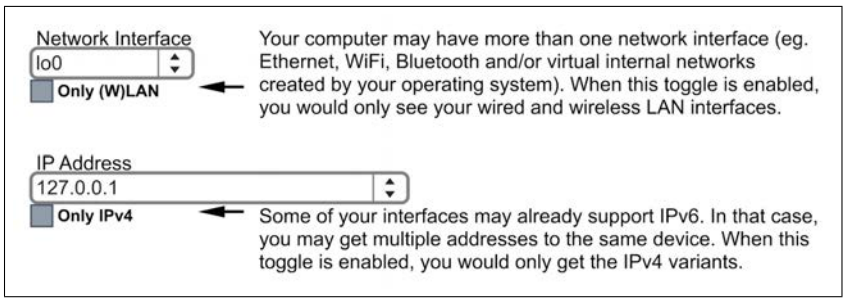


Figure 7.1: Find out the IP addresses of your network interfaces.

LEApp_07_02_contains a synthesizer that can be controlled via OSC (depicted in Figure 7.2). By default, the full OSC address of the synth is /dikom/syn/ID, where ID_stands for a randomly chosen ID in the range 0–99 (a value is automatically picked when the patch is launched). Both the ID and the name prefix can be adjusted manually. In addition, a port number (7887_by default) is selected. Use the IP address of your machine (which you can find by launching LEApp_07_01) and this port to 'call' your synth. You may run multiple instances of this patch simultaneously, sharing the same port. Just make sure that the Synth IDs of the patches are different.
The synth itself contains three different monophonic instruments, called Voice1, Voice2 and Voice3. To understand how they work, open the LEApp_07_03_application, a simple interface to control the OSC synth (see Figure 7.3)! On the lower right of this, the OSC Address and Synth ID fields indicate the address of the synth to which we wish to talk, although the '*To All Synths*' option would send our messages to all synths whose name begins with the OSC prefix in the *OSC Address* field. Choose the ID of a running synth, and make sure that the '*Enable Traffic*' option is checked (if this setting is disabled, the control patch won't send anything). Now, if you move the '*Panning*' or '*Gain*' slider, the respective values of the synth patch should change accordingly. With the DSP buttons, you can remotely turn on and off the audio processing on the synth. Finally, the three coloured panels contain controls for the three instruments of the synth. Voice1 and Voice2 may be controlled with a MIDI keyboard or the piano slider. Since Voice1 produces a sustained tone, the synth itself contains an option to turn off the hanging notes if, for some reason, the note-off message is lost in the network. Although Voice3 can be controlled with a MIDI keyboard, MIDI faders and rotary dials are more appropriate controllers. The pitch range can be adjusted by the controller patch.

Open Access 2026-10-22

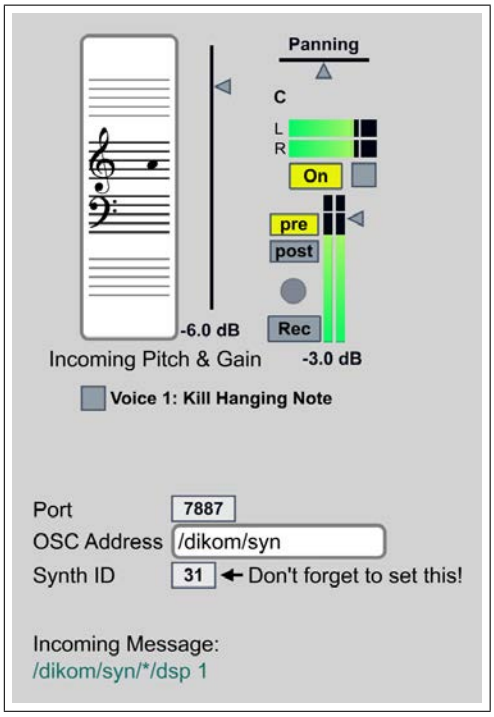


Figure 7.2: A synthesizer, consisting of three independent modules, which can be controlled via OSC.

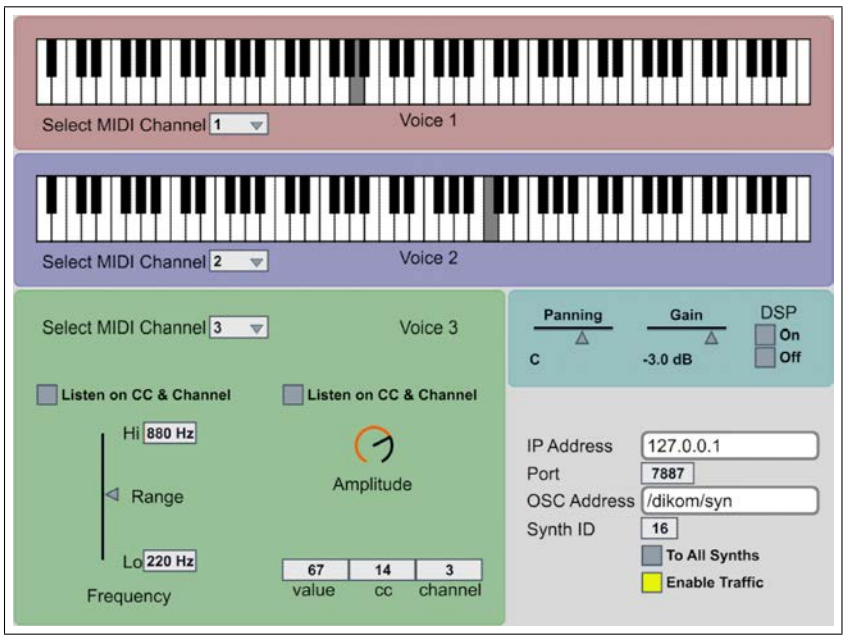


Figure 7.3: An OSC-based synthesizer contol interface.

7.3 Exercises

1. Open the control program and launch at least three instances of the synth. Set the *Synth ID* of the control program to control the first synth and play a few notes. Repeat this process with every instance of the synth (change the ID; play notes). Turn on the *To*

Open Access 2026-10-22

All Synth toggle and play some notes on all synths.

2. If you have access to a second computer, try running the synth on the one and the control program on the other! As long as the two computers are on the same network (and, supposing that you do not have a firewall running) you may be able to do this by filling in the IP Address of the computer running the synth in the appropriate field of the control patch running on the other computer. Play a few notes to test the connection!

3. In the bottom of the synth patch, you will see the incoming messages. By controlling the synth with the control patch, you may easily be able to learn the OSC commands that the synth expects. If you have any OSC-capable device at home (including non-free smartphone apps, like *TouchOSC* or *c74*), use this information to set up your device to control (at least, partially) the synth! You'll be able to do this by reading the documentation of your OSC-capable device. Play a few notes to test the connection!

4. It is also possible to send OSC messages over the Internet. However, unless the receiver host is directly connected to the Internet, you need to configure every network device (e.g. routers etc) standing between the host and the Internet in an appropriate way, which is not covered by this Chapter[2]. Try running the synth and the control program on two different computers which are connected over the Internet; play a few notes to test the connection!

[2]Basically, one needs to open every firewall blocking the chosen port and needs to set up the proper port forwardings. If you know how to do this, and you wish to experiment with our example patches, you might wish to know that our example patches use UDP exclusively — therefore, you won't need to open any TCP ports on your network.

Open Access 2026-10-22

Chapter 8

Pitch, Spectrum and Onset Detection

As explained in Chapter 2, interactive systems are those tools which react to their musical environment. This reaction has two separate levels. Firstly, an 'intelligent' musical device needs to 'understand' our performance: it requires tools able to capture our performance and translate it to musically meaningful parameters by the means of analysis. Secondly, the 'intelligence' of the machine can be defined by algorithms that build on these extracted parameters. In this Chapter, we concentrate on the first of these two steps.

8.1 Theoretical Background

There are no general criteria for a parameter to be 'musically meaningful'. In fact, it is up to the composer's own taste, at least to some extent, to decide upon this question. We may classify them according to a few categories, though. *Low-level* parameters are the physical ones that can be deduced directly from the signal (e.g. its current pitch). These can be used to extract *high-level* parameters, which describe the sound in complex musical terms (e.g. the performed melody or rhythmic structure, or even the genre of the piece).

8.1.1 Signal and Control Data

Information can reach an interactive system in two different forms: either as audio signals captured by microphones or direct control data arriving from sensors.
Simple controllers — like buttons, sliders or dials — give error-free, abstract control streams, which are normally very straightforward to interpret. These sources offer robust grounds to deduce higher level information about the current performance (e.g. a MIDI-based pitch sequence can easily be used for identifying recurrent motifs or scales). Complex sensors — like motion trackers or air pressure meters — set a more difficult task, as the (usually) continuous stream that they generate is not completely free from detection errors. However, since (motion) sensors operate at a relatively low rate and the tracked gestures are simple and slow (compared to the complexity and bandwidth of audio information), elementary error correction and gesture recognition algorithms may be used to manage these cases.
On the contrary, audio signal detection can become a very complex task. An audio signal — specially, if recorded live during a performance — contains a lot of noise, to say the least. Noise may not only arise from the micro-fluctuations that the performer(s) introduce into their music; any extraneous sounds in the concert hall will be recorded, often confusing the

Open Access 2026-10-22

very delicate detection algorithms. And even if there was no noise, the amount of information present in an audio signal makes any 'intelligent' recognition of pitches, durations, timbre etc. quite hard.

8.1.2 Detecting Low-Level Parameters

The most important musical parameters describing the temporal and timbral properties of a sound are *pitch*, *onset* (starting time), *duration*, *loudness*, and *spectral structure* (the structure of its partials), the latter equating to timbre.
There are complex tools which detect pitches in a polyphonic environment. However, such tools are still the subject of research. Current pitch and onset detectors work best with monophonic, pitched instruments (e.g. the singing voice). However, detecting the onsets of new tones is not straightforward even for these sounds. In fact, there are many different tools for onset detection, depending on how their designers define an 'onset'. Pitched onset detectors concentrate on finding even subtle changes in the signal. For example, a pitched onset detector would trigger a new onset even if the incoming sound doesn't change in loudness, just in its pitch (i.e. it will trigger new onsets for each individual note in a legato passage). Percussive onset detectors, on the other hand, define new notes based on their attacks: they outperform pitched detectors in identifying non-pitched sounds. Also, the latency of percussive detectors is usually smaller than that of pitched ones.
Low-level timbre detectors will extract the instantaneous spectrum of the signal. The two most popular ways of displaying the acquired data are the *spectrogram* and the *sonogram*. The spectrogram updates the display constantly with the instantaneous spectrum, while the sonogram is a convenient way of representing both temporal and spectral properties of the sound. The horizontal axis of a sonogram represents the elapsed time, while the vertical axis corresponds to the frequency. The relative amplitudes of the peaks are represented by a 'topographic map', consisting of colour densities are linked to the different amplitude levels. Extracting meaningful timbral data from a spectrogram or a sonogram is a more complicated process than other low-level extraction procedures. The main reason for this is that timbre is a multidimensional attribute and there are practically no low-level parameters to describe it.

8.1.3 Detecting High-Level Parameters

The acquired low-level descriptors may serve as a basis for further analysis. At this point, it is mostly up to the musician what they consider a 'high-level' parameter. In what follows, we present a few broadly used options.

Pattern Matching

In many musical genres, pitches and durations may be organized into melodic and rhythmic patterns. Thus, it is quite natural to identify the relevant melodies and rhythms of a piece as high-level descriptors.
At first glance, identifying a pattern is a straightforward task. One has to watch the incoming low-level descriptors (e.g. the identified pitches of the signal) and recognize the pattern (e.g. the melody). However, a pattern is always abstract, whereas the live music is real, which

Open Access 2026-10-22

prevents an exact match. Moreover, low-level detectors always carry some amount of statistical error, making pattern-matching based on signal analysis quite a difficult task. In fact, this is still one of the current research topics of live electroacoustic music.
It is much easier to analyse a stream containing (partially) abstract information. If, instead of analysing the incoming signal itself, we rely on the analysis of controller data (e.g. originating from an instrument-like or an instrument-inspired controller, see Chapter 4), we can remove much of the statistical error inherent in any low-level signal-based detection process. For example, by basing a pattern matching process on the data arriving from a MIDI-based instrument controller, there is no need to determine the pitch and onset separately, as this information is already present in the control stream itself.

Spectral Analysis

Another interesting family of high-level parameters arise from the spectrum itself. As we mentioned, timbre is a multidimensional attribute of sound and the spectrum has a key role in defining most of these dimensions. Based on the spectrum (or the sonogram), it is possible to compute descriptors such as the *loudness*, *noisiness*, *brightness*, *roughness*, *sharpness*, *warmth* etc. Some of these are well-defined dimensions of timbre space, however, most of them are semi-subjective parameters which, in some cases, are very difficult to express quantitatively. Nevertheless, the different means of spectral analysis are perhaps the most popular analysis tools of interactive systems after pitch and onset detectors.

Automated Score Following

This type of software is only just emerging from the research labs and being added to the toolset of many musicians performing live electroacoustic music. Through continuous reference to a (digital) score created offline, a score follower analyses the performer's musical activity in order to determine the exact current location within the score where the music currently is. Advanced score followers are able to detect embellishments or complex improvised sections and jump right to the next note when the performer returns to the score. Usually, they are also capable of determining the current tempo (and its fluctuations) and other general 'parameters' of the performance. *Antescofo*, a score follower developed at IRCAM, is a popular example of this approach.

8.2 Examples

8.2.1 Analysis in Integra Live

The `le_08_01_integra.integra_` file is downloadable using the following link: `http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_08_01_integra.integra_`

The project `le_08_01_integra.integra` (see Figure 8.1) presents a small example demonstrating pitch, onset and envelope detection (the latter is the time-dependent loudness of the incoming signal). The pitch and the envelope of the incoming signal define the pitch and gain of a generated sine wave. An effect (overdrive, see Section 12.1.2) is applied to this

Open Access 2026-10-22

sine wave, when an onset is detected in the incoming signal. The project contains two different detectors, a pitched and a percussive one.

Figure 8.1: An oscillator controlled by low-level musical parameters extracted from an input signal.

The large dial in the project shows the detected pitch, in MIDI cent values. A small toggle below this dial allows switching between pitched and percussion onset detection. The detected levels are displayed at the top centre of the project. However, a detection limit can be set individually for both methods using the number fields below the respective displays. The horizontal sliders on the bottom left and right set the incoming and outgoing signal's amplification, respectively.

8.2.2 Analysis in Max

`LEApp_08_01_` is downloadable for Windows and Mac OS X platforms using the following links: `http://www.typotex.hu/upload/Szigetvari_interaktivok/_SzigetvariAndrea_LiveElectronics_ENG/08_WIN.zip`, `http://www.typotex.hu/_upload/Szigetvari_interaktivok/SzigetvariAndrea_LiveElectronics_ENG/08_OSX.zip_`

`LEApp_08_01_` shows the instantaneous spectrum of a signal (see Figure 8.2). The spectrum is displayed on the bottom of the screen. The upper and lower frequency limits can be set by the two number boxes at the respective ends of the spectrum. The two toggles set the scaling of the horizontal (frequency) and vertical (loudness) axes. The large black display in the patch is a sonogram. The update interval can be set using the number box on the top; however, it is possible to stop updating the sonogram by pressing the '*Freeze*' toggle.
`LEApp_08_02`, depicted in Figure 8.3, presents a few simple attributes of timbre that can be deduced from the instantaneous spectrum. At the bottom, a spectrogram and a sonogram is shown. Three descriptors (both their instantaneous values and a running history of the last values) are displayed on the top right corner of the screen. The displayed timbral dimensions are *loudness*, *noisiness* and *brightness* of the incoming sound (from the top to the bottom, respectively). The loudness gives us the average power of the signal in decibels; the noisiness expresses how harmonic or inharmonic the signal is (0% indicates fully harmonic sounds — e.g. a single sine wave — and 100% describes white noise); brightness shows the 'centre of mass' of the spectrum[1] (the higher this value, the brighter the sound).
`LEApp_08_03_` and `LEApp_08_04_` are two simple pattern matchers, depicted in Figures 8.5 and 8.4, respectively. Both are based on incoming MIDI pitch and velocity information.

[1] A weighted average of the frequencies, where each frequency has a weight that is proportional to its amplitude. Also called *spectral centroid*.

Open Access 2026-10-22

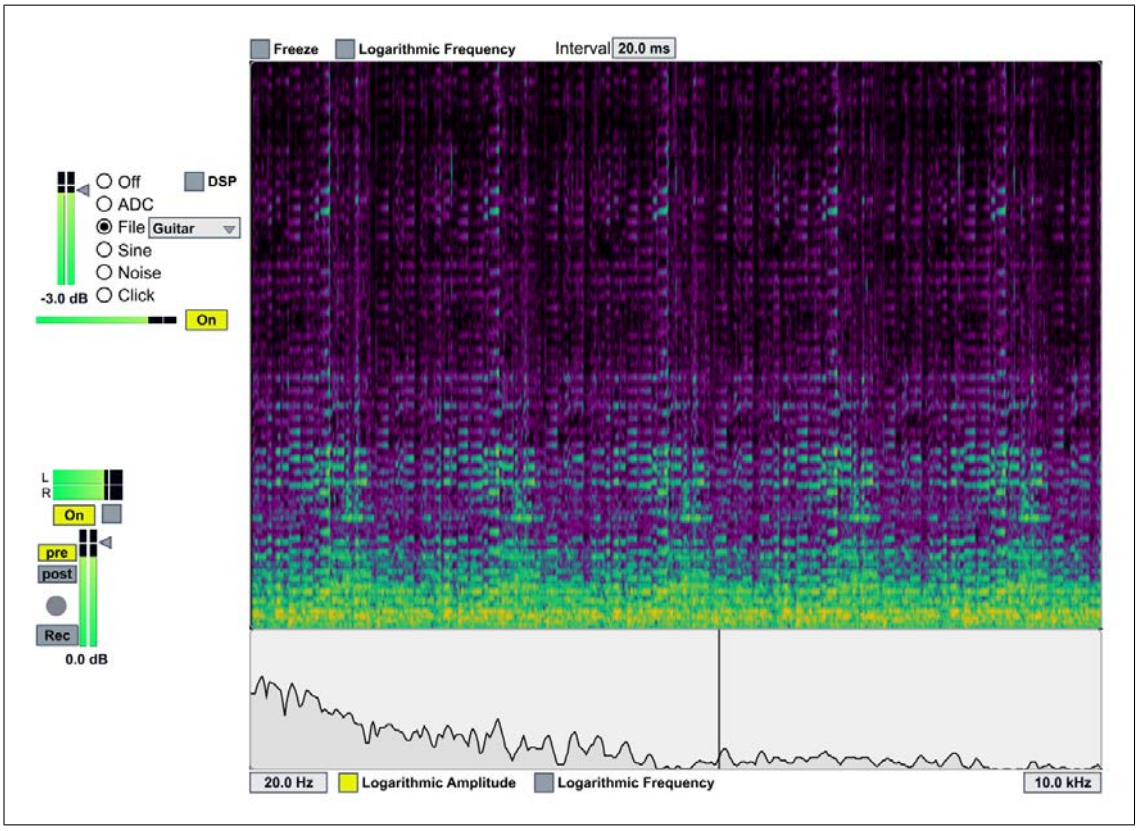


Figure 8.2: The spectrum and sonogram of an incoming signal.

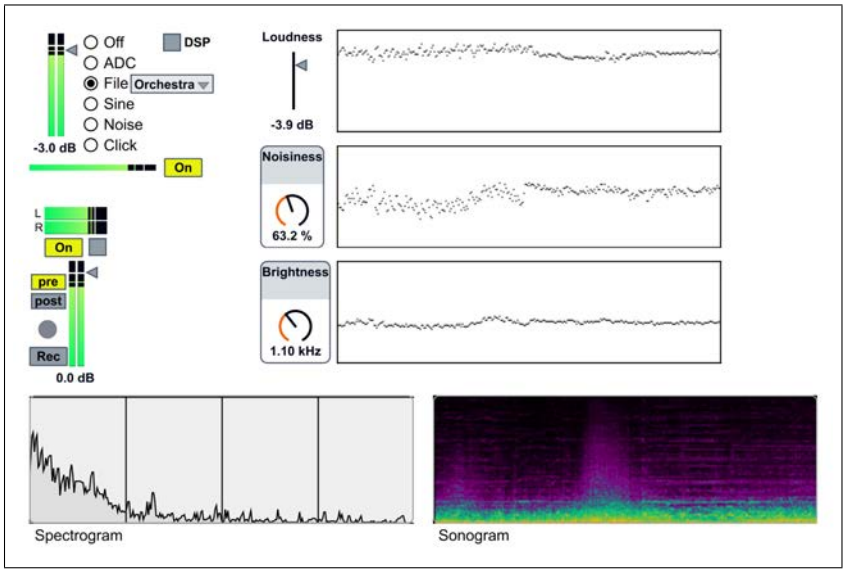


Figure 8.3: Simple spectral analysis tools in Max.

`LEApp_08_03` detects rhythmic patterns. The program listens to the MIDI keyboard selected with the '*Select MIDI Input*' menu. However, MIDI events can be imitated with the mouse (by clicking on the keys) as well as with the computer keyboard (for this, you need to turn '*Enable Computer Keyboard*' on and press any alphanumeric key on the keyboard).
The program can operate in two modes. In '*Record*' mode, you can record a new pattern:

Open Access 2026-10-22

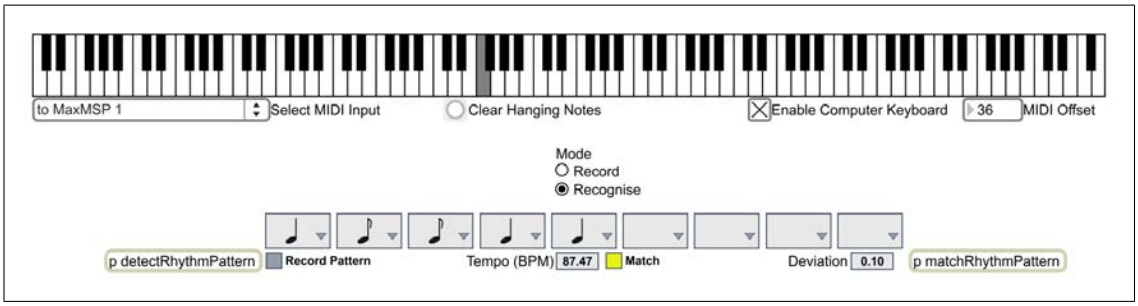


Figure 8.4: A MIDI-driven rhythmic pattern matcher in Max.

the sequence to recognise, containing up to 9 rhythmic values, can be set either with the dropdown lists (lists left blank are not taken into account) or by pressing the '*Record Pattern*' button (only while in recording mode). In the latter case, you just need to play the pattern; the application will automatically quantise it and copy it into the row.
In '*Recognise*' mode, the program computes the standard deviation between the durations of the last played notes and the pattern itself. Whenever the deviation is below the amount set by the number boxes at the bottom right, the '*Match*' button will be triggered and the current '*Tempo*' will be updated.
The pattern recording and matching algorithms are contained by the two subpatches, `[p_detectRhythmPattern]` and `[p_matchRhythmPattern]`, respectively.

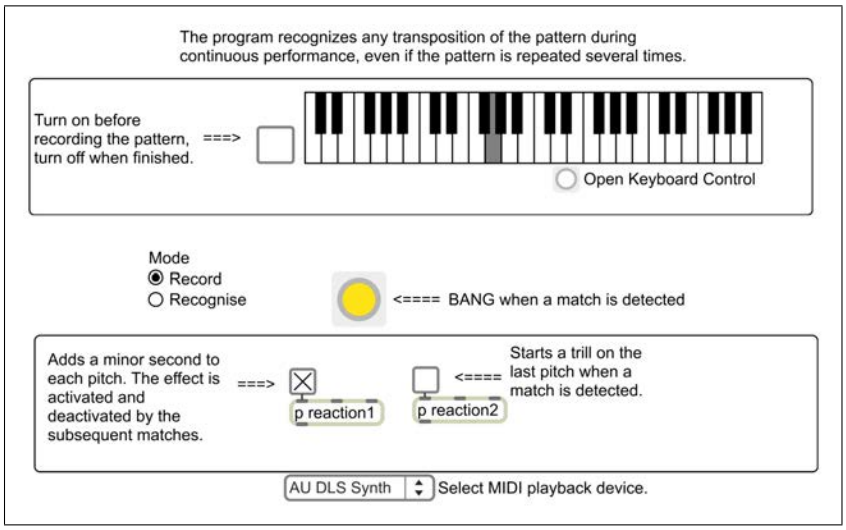


Figure 8.5: A MIDI-driven melodic pattern matcher in Max.

`LEApp_08_04` detects motifs. As with the previous example, the program is controlled with incoming MIDI notes; to open a MIDI control panel similar to the one contained by `LEApp_08_03`, press the '*Open Keyboard Control*' button.
This program also has two operational modes. In '*Record*' mode, you can record a new pattern by turning on the toggle at the top left of the window (and turning it off when you finished playing the pattern). In '*Recognise*' mode, this sequence will be recognised each time when you play it, even if you transpose it. In addition to recognising the pattern itself, this program reacts to the matches, too. Upon a successful detection, one of the 'reacting' modules will add a minor second to every subsequently played note until when the pattern

Open Access 2026-10-22

is recognised again, in which case the effect turns off automatically. The other 'reacting' module will play a trill on the last note of the motif. Both modules can be completely disabled with the toggles above them.

8.3 Exercises

1. Observe the difference between pitched and percussive onset detection with `le_08_01_integra.integra`! Invent a melody containing both legato passages (keeping dynamics constant) and long-held notes with abruptly changing dynamics (keeping the pitch constant). Adjust the detection limits so that when you sing this melody twice, using pitched the first time and percussive detection second, you can catch every pitch change in the first and every dynamic change in the second time!

2. Categorize every sound sample in `LEApp_08_02` according to the main characteristics of their loudness, noisiness and brightness! Pick three different samples and find sounds on your hard drive which have similar characteristics in terms of these three descriptors!

3. Record a simple rhythmic pattern with `LEApp_08_03`. Play a melody which contains this rhytmic sequence using different base durations (e.g. so that the pattern appears once in crotchets and the next time in quavers). Explore the efficiency of detection using different deviation limits between 0 and 1! Find a deviation limit which creates an acceptable balance, alowing space for artistic freedom in terms of tempo fluctuations (e.g. rallentando, accelerando etc.) without signaling too many 'false positives'.

4. Record a motif with `LEApp_08_04` and play a melody which contains this motif, possibly with many different transpositions. Turn the patches `[p_reaction1]` and `[p_reaction2]` on (only one at a time) and explore how they react to your performance. Turn on both, and improvise a melody on your recorded motif!

5. Only for advanced students: explore the subpatches `[p_detectRhythmPattern]` and `[p_matchRhythmPattern]` in `LEApp_08_03` and explain what exactly they do! Observe `[p_matchRhythmPattern]` first, as that is much easier to understand than the other one.

Open Access 2026-10-22

Chapter 9

Capturing Sounds, Soundfile Playback and Live Sampling. Looping, Transposition and Time Stretch

There are several situations in a live performance when recorded sounds may come in handy. Playing back sound samples has its own importance within the field of live music.
These samples, however, do not necessarily need to be recorded in advance. Recording a fragment in real-time in order to play it back later during the same performance is a quite prevalent technique, called *live sampling*. Although the method resembles *delay* (presented in Chapter 13), there is a very important difference: sampled sounds can be played back as many times as needed, and can be triggered at anytime; a delayed sound may not be available more than once, and its exact timing needs to be planned in advance. Besides, sampled sounds are much easier to process during a performance.
This chapter presents methods of recording and playing back sounds in live situations, and presents two important effects usually applied to these.

9.1 Theoretical Background

9.1.1 Methods of Sampling and Playback

Sampling is based on *recording*, *storing*, and *playing back* sounds. Due to its complex and very specific nature, we do not present the issues of sound recording in this Chapter.
There are two options to store samples, depending on whether we use (volatile) memory or a permanent storage device:

Recording sound into a temporal buffer. Buffers are blocks of memory dedicated to audio storage. The biggest benefit of using buffers is that they can be accessed instantaneously (unlike permanently stored samples, which need to be reloaded into the memory before using them). Their drawbacks are the limited amount of music that can be stored and accessed this way (due to the limited size of available memory in most devices) and the volatility of the stored information: if, for any reason, the program needs to be restarted during the performance[1], the recorded sounds would be lost.

[1] This happens quite often in live electronic music.

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

Recording sound to a permanent storage device (e.g. hard disk or tape). If longer recordings are needed, buffers may not have the capacity necessary for storing these. To record long samples of sound, and especially, if we need to play them back only once (or with very long delays, e.g. half an hour after we recorded it), recording to hard disk might be a better option. Besides, these samples would still be available after an accidental crash of the performing software. The drawback of permanent storage recording is the increased access time[2] and the fact that most real-time sample adjustment processes do not work well with samples stored outside of the memory.

Some methods mix these two approaches. For instance, it is possible to record samples directly on the hard drive and load them into the memory before playing them back. There are also some advanced storage concepts, like *circular buffering*, where the sound is being constantly recorded and, after a fixed amount of delay, is played back. They are also called *delay lines*; we discuss their properties in Chapter 13.
There are two main methods to play back sounds, no matter whether they were live-sampled or pre-recorded:

One-time playback. The playback starts at a given moment of the sample (usually, its beginning) and, once reaching its end, the sound stops.

Looping. The playback head jumps back to the beginning of the sample each time when it reaches its end and continues playing. This latter technique is extremely popular in electronic dance music, as loops are the best (and also, easiest) tools to create robust rhythmic patterns.

Freezing. A very short section of the original sample is looped (using granular synthesis methods, see Chapter 11).

9.1.2 Transposition and Time Stretch

One possible role of adjusting the transposition and duration of samples in a live scenario is to assure that, when mixing them with live-performed musical material, they stay in tune and in rhythm. This need is evident when using pre-recorded sound files; however, samples taken live may also need adjustments, either to synchronize well with the current material or to create interesting effects (e.g. deliberate de-tuning of the piece). Most professional samplers offer possibilities to make these tiny alterations in real-time, and in many cases, they contain analysers that find the required parameters in an automated way.
However, as both transposition and time stretching are essential musical operations, their roles are not limited to the one described above. Transpositions and time stretches with continuously changing parameters play a key role, for example, in musical articulation and phrasing.
The easiest way of changing the duration of a sample is by playing it back at different speeds. We use the term *speed factor* — the ratio of the playback and recording speeds — to describe

[2] Even with a hard disk, this can easily create a serious bottleneck for the performing software — with tape recorders, the situation is even worse, as jumping to the appropriate point of the tape might take several seconds.

Open Access 2026-10-22

this effect. However, changing the playback speed of a sample will change its pitch. The relation between the speed factor and the amount of transposition is

$$\text{Transposition [Cents]} = 1200 \ln_2 \text{Speed Factor} \tag{9.1}$$

By using more advanced techniques, it is possible to change the duration of a sample by stretching/compressing it (hence the name '*Time Stretch*') without altering the pitch and, conversely, modifying the pitch without altering the duration ('*Transposition*') is also achievable. However, the quality of these tools cannot be perfect: the bigger the deviation from the rule set out by Equation 9.1, the poorer results one should expect. This should not surprise us, though. When changing the pitch and duration of a sample independently, we also change the amount of audio information present in the sample. Whether we need to add or remove information, the machine has to make guesses as to either which data to throw away or to invent the missing information. As a workaround, most professional transposition and time stretching tools allow the user to 'help the machine's guess' by implementing several different algorithms optimised for individual scenarios, e.g. for percussive sounds, speech, monophonic (pitched) instruments etc.
An important device based entirely on real-time transposition is the *Harmoniser*, a tool that creates several transposed copies of an incoming sound and mixes them at its output (like additive synthesis, but performed on sampled sounds). As with additive synthesis, it is literally impossible to describe every possible type of alteration that this device can achieve. Here are a few of them, though:

- By using only harmonic transpositions, the sound can be made brighter (the effect is similar to *overdrive*, see Section 12.1.2).
- By adding subharmonic transpositions, we emphasize the bass, thus, make the sound darker. If these transpositions differ from octaves, we may also change the base pitch of the incoming sound.
- By using inharmonic transpositions, we can 'destabilize' the 'pitchedness' of the original signal. Depending on the actual settings, this may either result in minor instabilities, timbres with multiple base pitches or completely unpitched sounds.

9.2 Examples

9.2.1 Sampling in Integra Live

The `le_09_01_integra.integra_` file is downloadable using the following link: `http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_09_01_integra.integra_`
A simple sampler can be found in the project `le_09_01_integra.integra_`(see Figure 9.1). The right side of the display contains a small mixing board, where the gains of the input signal and the sampler's output can be set individually as well as their pannings. The big slider on the far right side controls the main loudness.
The sampler itself is on the left side. The controls at the top include buttons for recording ('sampling'), playing back, and stopping, together with the slider on the top left which indicates the current playback position. The *Load/Save File* buttons serve to load (save) files from (to) the hard drive. The big button next to the playback controls turns on or off looping.

Open Access 2026-10-22

The large dial at the bottom left sets the *speed factor* of the playback. If the small toggle at the bottom centre of the display is turned on, the transposition would be set accordingly to Equation 9.1 to avoid pitch shifting. Otherwise, transposition (expressed in cents) can be set using the big dial at the bottom centre.

Figure 9.1: A sampler with individual control over transposition and playback speed.

9.2.2 Sampling in Max

`LEApp_09_01_` is downloadable for Windows and Mac OS X platforms using the following links: `http://www.typotex.hu/upload/Szigetvari_interaktivok/_SzigetvariAndrea_LiveElectronics_ENG/09_WIN.zip`, `http://www.typotex.hu/_upload/Szigetvari_interaktivok/SzigetvariAndrea_LiveElectronics_ENG/09_OSX.zip_`

`LEApp_09_01_`(see Figure 9.2) demonstrates both the permanent and volatile approach of sampling and play-back. The block in the upper right corner will record a sample to the hard drive and, optionally, play it back. To record a sound, press the '*Open*' button above the '*Record*' toggle first to set the name and path of the file which will store the sample. Start and stop recording with the '*Record*' toggle. To play back the recorded (or any other sample), press the '*Open*' button above the '*Play*' toggle to choose the file containing the sound. Then, you can start and stop play-back with the '*Play*' toggle. The slider below this toggle lets you define the play-back region within the file — the in and out points. By enabling the '*Loop*' toggle, playback will jump and continue from the beginning of this region each time when the playhead reaches the end; otherwise, playback will simply stop at that point. The '*Speed*' slider influences the playback speed (thus, the transposition and time stretch of the sound). To get an unaltered sound, press the '*Reset*' button. This will set the slider to its default position.
Volatile (memory-based) sampling and playback is demonstrated in the bottom of this program. The left side controls recording, while playback options are on the right. The '*Size*' slider sets the full length of the storage buffer, whose content is displayed on both sides of the window.

Open Access 2026-10-22

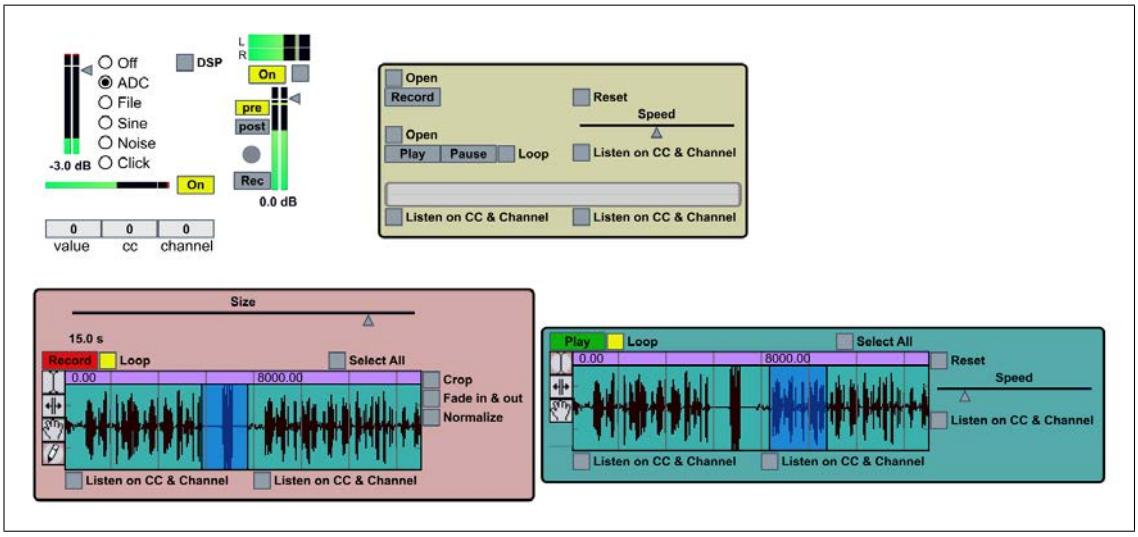


Figure 9.2: A simple sampler presenting both hard drive- and memory-based sampling techniques. The current setting gives an example of circular recording and looped play-back of the same buffer: while recording into the second half of the buffer, the first part of the same buffer is being played back.

To record into the buffer, select a region with the mouse (or press '*Select All*') from the waveform display on the left side. Only the selected part of the buffer will be overwritten; the rest is left untouched. There are two recording modes. Normally, the recording starts when you press the '*Record*' toggle, and once the recorder head reaches the end of the selected region, recording will finish. Conversely, by turning on the '*Loop*' toggle near the recording one, you can enable *circular buffering*, when the recording head jumps and continues from the beginning of the region each time when it reaches the end of it.

After recording, you have the possibility to normalize the contents of the buffer as well as create a fade in and a fade out (to better avoid clicks). After selecting a region of the buffer, you can delete the rest by pressing the '*Crop*' button.

The four icons on the left side of the waveform display allow you to create a selection with the mouse, move that selection, zoom in and out the waveform and edit the waveform by hand, from top to bottom respectively.

The playback engine on the bottom right side looks similar to the recording block. The region that needs to be played back can be specified with the mouse, and the playback mode (one-time play-back or looping) can be set with the appropriate toggle. Therefore, it is possible to use one region of a buffer as play-back material while recording to a different region of the same buffer simultaneously (this is the exact case presented in Figure 9.2. The speed and transposition of the playback can be manipulated in the same way as a hard drive-based player.

A 4-channel hard drive-based circular buffering solution is presented by `LEApp_09_02`. Each block has a recording toggle on the top left and a dropdown menu that lists all recorded samples during the current session. New files are created automatically[3] each time you start a new recording. You can control the recording process in a centralised way using the

[3]They are saved into the folder where `LEApp_09_02` resides. However, they are not deleted automatically when you quit the program — you have to take care of that manually.

Open Access 2026-10-22

big toggle on the top of the window. In this case the patch will automatically choose the recording channel.

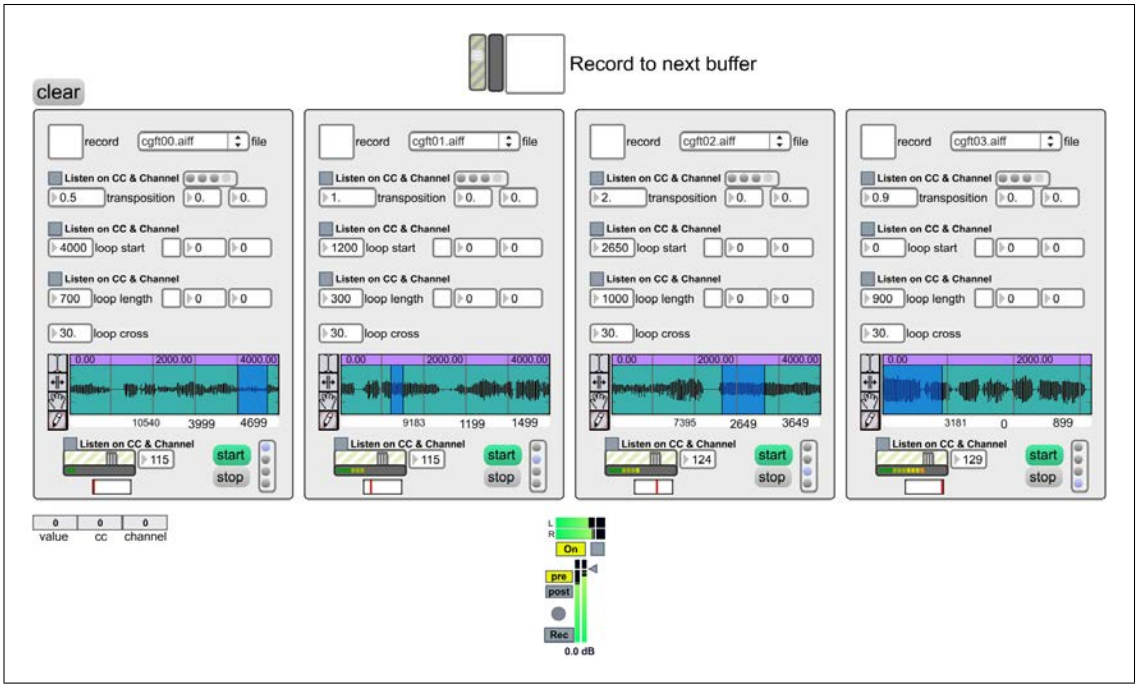


Figure 9.3: A 4-channel hard disk-based circular recording and play-back engine in Max. All channels use separate playback settings, including the possibility of turning them on and off individually. Sound files are created and saved automatically.

The samples are always looped during play-back. The looped region can either be set graphically with the mouse (using the waveform displays) or by setting the '*loop start*' and '*loop length*' parameters. It is possible to control these values by MIDI CC, too. The toggles and number boxes near these fields also allow randomisation: when the toggle is enabled, a random value is generated based on the tempo and range values set by the number boxes within the same row.
The speed factor of the playback (thus, the transposition) is set by the number box called '*transposition*', also controllable through MIDI CC. Here, the two additional number boxes define the minimum and maximum speed factors when controlled by MIDI.
The program `LEApp_09_03` (see Figure 9.4) contains a harmonizer with 8 channels, each one allowing a range of ± 36 semitones (3 octaves) for the transpositions. The different stored settings show examples for all scenarios described in Section 9.1.2.

9.3 Exercises

1. Compare the sampling tools of this Chapter (`le_09_01_integra.integra`, `LEApp_09_01` and `LEApp_09_02`)! Record a short accompaniment motif (e.g. sing the bass line of a standard cadence) and play it back in a loop. Improvise different motifs over the recorded accompaniment. Change the speed and transposition of the loop as needed. Change the looped region if needed. Which of the three tools is most suitable for speed and transposition adjustment? Which is most efficient in changing

Open Access 2026-10-22

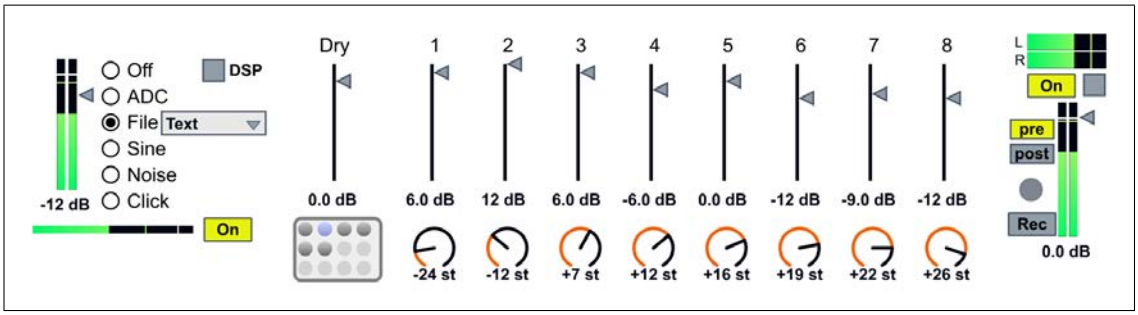


Figure 9.4: A harmoniser in use. The current setting, adding both subharmonic and harmonic components, adds brightness while transposes down the input signal's base pitch by several octaves.

the loop regions? Which has the best tempo syncing possibilities? Find strategies to stay in sync with the loops.

2. Improvise with `LEApp_09_02` in two ways at least! First, record four different samples and create music using these loops. Feel free to use transpositions and different loop regions (thus, different looping times). Second, record a single musical sample and load this into all the four players. Create music with loops derived from different regions and transpositions of this same sample.

3. Observe how the circular buffer of `LEApp_09_01` works! Choose a buffer size of a few seconds and start the recording in circular mode (i.e. by enabling '*Loop*' first). Shortly after, start the play-back of the same buffer (also in looped mode). Observe the result while talking or singing into the microphone. Experiment with changing the play-back region! What happens if the region of recording and the region of the playback doesn't contain each other? What happens when the recording region contains the play-back region entirely? What if the two regions have only a small intersection?

4. Use the same buffer for creating a multiple-step recording with `LEApp_09_01`! Choose a buffer size of approximately 10 seconds and record a long, steady sound. Eliminate any unnecessary clicks by clipping the recorded sound. Normalize your result! Now, set the play-back region to the first few seconds of the buffer while setting the recording region to the last few seconds. Start a looped play-back and a circular recording. Experiment with the input and output gains until you can see that the sound arriving from the loudspeaker is recorded with (approximately) the same amplitude as the original signal. Now sing another long, steady sound, creating an interval with the original one (which is now being played on the loudspeakers). This way you can mix your 'old voice' with the new one. Repeat this process a few times to create interesting chords. You may also experiment with transposing the buffer during play-back to create chords with large intervals.

5. You can freeze a sound by choosing a very short loop interval. In this case, you will hear a static timbre which is derived from the actual timbre of the original sample. Observe how it is possible to freeze sounds with every different tool presented in this chapter. Until which point does the cropping and normalizing utility of `LEApp_09_01_` enhace the creation of frozen sounds (is there a point after which the further cropping

Open Access 2026-10-22

of the signal destroys the timbre drastically)? How does the fade in & out option of this same program affect the frozen timbre? What is the main problem with all freezing possibilities offered by these programs? How could you overcome that problem? See Chapter 11 for more details about freezing the timbre.

6. Try each setting of `LEApp_09_03`and explain their effects! After listening to each preset with at least three different sources, set up different chords (e.g. major, minor, diminished seventh etc.) and observe the result if you talk or sing into the microphone. How does the harmoniser affect percussive sounds? What effect does the tool have in a polyphonic environment?

Open Access 2026-10-22

Chapter 10

Traditional Modulation Techniques

We use the term 'modulation' to describe processes that directly alter the physical properties of a signal. As seen in Chapter 8, the most important low-level descriptors of a sound are its pitch, loudness, timbre, duration and onset. Of these, the loudness and the pitch are more-or-less directly connected to the most important physical attributes of the signal: the amplitude and the frequency. As a consequence, these two properties of the sound are easiest to modulate; this Chapter presents amplitude and frequency modulation.

10.1 Theoretical Background

10.1.1 Amplitude and Ring Modulation

Ring modulation[1] occurs when two signals are multiplied. One of them is normally a sinusoid with a fixed or slowly changing frequency, and the other is an arbitrary incoming signal. Where the incoming signal is another sinusoid, the result would be the sum of two sinusoids, comprising the sum and difference of the frequencies of the original two signals, as depicted in Figure 10.1a. When the incoming signal arrives from a more complex source, ring modulation will 'duplicate and shift' it, creating a 'down-shifted' and 'up-shifted' copy of the incoming sound. However, if the frequency of the sinusoid is small enough (in the range of approx. 0–10 Hz), the effect will be perceived as a tremolo.

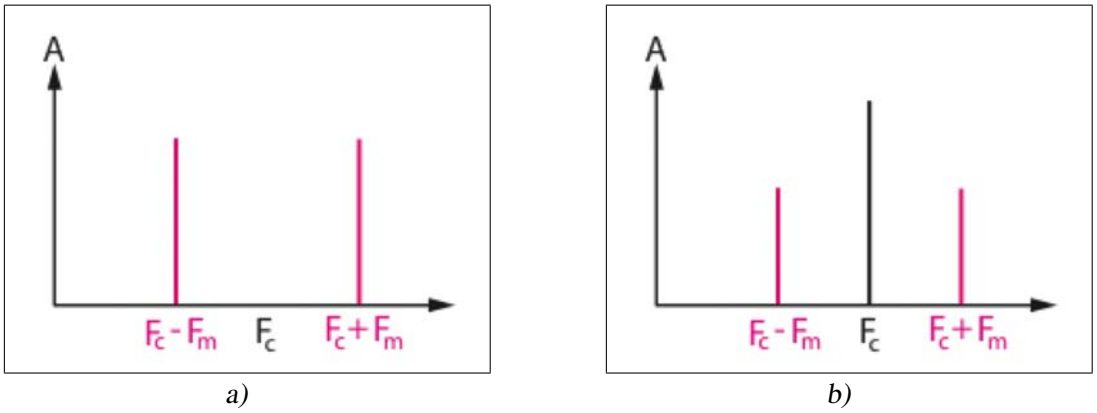
By mixing the ring-modulated signal with the original, we arrive at the more general *amplitude modulation* (AM): the only difference between AM and ring modulation is that the former delivers the original signal together with the two sidebands as well (see Figure 10.1b). As a consequence, the timbre of an amplitude modulated signal is much closer to the original timbre than the ring-modulated version. Hence, it is very easy to interpolate between a signal and its ring-modulated version by the means of AM.

Ring (and amplitude) modulation may be used for several purposes. As already stated, if the modulation frequency is very small, it adds tremolo to the sound. A modulation frequency in the range 10–50 Hz creates a rasping version of the original. Above this limit, ring modulation noticeably transforms the timbre. Since the effect, when applied to a pure sinusoidal wave, creates a very simple spectrum (consisting only of two sinusoids), the method is normally used with more complex incoming signals. For example, when a piano sound

[1] The name originates from the shape of the analog circuit implementing the effect.

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22



a) b)

Figure 10.1: Spectrum of Ring (a) and Amplitude (b) Modulation.

is being ring-modulated, the result will be similar to a prepared piano; when applied to human speech, the result would remain comprehensible speech, although with a non-human timbre. Depending on the modulation frequency (and the incoming signal) we may either get a complex timbre or a chord consisting of a high and a very low sound; by changing the modulation frequency in real-time, the components of this chord would create glissandi in opposite directions.

10.1.2 Frequency Modulation

During *frequency modulation* (FM), the frequency of an oscillator (a sine wave generator, in most cases) is modified by the incoming signal (see Figure 10.2). If this signal is a sinusoid as well, the result will be a harmonically rich sound, consisting of sine waves with equidistant frequencies and decaying amplitudes. These 'partials' will be centred around the frequency of the oscillator, and the distance between subsequent components will be defined by the frequency and amplitude of the incoming signal. Thus, FM is defined by the following parameters:

Carrier frequency, the frequency of the oscillator (f_c).

Modulation frequency, the frequency of the incoming sine wave (f_m).

Modulation index, the quotient of the frequency deviation (f_Δ) and the modulation frequency $\left(I = \frac{f_\Delta}{f_m}\right)$. The frequency deviation is the biggest possible difference between the frequencies of the original (unmodulated) and the modulated signal, whose value is proportional to the amplitude of the modulator signal.

As stated above, the spectrum of FM consists of a sine wave with frequency f_c and an infinite number of sidebands with frequencies $f_c \pm k f_m$, where k is a positive integer (see Figure 10.3). The amplitudes (denoted as A_k) of these sidebands decay according to the kth *Bessel-function of first kind*: $A_k = J_k(I)$, where I is the modulation index introduced above. As with ring modulation, if the modulator's frequency is small enough (in the range of approx. 0–10 Hz), the effect will be perceived as a vibrato.
In contrast to amplitude (and ring) modulation, FM creates a rich timbre even when using only two sine wave oscillators. Therefore, FM does not necessarily require an incoming

Open Access 2026-10-22

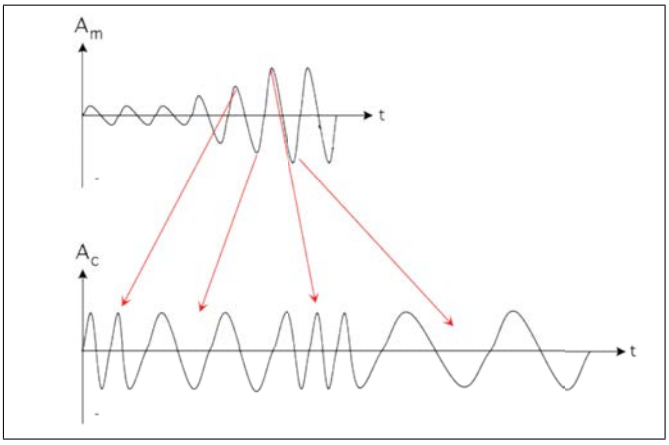


Figure 10.2: Frequency Modulation.

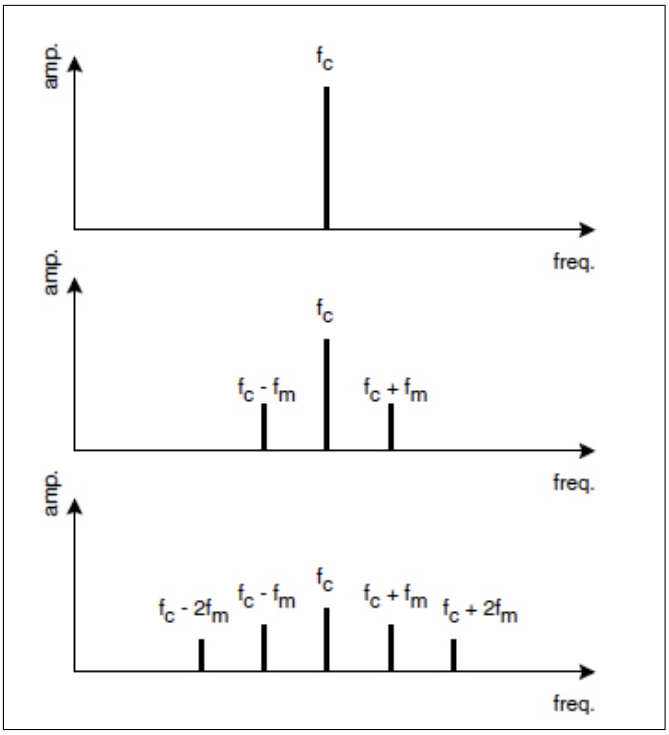


Figure 10.3: FM spectrum.

sound to create interesting spectra. Even by live-controlling only the modulator signal, FM allows a wide range of timbres and interpolations between these. It is also possible to cascade multiple oscillators, offering almost unlimited timbral possibilities with a minimal computational overhead.

10.1.3 The Timbre of Modulated Sounds

The *harmonicity ratio*, defined as $H = \frac{f_m}{f_c}$, describes the behaviour of the modulated spectra if both the carrier and modulator signals are sinusoids. Namely, if H is rational and its irreducible form is $\frac{p}{q}$, then the timbre is harmonic with a base frequency $f_0 = \frac{f_c}{q} = \frac{f_m}{p}$. On the other hand, if H is irrational, the spectrum becomes inharmonic.

Open Access 2026-10-22

Aliasing effects also have to be taken into consideration: if a sideband has a negative frequency, that sideband would appear with a frequency equivalent to the absolute value of its original (negative) frequency, with an inverted phase. This phenomenon can substantially alter the originally computed spectrum.

The main difficulty with both amplitude and frequency modulation arises from the fact that the timbre depends on the rationality of H. In the case of AM, the limited number of sidebands makes it slightly easier to predict the sound (thus, to 'improvise' with the settings in real-time), although achieving a specific timbre (e.g. a harmonic modulation of the signal) requires planning in advance in all cases. However, the complexity of an FM spectrum makes it impossible to predict the timbre by intuition; the only option for controlled improvisation is the extensive use of properly defined presets.

10.2 Examples

10.2.1 Ring Modulation in Integra Live

The `le_10_01_integra.integra_` file is downloadable using the following link: `http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_10_01_integra.integra_`

The project `le_10_01_integra.integra_`contains three blocks, allowing the ring modulation of a test tone, a file and direct input, respectively. The core of the effect is identical in all cases: the main controls are depicted in Figure 10.4. These include the amplitude (called '*Drive*') and frequency of the modulator as well as the parameters of a low-frequency sinewave oscillator, which is responsible for modulating the modulator signal itself.

Figure 10.4: Ring modulation in Integra Live: the most important controls of the ring modulator.

The main controls for the test and file inputs are depicted in Figure 10.5. The test tone can either be a sine wave with a fixed frequency or a noise. The '*Burst*' option creates a burst of white noise each second. The file player will start playing the loaded file after pressing '*Bang*'.

Open Access 2026-10-22

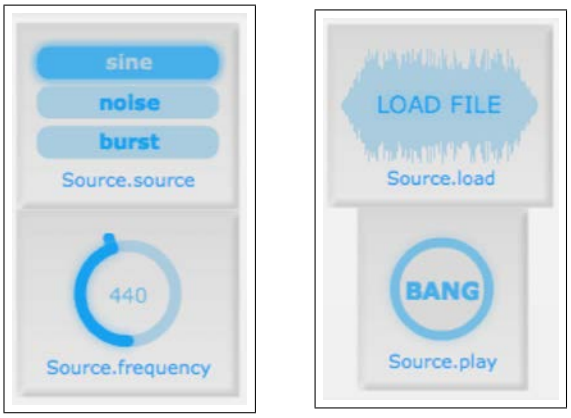


Figure 10.5: Ring modulation in Integra Live: the test tone and file player sources.

10.2.2 Modulation in Max

`LEApp_10_01_` is downloadable for Windows and Mac OS X platforms using the following links: `http://www.typotex.hu/upload/Szigetvari_interaktivok/_SzigetvariAndrea_LiveElectronics_ENG/10_WIN.zip`, `http://www.typotex.hu/_upload/Szigetvari_interaktivok/SzigetvariAndrea_LiveElectronics_ENG/10_OSX.zip_`

Amplitude and Ring Modulation is presented by `LEApp_10_01`, depicted in Figure 10.6. The program can modulate the source signal in two different ways. In both cases, the amount of amplitude modulation can be set with the '*Dry/wet* slider. When this slider is set to its maximum value, pure RM happens, while the minimum value turns off any modulation. Any value inbetween creates AM.

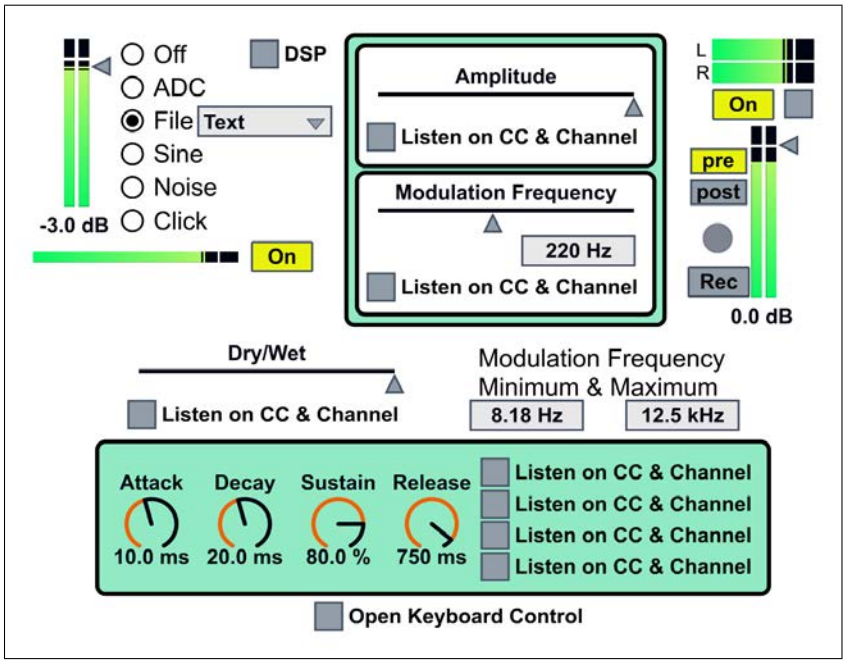


Figure 10.6: Amplitude and Ring modulation in Max.

Open Access 2026-10-22

The upper block applies a modulation of fixed frequency. which can be set by the '*Modulation Frequency*' slider and number box. The '*Amplitude*' slider sets the overall amplitude of this block. The lower block contains a polyphonic synthesizer controllable either by a MIDI keyboard, mouse, or the computer keyboard. The input of the synthesizer can be accessed by pressing '*Open Keyboard Control*', which opens the generic keyboard control panel. The pitch and velocity of the input control the modulation frequency and the overall amplitude of the signal. This latter is modulated with and ADSR envelope whose parameters can be set with the respective dials.
The '*Modulation Frequency Minimum & Maximum*' settings map the incoming MIDI values to frequencies. This applies both to the Modulation Frequency of the upper block and the MIDI Pitches incoming into the synthesizer of the lower block.
`LEApp_10_02` — depicted in Figure 10.7 — presents an FM synthesizer similar to the lower block of the previous one, also controllable by a MIDI keyboard. Here, the incoming MIDI pitches define the carrier frequency of the FM signal. The modulator frequency is computed as the product of the carrier frequency and the '*Harmonicity Ratio*', which can be set manually or using a MIDI CC value. The '*Modulation Index*' defines the width of the spectrum, as discussed previously.

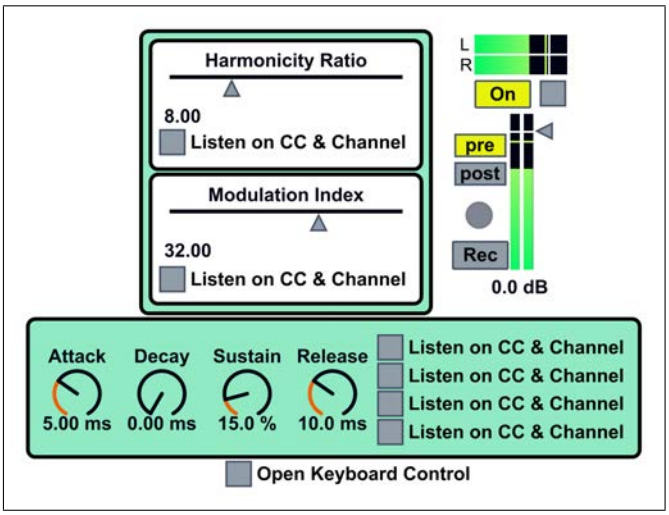


Figure 10.7: Simple FM in Max.

A slightly more complex FM synthesizer is presented by `LEApp_10_03` (see Figure 10.8). In this program, a sine wave is frequency-modulated by itself, i.e. the sine wave is fed-back into its own input with a short delay. The feedback gain defines the modulation index. The whole process can be seen as an infinitely cascaded frequency modulation, where the modulator frequency of each cascaded oscillator is defined by the delayed signal. As a consequence, the computation of the actual harmonicity indices (and thus, the modulator frequencies) can be extremely hard, if not impossible. In other words, the spectrum is quite unpredictable.

10.3 Exercises

1. Make the *le_10_01_integra.integra* project interactive! Route MIDI CC values to the most important parameters of the modulator (depicted in Figure 10.4). Load different

Open Access 2026-10-22

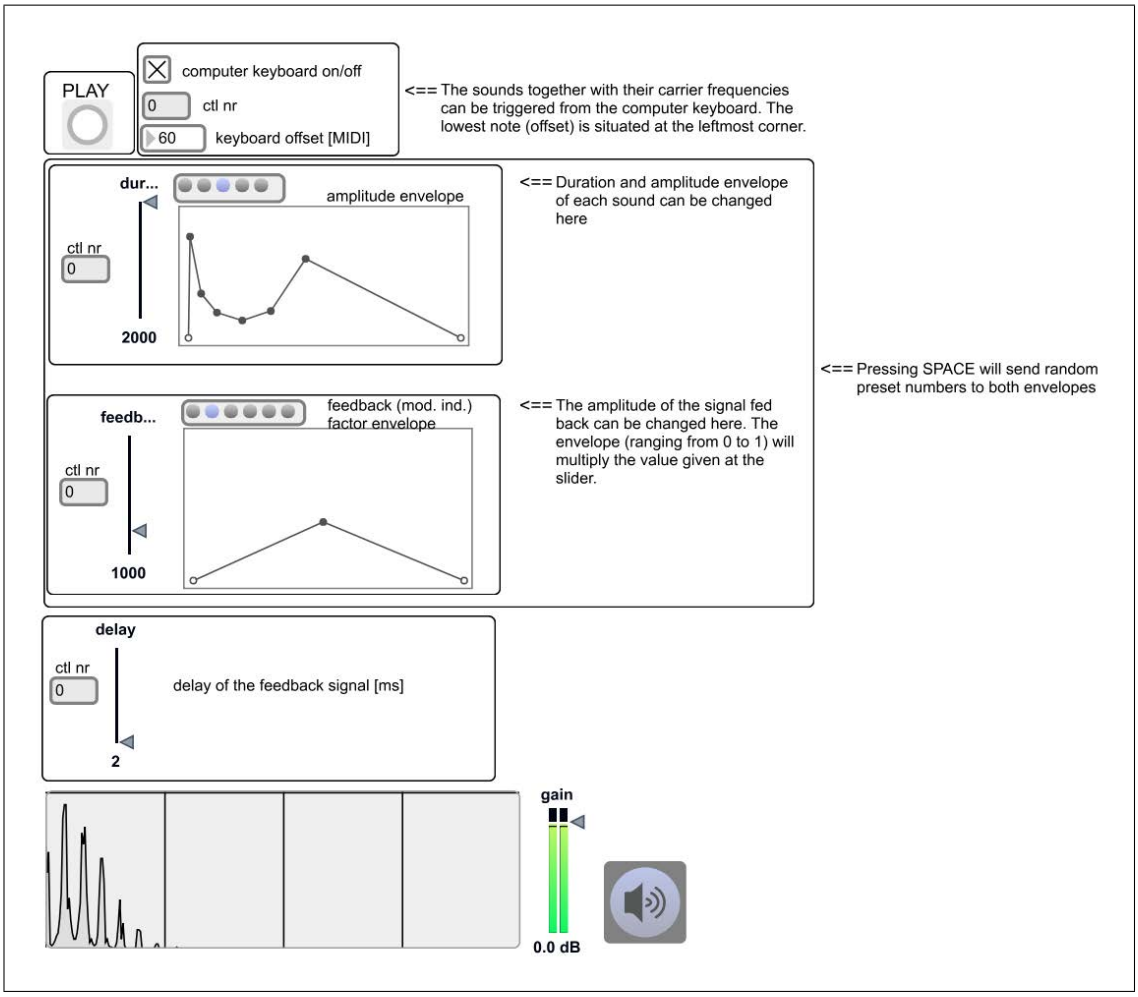


Figure 10.8: Infinitely cascaded frequency modulation by means of feedback.

sounds into the file player and observe how RM changes their timbre. Create the same modulations with the upper block of `LEApp_10_01`.

2. Recreate the Dalek voice (see e.g. `http://www.youtube.com/watch?v=_rSNkSAa1eG4`), both with Integra Live and `LEApp_10_01`!

3. Find and explore the three different regions (tremolo, rasping, timbre transform) of AM/RM! Choose a sound sample, and create three differently modulated versions based on the sample. In the first case, the modulation frequency should stay below 10 Hz; in the second, between 10 and 50 Hz; in the third, above 50 Hz. Do this process with pure sine wave, noise, human speech, percussive music, instrumental music and chamber/orchestral music! What are the differences between the results? On which sources does the RM have the biggest effect? Find the approximate frequency limits of the three aforementioned regions for two different sample types (e.g. human speech and orchestral music). Do these values depend on the type of the carrier?

4. We have mentioned previously that ring modulation of piano sounds creates an effect similar to the preparation of the piano. Explore this by loading a piano sample

Open Access 2026-10-22

into the modulator (either `le_10_01_integra.integra` or `LEApp_10_01`) and ring-modulating it! In which conditions would the original piano sounds become non-pitched? Compare the ring-modulated piano sound with the prepared piano! Compare it with the ring-modulated guitar sound, too!

5. Observe the difference between AM and RM by continuously changing the '*Dry/wet*' slider of `LEApp_10_01` while holding the same note(s) on the keyboard.

6. Listen to the affect the harmonicity ratio has on the FM timbre! Listen to a few FM sounds of `LEApp_10_02` with the following harmonicity ratios: $\frac{1}{2}, \frac{1}{3}, \frac{1}{5}, 2, 7, \sqrt{5}, \pi$. For each case, choose a small (below 5) and a big (above 20) modulation index. Use carrier pitches from the central range of the piano.

7. Observe aliasing effects! Set '*Modulation Index* to 1 and choose a rational harmonicity ratio between 0.5 and 1 (e.g. $\frac{1}{2}$ or $\frac{3}{5}$). Find the region (around the centre of the keyboard) where the timbre of the created sounds is similar (identical). Starting from this region, find the pitches in both directions where the timbre starts changing considerably (you'll notice that the sounds do not follow a musical scale any longer). Repeat this process with a bigger modulation index (e.g. 10)! Repeat this with several harmonicity ratios (both rational and irrational, both above and below 1).

8. Choose any pitch, and play it on the keyboard. Without releasing the key, change the harmonicity ratio and the modulation index. Give an explanation to what you hear!

9. Observe how the feedback amplitude, the feedback envelope and the delay time in `LEApp_10_03` affect the timbre of the original sine wave! Try every preset of the feedback envelope with high and low values of both the delay time and the feedback strength.

Open Access 2026-10-22

Chapter 11

Granular Synthesis

Granular synthesis may be used in several different ways, which makes the technique extremely versatile. By setting the parameters properly, we may transpose, slow down or accelerate a sample as well as 'freeze' it at a given moment, converting the instantaneous timbre of the recording into an independent sound for further processing. Moreover, it can be implemented in real-time with very low computational cost, which makes it perfectly suitable for real-time performance. Adversely, the sound quality of transposition and time stretch, in particular, is not comparable to the sound of non-real-time solutions of the same effects.

11.1 Theoretical Background

Granulation consists of taking short segments ('*grains*') of either a recorded or a generated sample and mixing them together after some modifications. These modifications include changes in the *playback speed/transposition*, *loudness*, *envelope* or *panning* of the grain. Although different implementations exist, the parameters involved describe probability distributions in most cases, which are used in order to obtain the actual parameters of the grains during the synthesis process. The preferred variables include, in addition to the above, the *lenghts* and *positions* (within the full sample) of the grains.
Either sampled or generated, the grains are always multiplied by an amplitude envelope in order to avoid clicks. The timbre of the grain depends both on the spectrum of the (unmodulated) grain and on the applied amplitude envelope.
The most important parameters of granular synthesis are:

Waveform of Grains. The timbre of the final sound is mainly defined by the timbre of the individual grains. This latter, as previously mentioned, depends on both the amplitude envelope and on the spectrum of the original signal. It must be noted that the less smooth the envelope, the less important the original signal will become. The reason is that a segmented envelope adds noise (and clicks) to the timbre.

It is important how the changes of the waveforms are defined. The main options are:

- Keeping the waveforms constant creates a constant, 'frozen' timbre.
- By sampling new waveforms from the original signal using a constant 'playback speed', we can transpose and/or stretch the original sound.

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

- By choosing the waveforms according to an algorithm (e.g. random selection within a range) we can create very complex, constantly changing sonic textures.

Grain Duration. Defines the duration of the individual grains (usually in ms).

Grain Transposition. By playing back the same grain with different speeds, we can alter the pitches of the grains. This can either be used for transposing the original signal (when applying the same amount of transposition) as well as for creating 'sound clouds' (with varying transpositions on each grain).

Grain Density. Defines the amount of grains played back simultaneously. Below 8 Hz, we usually get a segmented sound rather than a solid timbre. By increasing this value, we reach trembling sounds. Above a certain value (usually, about 80 Hz) we reach a continuous timbre.

Spatialisation. It is also possible to set the spatialisation of each grain, to create more dense sounds.

11.2 Examples

11.2.1 Granular Synthesis in Integra Live

The `le_11_01_integra.integra_`and `le_11_02_integra.integra_`files are downloadable using the following links: `http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_11_01_integra.integra`, `http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_11_02_integra.integra_`

The project `le_11_01_integra.integra_`contains a standard granular synthesizer, as depicted in Figure 11.1. The waveform source position, grain duration, transposition, grain density and spatialisation can be controlled by setting an interval, defining the allowed range of these parameters. Then, the actual parameters of the grains are selected randomly (within the defined range). The grain amplitude envelope is controlled by setting their attack and release times. The synthesizer can use either a sound file or a live-sampled buffer as a source for the grains.
As we can see from this interface, even with the automatic computation of each grain based on parameter ranges, the device has too many parameters for simultaneous live control. Of course, one may fix a couple of parameters and limit the number of live-controlled ones during performance. A convenient way to do this is the use of presets. In addition, the envelope-based automation feature of Integra Live gives us a further option. The project `le_11_02_integra.integra_`(depicted in Figure 11.2) contains the same granular synthesizer, *this time controlled by a set of envelopes*. By starting the timeline we hear how this automation affects the sound being played.
The average starting position of the grains, controlled by the '*relMeanCloudPosition*' parameter, will advance automatically from the beginning to the end of the loaded sample. As the allowed deviation from this average is zero at the beginning (and stays zero for a while), the playback follows more or less the original sample.

Open Access 2026-10-22

Figure 11.1: A granular synthesizer in Live View. It can process either pre-recorded sounds (stored in `Granular.loadsound`) or live input (which can be recorded with `Granular.recSound`).

As time advances, the original parameters — which were set so that the playback would resemble the original sample — start deviating more and more, creating a sound cloud by the end of the Scene.

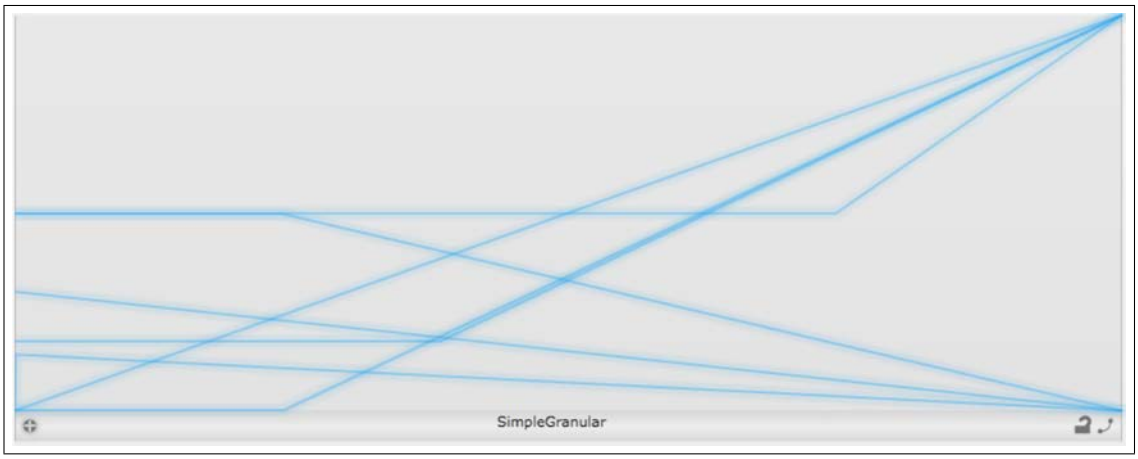


Figure 11.2: The same granular synthesizer, this time with parameters controlled by envelopes.

11.2.2 Granular Synthesis in Max

`LEApp_11_01_` is downloadable for Windows and Mac OS X platforms using the following links: `http://www.typotex.hu/upload/Szigetvari_interaktivok/_SzigetvariAndrea_LiveElectronics_ENG/11_WIN.zip`, `http://www.typotex.hu/_upload/Szigetvari_interaktivok/SzigetvariAndrea_LiveElectronics_ENG/11_OSX.zip_`

Granular synthesis is presented by `LEApp_11_01`, a granular synthesizer capable of synthesizing up to 8 simultaneous grains, whose interface is depicted in Figure 11.3. The uppermost row controls the playback of the sound, which can either be continuous or triggered. In the latter case, an envelope selector, together with a sample duration control and a trigger to start the sound will be displayed on the screen.

There are three different sources:

Open Access 2026-10-22

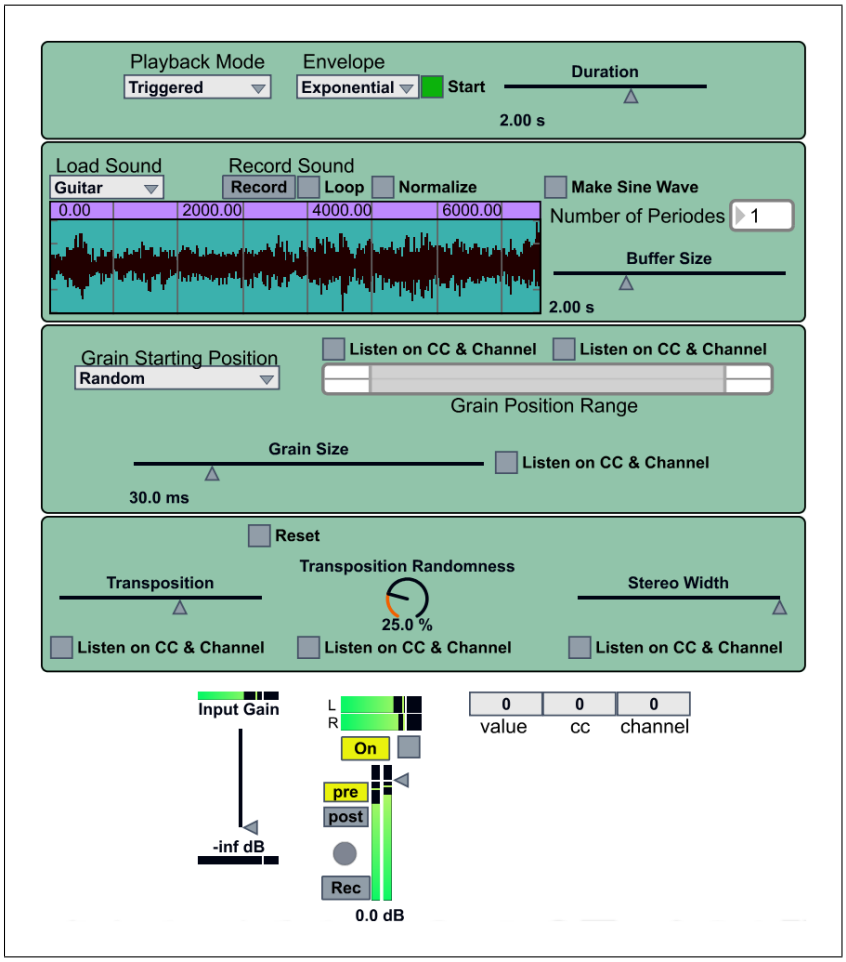


Figure 11.3: A simple granular synthesizer in Max with live-sampling capabilities.

- Pre-recorded and pre-loaded samples from the hard drive by using the '*Load Sound*' option.

- Live-recorded sounds by using the '*Record Sound*' option. Recording can be started and stopped with the '*Record*' toggle. Circular recording (in which case the recorder head will jump to the beginning of the buffer and continue recording each time it reaches the end of the buffer) can be activated with the '*Loop*' toggle. The recorded buffer can be normalised with the '*Normalize*' button.

- Pure sine waves with the '*Make Sine Wave*' button. The number of periods of the generated sine wave can be set as well as the total length of the buffer can be set separately.

The parameters of the grain are accessible through the '*Grain Starting Position*' and '*Grain Size*' settings. The starting position of the grains can either be '*Auto advance*' or '*Random*'. In the first case, the starting position of the grains will automatically increase as time goes on, controlled by the '*Playback Speed*'. In the second, the starting position will be determined randomly for each grain within the boundaries determined by the '*Grain Position Range*' setting.

Open Access 2026-10-22

The lowermost row controls the transposition and panning of the grains. '*Transposition Randomness*' controls the individual deviations of the amounts of transpositions from the average value, set by '*Transposition*'. The '*Stereo Width*' controls the spatialisation of the individual grains. When this slider points to the left, the two speakers will receive the same (monophonic) signal. The more the slider is moved to the right, the more distributed the grains will be between the two speakers; when the slider is moved completely to the right, the individual grains will be routed completely either to the left or to the right speaker.

11.3 Exercises

1. Examine the parameter changes in the `le_11_02_integra.integra`_project. How does...
   - ...pitch transposition (both its minimum, its maximum and its range)...
   - ...grain size (both its minimum, its maximum and its range)...
   - ...cloud density (the number of maximally allowed simultaneous grains)...

   ...affect the result? Modify the envelopes and explain how your alterations change the sound!

2. Make the `le_11_01_integra.integra`_project interactive! Route four different MIDI CC values to different parameters of the granular synthesizer. Create at least three different routings and classify them based on their usability for different purposes! Find synthesis parameters where controlling them by the same MIDI CC could make musical sense!

3. Use the circular recording option of `LEApp_11_01`_to record your own voice. Create an interpolation that fades smoothly between the beginning and the end of the recording! Observe how transposition affects your own voice.

4. Compare the granulation of percussive and harmonic sounds, either with the help of the `le_11_01_integra.integra`_project or the `LEApp_11_01`_application. Load a drum loop and explore the timbral space that you can reach by changing the transposition, the grain size and the grain positioning parameters. Repeat the same with a guitar loop. Are there parameters which play a greater role for either of the two sample types? If yes, which parameters are the more dominant for which sound type?

5. How much does the content of the actual source buffer affect the final sound if the grain size is set to an extremely short (e.g. below 5 ms) value? How does the grain size affect the timbre generally?

6. Observe how the same settings affect different types of sound! Prepare at least 3 different settings, and listen to the sound that they produce if the source is...
   - ...a pure sine wave (try sine waves with different number of periods).
   - ...speech.
   - ...percussive (e.g. a drum loop).

Open Access 2026-10-22

- ...harmonic (e.g. solo instrument).
- ...an orchestral sound (either chamber or symphonic).

7. Recall the term *freezing*, which we use to describe the phenomenon that occurs when the starting position of the grains is restricted to a very narrow time interval. Experiment with freezing different sources (both pre-recorded and live-recorded samples). How would you describe the most essential common property of these sounds? How do these sounds serve for transposition purposes (both with and without random deviation)?

Open Access 2026-10-22

Chapter 12

Sound Processing by Filtering and Distortion

This Chapter presents methods of altering the *timbre* of a sound. More precisely, we revise two common approaches of modifying spectral structure.

12.1 Theoretical Background

We use the term '*filter*' to describe a device which modifies (by either increasing or decreasing) the gain of existing spectral components. By contrast, '*distortion*' happens when, in addition to modifying the existing components, new frequencies are also introduced within the spectrum.

12.1.1 Filtering

As already mentioned, filters can soften and/or raise selected parts of the spectrum of an arbitrary input while keeping the rest untouched (as illustrated by Figure 12.1). Whether considering analog or digital filters, the concepts used in both are basically the same.

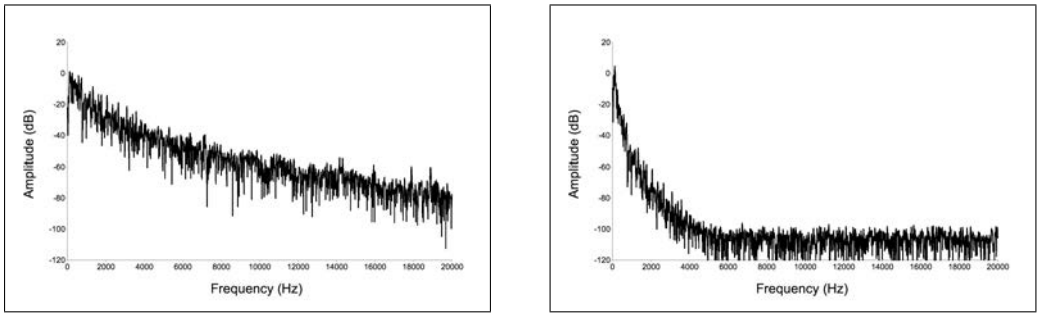


Figure 12.1: The spectra of an incoming signal and a filter's response to it.

For a musician the most important characteristic of a filter is its amplitude response, which describes the frequency-dependent multipliers that the filter applies in order to scale the incoming signal's spectrum.
The basic shapes of amplitude responses, as illustrated by Figure 12.2, include:

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

Low-Pass (LP) and High-Pass (HP): passes frequencies only below (LP) or above (HP) a certain limit (called the *cutoff frequency*).

Band-Pass (BP) and Band-Reject (BR): passes (BP) or rejects (BR) frequencies within a certain range, rejects (BP) or passes (BR) the rest. The range (also called 'band') is defined by its *centre frequency* and *bandwidth*. BR is also called '*notch*'.

Shelf filters: pass all frequencies, but increase or decrease frequencies below (for Low-Shelf — LS) or above (for High-Shelf — HS) a certain shelf frequency by a constant amount.

All-Pass (AP): passes all frequencies (these filters are used in scenarios where only the phase of the signal is to be altered).

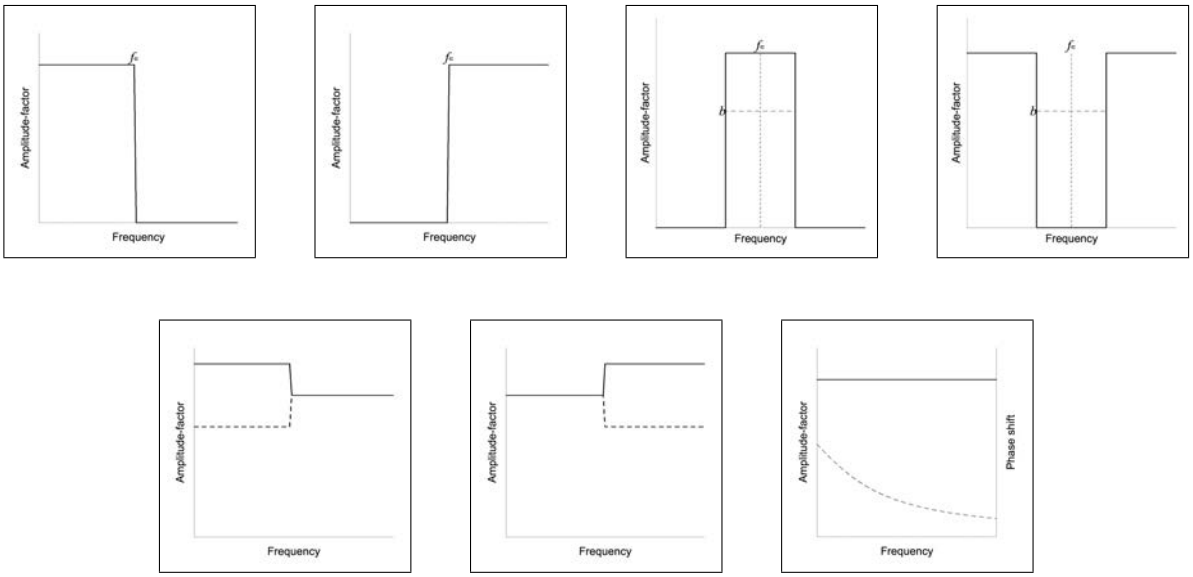


Figure 12.2: Idealised amplitude responses of several filter types. Upper row (from left to right): LP, HP, BP, BR; lower row (from left to right): LS, HS, AP. f_c stands either for cutoff-frequency or centre-frequency; b stands for bandwidth. The AP response depicts the phase response as well, to illustrate the role of these filters.

Two important parameters are *gain* and *Q-factor* (related to the maximum level and the 'flatness' of the amplitude response[1], respectively). For BP filters, the Q-factor is approximated with the ratio of the centre frequency and the bandwidth. Musicians and engineers tend to prefer the Q-factor over the bandwidth as it expresses much better our expectations towards a frequency response curve: the higher the Q-factor, the narrower the pass-band.

12.1.2 Distortion

Distortion adds new frequencies to the spectrum of the input sound. There are many different ways to distort a sound. Here, we present the two most widely used.

[1]Passive filters (which do not contain amplifiers) may not have a gain bigger than 0 dB. This restriction does not hold for active filters.

Open Access 2026-10-22

Harmonic and Intermodulation Distortion

The most straigtforward way of distorting a signal is by scaling it in a non-linear way. It is easy to prove that if we apply such a scaling to a pure sine wave, new partials will appear in the spectrum of the response, whose' frequencies will be integer multiples of the incoming sine's frequency. In other words, any distortion that is based on non-linear scaling of the input will add overtones to an incoming sine wave. Hence, this family of distorters is called *harmonic*.

Pure harmonic distortion only happens when the input is a pure sine wave. When the input consists of more than one frequency, the output will contain the sums and differences as well as the pure overtones of the original frequencies. This phenomenon is called *intermodulation distortion* (IMD).

The simplest form of harmonic distortion is *hard clipping*, where the signal is limited to a threshold. If the incoming amplitude is below this, the signal is untouched; however, any sample exceeding this value would be replaced by the threshold itself. A more sophisticated method is *soft clipping*, where the limiting happens somewhat smoother. Clipping (specially, soft clipping) is better known as '*overdrive*'. It is the most common distortion method.

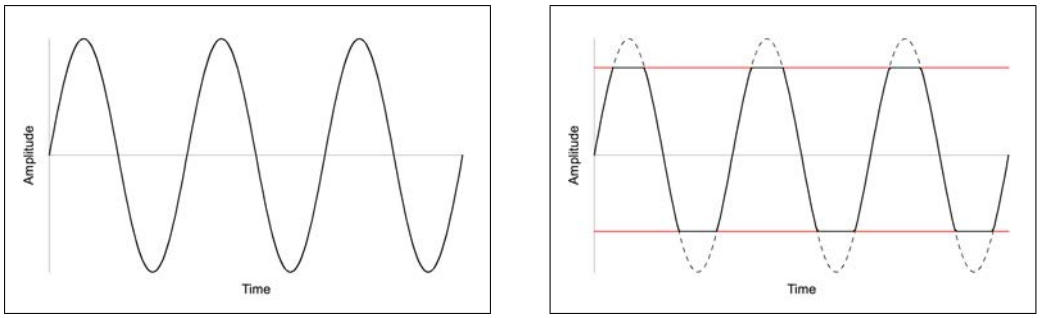


Figure 12.3: Hard clipping (right) applied to a pure sine wave (left). The red lines indicate the threshold.

Quantization Distortion

The digitalization of sound has two main parameters:

- The *sampling frequency*, which defines the time-resolution. It is normally defined through the *sample rate*, which tells us the number of samples taken during a given time (usually, one second). According to the Nyquist–Shannon sampling theorem, one has to set the sample rate to be at least twice the highest frequency that appears in the signal to be digitalized. Otherwise, the high components of the original signal would be 'reflected': they would create fake frequencies which were not present in the original. This phenomenon is called *aliasing*.

- The *resolution of the quantization* or *amplitude-resolution*. It can either be defined through the *bit depth* (showing the number of bits required to store a single sample) or the *dynamic range* (expressing the difference of the loudest and the softest sample allowed by the representation in decibels). A low resolution of quantization might add random noise to the signal — *quantization noise*.

Open Access 2026-10-22

Aliasing and quantization noise are both minimised by Analog-to-Digital (ADC) converters. However, one might reproduce these phenomena in a digital environment purposely, adding unexpected frequencies and/or noise to a signal. Figure 12.4 presents a poorly quantized sine wave.

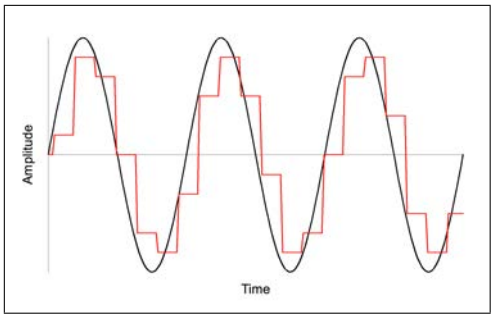


Figure 12.4: Illustration of poor quantization (red) of a pure sine wave (black).

12.2 Examples

12.2.1 Filtering and Distorting in Integra Live

The `le_12_01_integra.integra_` and `le_12_02_integra.integra_` files are downloadable using the following links: `http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_12_01_integra.integra`, `http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_12_02_integra.integra_`

The `le_12_01_integra.integra_` project contains a white noise source, which is driven through an LP, an HP, a BP and a notch filter. Open the project file (as illustrated by Figure 12.5). The four filters are arranged the one below the other, with a level meter and a loudness slider on their left sides. The large slider on the right side controls the overall gain. The `le_12_02_integra.integra_` project contains a simple clipping distorter (see Figure 12.6). After opening the file, one will find a gain controller for a microphone (or line) input and a dial controlling the amount of distortion. Hard clipping can be achieved by setting this dial to its maximum value; at zero, the sound remains unprocessed. The values in between are examples of soft clippings of different strength.

12.2.2 Filtering and Distorting in Max

`LEApp_12_01_` is downloadable for Windows and Mac OS X platforms using the following links: `http://www.typotex.hu/upload/Szigetvari_interaktivok/_SzigetvariAndrea_LiveElectronics_ENG/12_WIN.zip`, `http://www.typotex.hu/_upload/Szigetvari_interaktivok/SzigetvariAndrea_LiveElectronics_ENG/12_OSX.zip_`

`LEApp_12_01_` (depicted in Figure 12.7) contains a biquadratic filter, the simplest one able to realize all filter types presented above. The filter is most easily controlled by the

Open Access 2026-10-22

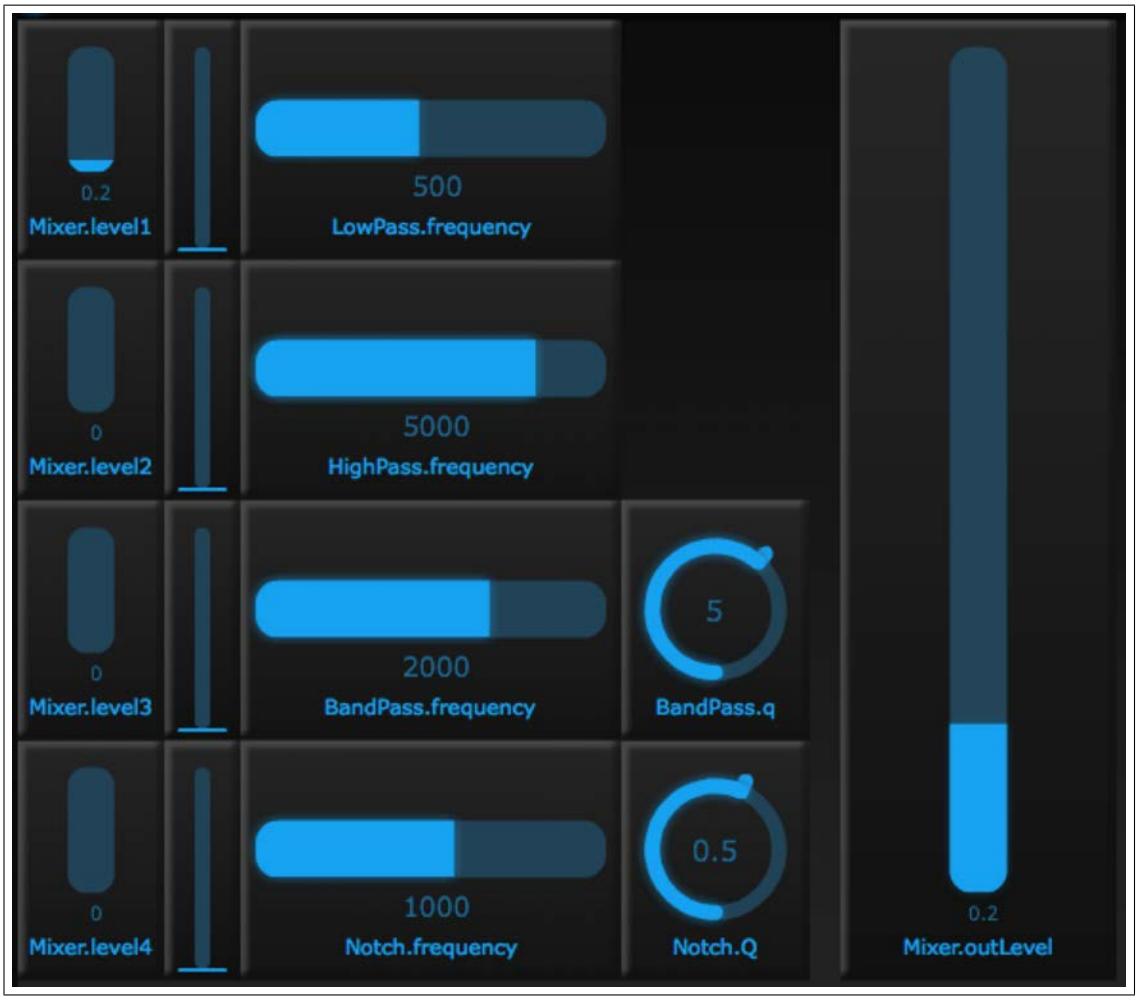


Figure 12.5: The `le_12_01_integra.integra` *project in Integra Live.*

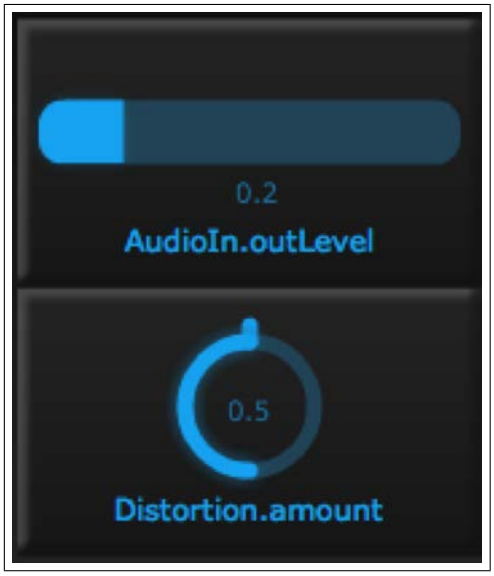


Figure 12.6: Overdrive (soft and hard clipping) in Integra Live.

Open Access 2026-10-22

[filtergraph~]_object, which — in addition to implementing the basic filter types — allows a few additional options as well:

- The '*Analog*' setting will emulate analog BP filters.
- The '*Enable Gain*' setting will enable the *Gain* parameter, converting our (initially) passive filter into an active one.
- The '*Logarithmic Frequency/Amplitude Display*' toggles would switch between linear and logarithmic representations.

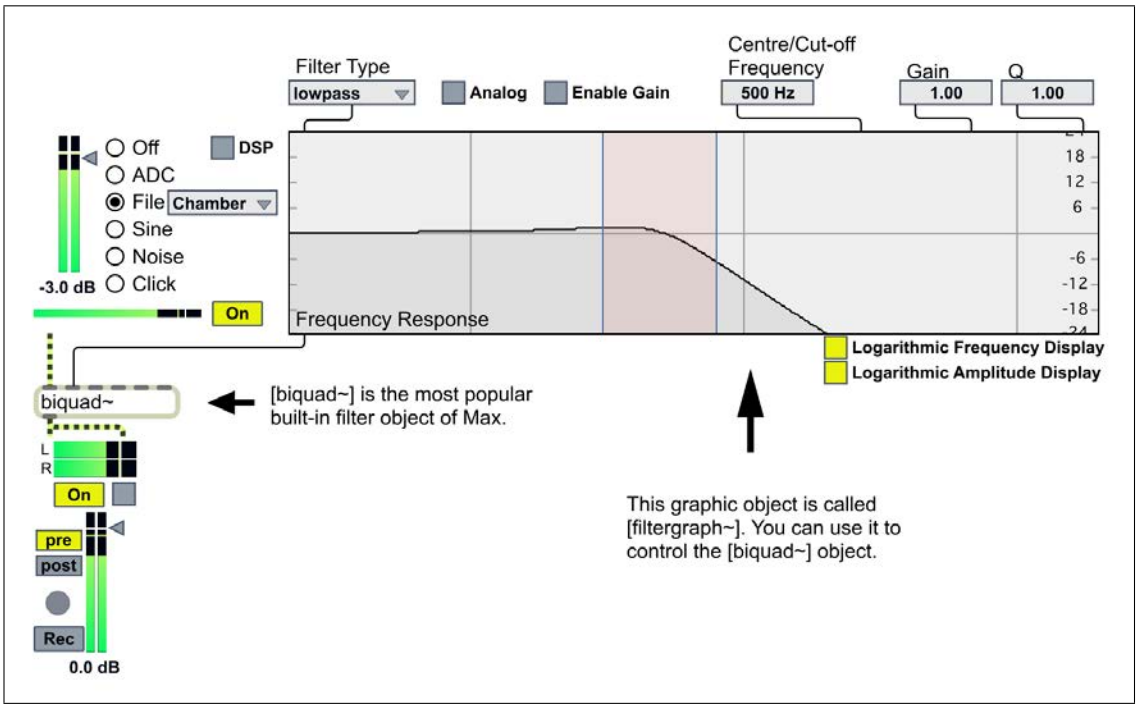


Figure 12.7: LEApp_12_01.

The program LEApp_12_02_(see Figure 12.8) contains a BP filter whose centre frequency and Q-factor can be controlled with MIDI controls. When controlled by MIDI, the '*Lin/Log*' toggle (in the top centre of the screen) switches between linear and logarithmic frequency scaling.

The *LEApp_12_03* application (depicted in Figures 12.9 and 12.10) contain — besides the standard sound source and output — an oscilloscope, which will plot the incoming and the distorted signals in real-time (to inspect the details of the signals, one can freeze the oscilloscope image by turning off audio processing). The dropdown menu above the oscilloscope allows one to choose between two different distortion methods. In this program, we did not hide the 'core' functionality of the patch; by double-clicking on either of the two subpatches ([p_visuals], [p_core]), one can see how these functions are implemented in Max.

(Hard) Clipping is indicated by a red band, which shows the lower and upper clipping thresholds (as seen on the right of Figure 12.3). Click anywhere over the oscilloscope to set the lower threshold and, without releasing the left button of the mouse, drag the mouse pointer to set the higher threshold. This is illustrated on Figure 12.9.

Open Access 2026-10-22

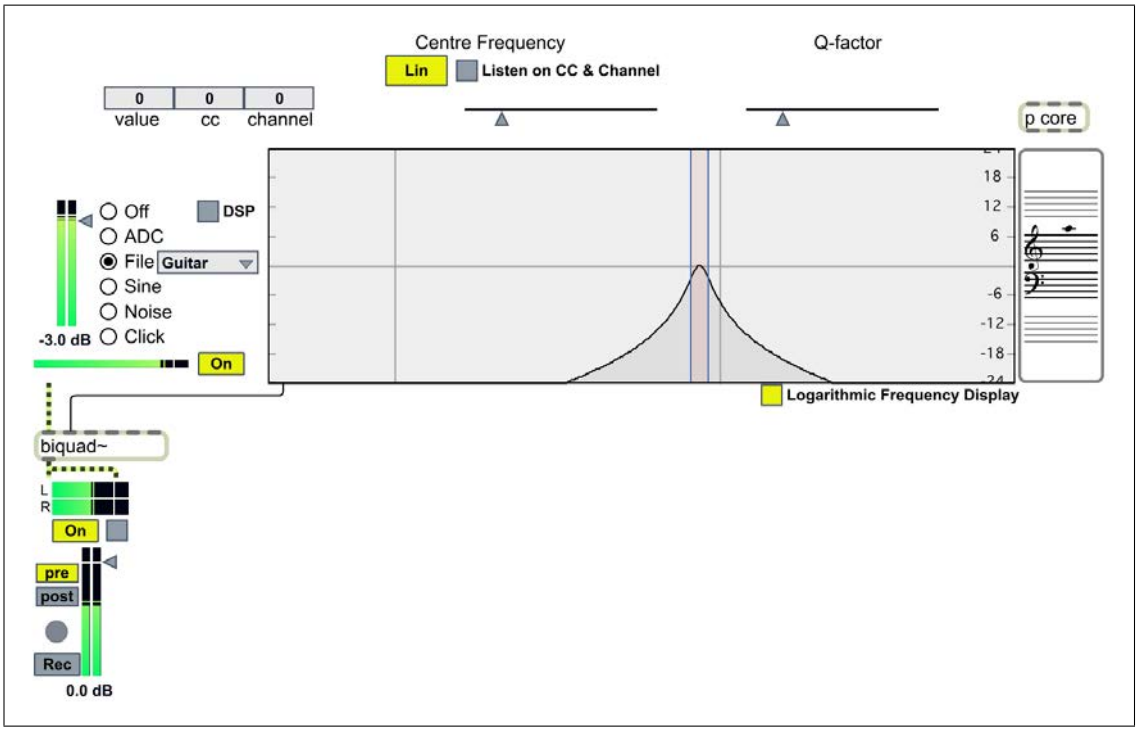


Figure 12.8: `LEApp_12_02`.

For the '*Quantization*' option, two additional dials will appear. The left one sets the bit depth as the number of bits used for representing a sample. The right one sets the sampling rate as a percentage of the original sampling rate of the sound card.

The lower and upper thresholds of hard clipping as well as the bit depth and sample rate of quantization distortion can be controlled by MIDI faders, whose incoming values are routed through a 2D control. This latter can also be controlled by the mouse. When hard clipping is active, the horizontal value controls the distance between the two thresholds while the vertical sets the centre of the band; conversely, when quantization is selected, the horizontal value is responsible for the bit depth while the vertical controls the sample rate.

The program `LEApp_12_04`, depicted in Figure 12.11, contains 9 resonators in parallel and a clip distorter. The sound source is white noise, modulated with an exponentially decaying amplitude envelope. Sounds can either be triggered individually (using the '*One sound*' button) or by starting a random sequence. In this case, the number of generated sounds and the tempo can be set with the number boxes in the top left side of the program.

The resonators are controlled by the lower (red) row of the interface, where each resonator is represented by three values: the centre frequency (in Hz), the linear gain and the Q-factor. The centre frequency of the leftmost resonator is specified in Hz. The centre frequency of the other 8 resonators are specified as multiples of the first resonator. Resonators 2–9 have toggles which, when enabled, will synchronise the gain and Q-factor values with the respective values of the leftmost resonator.

The durations of the individual sounds as well as their base frequencies and Q-factors are defined by random intervals, located in a single row above the resonator control boxes. The '*offset*' values define the lower limit of the random ranges, while the '*rand*' settings express the intervals themselves (note that durations are expressed in ms and frequencies in Hz). The

Open Access 2026-10-22

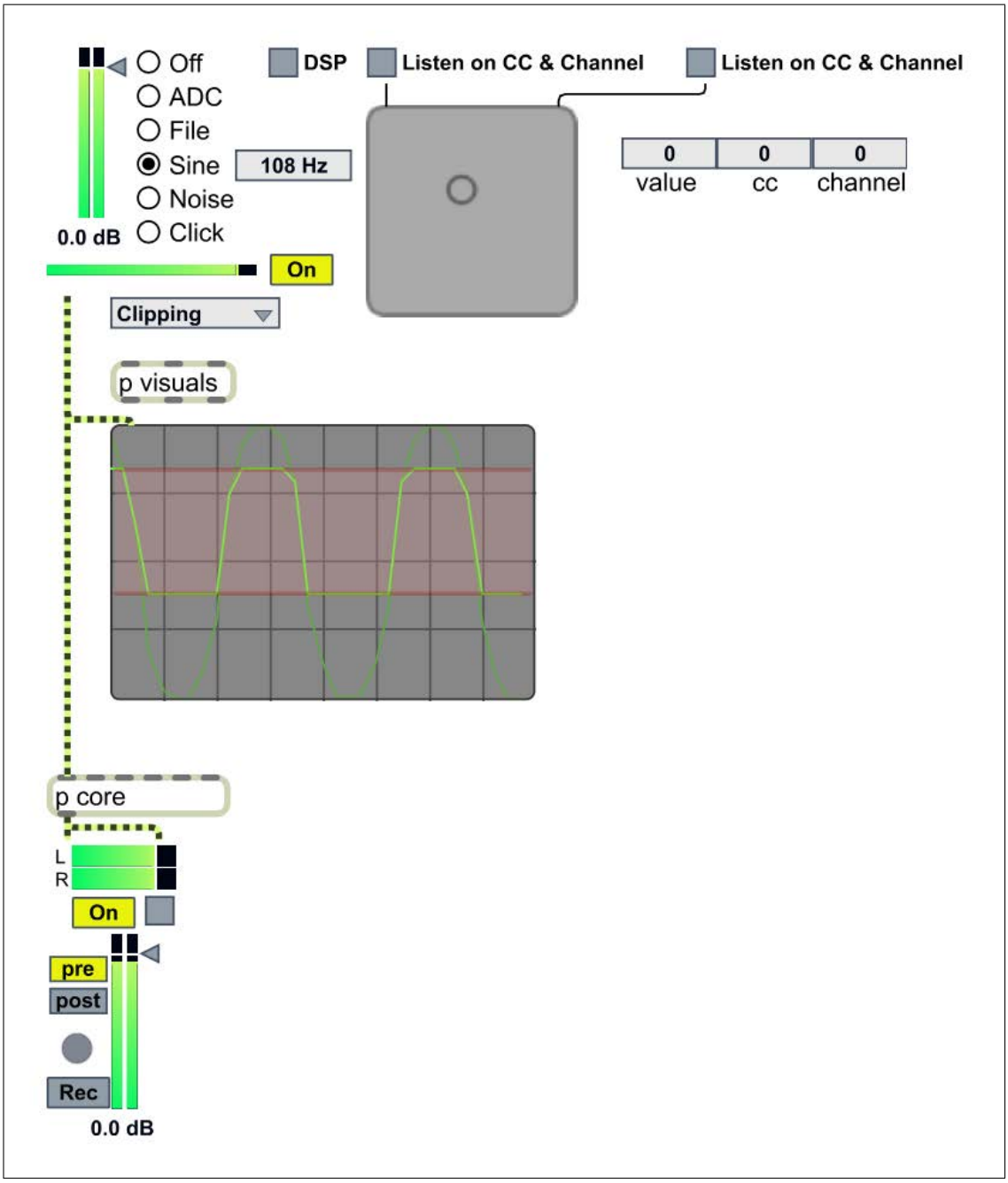


Figure 12.9: Hard clipping in Max.

rightmost setting of this row defines the amount of hard clipping: the signal will be multiplied by this number before being restricted to the interval $[-1;1]$. The main parameters can be controlled with the MIDI faders located above them; the incoming MIDI values will be scaled according to the '*min*' and '*max*' settings below the respective faders.

The program comes with a number of presets, creating quite different sounds. For example, the 1st and 9th preset imitates heavily distorted plucked strings, the 3rd and the 10th generate bell-like sounds, while presets like the 2nd or the 7th create very short glitches and scratches.

Open Access 2026-10-22

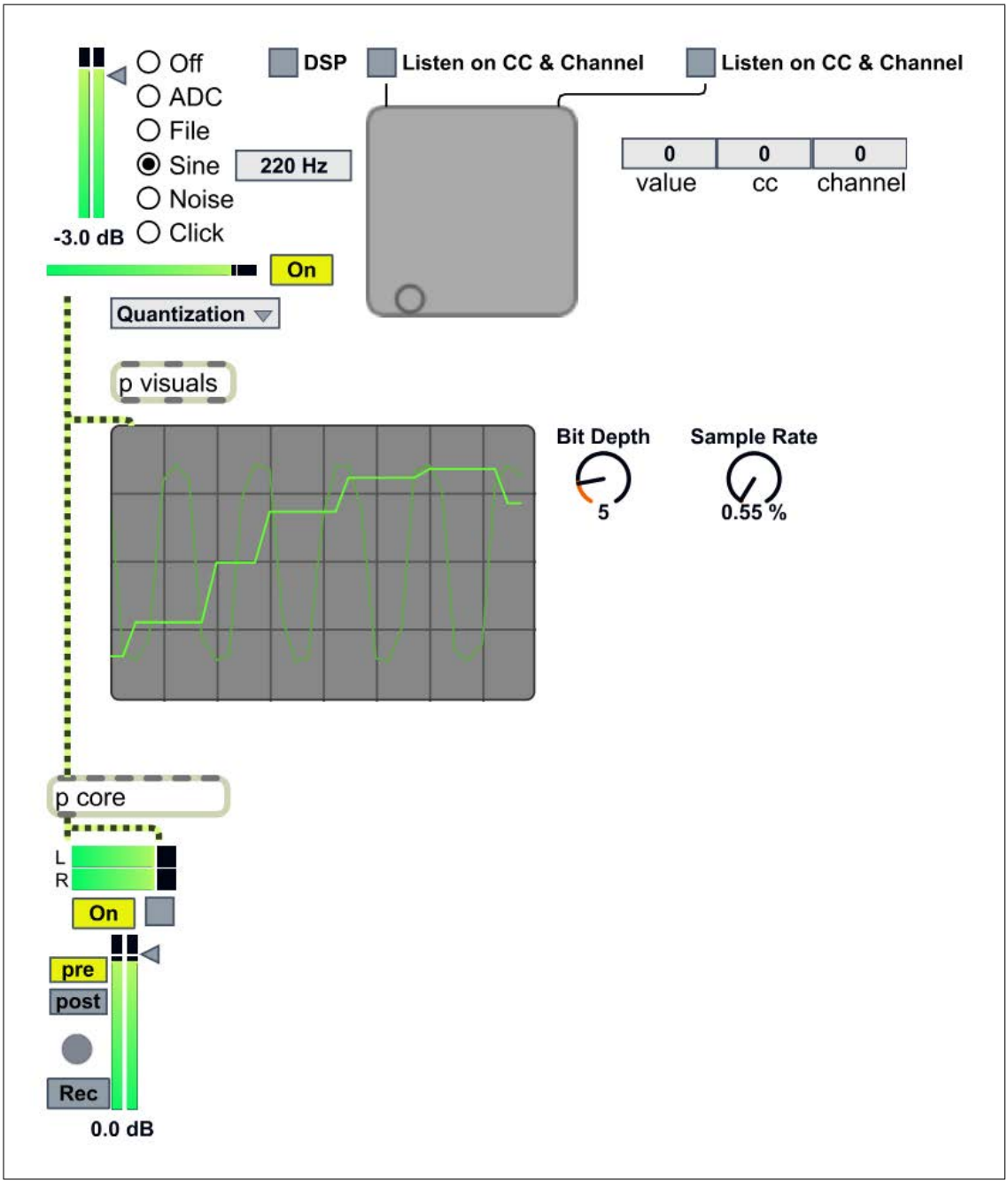


Figure 12.10: Quantization distortion in Max.

12.3 Exercises

1. Experiment with the different filter types of `LEApp_12_01`! See what happens if you change one or more of the filter settings (e.g. try the difference between active and passive filters by enabling and disabling the gain)!

2. Explore the different filters in `le_12_01_integra.integra`. Compare them to the respective `[biquad~]` filters of Max by setting identical settings in `LEApp_12_01`.

3. Try the difference between logarithmic and linear frequency scaling in the

Open Access 2026-10-22

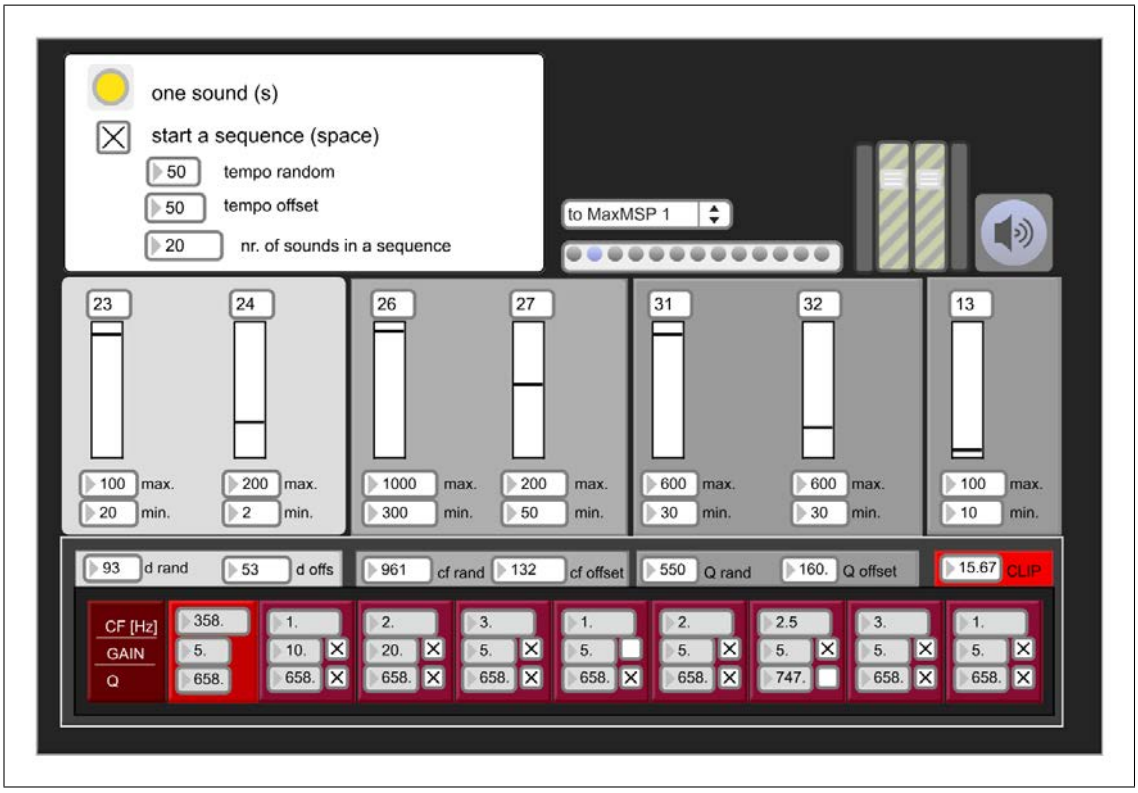


Figure 12.11: `LEApp_12_04`: an instrument built entirely on filtering and distorting short bursts of noise.

`LEApp_12_02` application! Use identical sources in order to make a better comparision.

4. Observe how hard clipping works with the guitar sample of `LEApp_12_03`. Start with the thresholds set to their extremities (so that no distortion happens). Start narrowing slowly the distance between the upper and lower thresholds, kepping the band centred around 0. Listen to the changes of the timbre of the guitar. Repeat this process, but without centring the band around 0. What difference do you hear?

5. Load the drum sample of `LEApp_12_03`. Select hard clipping and set up a very narrow pass-band, centred around 0.5. Listen to the result and describe what you hear. Now, select quantization distortion and observe how it affects the drum sound. What happens in the region where bit depth is between 10–16 and sample rate between 2 and 7 percent? What effect does lowering the bit depth below 8 have?

6. Try all settings of *LEApp_12_04*. Observe how changing the main parameters (duration, centre frequency, Q-factor) affect the sound!

7. Implement the basic functionality of `LEApp_12_04` in Integra Live! Create an empty project with a single Block. Open the Block and remove all pre-created Modules. Add a Test Source Module, a few (4–9) Resonant Bandpass Filter Modules, a Distortion Module and a (Stereo) Audio Out Module. Connect the output of the Test Source to the input of the Resonant Bandpass Filters, the outputs of the filters to the input of

Open Access 2026-10-22

the Distortion and the output of the Distortion to the (Stereo) Audio Out. Set the test sound to white noise by choosing '*noise*' in Test Source and make sure that all filters are set to '*peak*' normalisation. Listen the result! How does the sound depend on the distortion type (which you can set in the Distortion Module) and the input gain of the noise (controlled by the output level of the test source)?

8. Add control capabilities to the previously created Integra Live project! In Arrange View, create routings that connect different MIDI control values to the '*frequency*' and '*q*' parameters of the resonant filters. To achieve fixed ratios between the frequencies, route the same MIDI CC value to all centre frequencies using different ranges for the mapping (e.g. 100–200 Hz for the first filter, 200–400 Hz for the second one, 300–600 for the third etc.). You can also use another single MIDI CC to control the Q-factors in a centralised way. Don't forget to link a separate MIDI fader to the '*amount*' of Distortion!

9. Imitate the playing modes of `LEApp_12_04`! Add an Envelope to control the output level of the Test Source Module and create two exponentially decaying Envelopes. Add two Scenes which contain exactly these two Envelopes and a third Scene where the output level is set to 0. Set the State of the first Scene to Play, the second one to Loop and the silent Scene to Hold. Go to Live View. Activate the scenes one by one (using their respective keyboard bindings) and listen to the result. Observe how the shape of the Envelopes and the duration of the Scenes affect the sound.

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

Chapter 13

Sound Processing by Delay, Reverb and Material Simulation

This Chapter presents methods based (at least, partially) on *delay*, that is, the temporal shifting of an incoming signal. After introducing *delay lines* or *tape delays*, we turn towards the basics of *reverberation* and *material simulation*, two techniques which achieve similar results through different methods.

13.1 Theoretical Background

13.1.1 Delay

By recording a sound and playing it back at some later point in time, we can delay the signal. A delay line may be characterized by a single parameter, that is, the amount of time with which the signal is shifted.
It is quite common to *feed back* the resulting signal to the input of the delay engine. If the fed-back signal is attenuated before being re-input, the result will sound like an echo. If it is boosted before being fed back, an uncontrolled, 'blown up' sound will be created.
Delay units without feedbacks usually form parts of more complex systems (as we will see in the next sections of this chapter), although they may be used as stand-alone effects as well. In spite of being simple, delays offer a big range of effects. Multiple (fixed) playback times allow for the creation of interesting rhythmic patterns. A long delay (e.g. at least 50–100 ms) and feedback creates echoing effects. Since continuously changing the delay time changes the playback speed as well, delay units can be used to modulate the pitch of the sounds. This happens e.g. during *scratching* (a popular technique among DJs), but with a regular (e.g. sine wave) modulation of the delay time, one can also introduce *vibrato* (FM) or *chorusing* effects as well. These can easily controlled live if regular, stepped presets are used. Continuously changing the delay time can cause unpredictable modulations.

13.1.2 Reverberation

When a sound is produced in an enclosed space, it causes a large number of reflections. These first build up and then slowly decay as the sound is absorbed by the walls and the air. This phenomenon is called *reverberation*. The first few sounds that reach the listener after

Open Access 2026-10-22

being reflected off the walls are called *early reflections*. The mixture of later reflections and room resonances is called *reverb* or *late reverberation* (or *tail*).
The simplest reverberation processors would split the incoming signal into three parts:

Direct signal, sent to the output unprocessed, without any delay.

Early reflections, treated with multiple tap delays and filters.

Late reverberation, delayed, filtered and processed in other possible ways in order to create an exponential decay.

This model has two very important parameters:

Pre-delay: the time between the arrival of the direct signal and the first reflected one.

Reverb time: the time that it takes the late reverb to decay by 60 dB (also called t_{60}).

The purpose of reverberation (combined with spatialisation) is the simulation of a sonic space. By combining the direct sound with a number of processed and filtered delays, reverberation creates the sensation of the sound occuring in a physical space. In a live situation, changing parameters such as the pre-delay or the reverb time will create the sensation of sonic objects advancing or receding. Also, applying different reverberations to different sounds creates a spatial context for the music, e.g. by moving the accompaniment to the background and emphasizing a voice situated (virtually) closer to the audience.

13.1.3 Material Simulation

The princpiles behind material simulators are similar to those of reverberation, although in this case the reflective and resonant qualities of a specific material are simulated. When computing the reverberation of a room, reflections usually play a much important role than resonances; for material simulation, it is the opposite. Therefore, it is possible to build material simulators relying on a set of resonant (band-pass) filters designed with high accuracy and distortion effects which emulate the (usually) non-linear behaviour of specific materials. In this case, the incoming signal is assumed to simulate an impact on the surface of the object.
However, this is not the only way to go. One may compute the '*resonant modes*' – these are standing waves or vibrations and will be determined by the physical properties of the material; size, elasticity, density, etc. Each standing wave occurs at a specific frequency and the total sound emitted by the object can be obtained by mixing these modes. This method is called *modal synthesis*, and gives highly accurate results for certain types of objects (for instance, plates or cavity resonators). Unfortunately, modal synthesis is usually quite expensive computationally.
Other methods are also available for modelling materials. A collective term for these techniques is *physical modelling* or *physical modelling synthesis*, as the applied procedures are always deduced from physical models describing the actual objects.
Material simulation is a convenient synthesis method, offering an alternative to sampling. Situations when material simulation may be preferred over sampling include 'sound extrapolation' (when, during simulation, one changes the physical parameters of a material in ways that would not be achievable by real objects) or when the 'realness' of a stored sample does not compensate for the lack of variability of the recorded sound.

Open Access 2026-10-22

13.2 Examples

13.2.1 Material Simulation in Integra Live

The le_13_01_integra.integra_ file is downloadable using the following link: http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_13_01_integra.integra_

A simple material simulator is presented in the le_13_01_integra.integra project (see Figure 13.1). Here, a string is being plucked, and this sound is sent through a material simulation module. The simulated material can be choosen from the list on the right. The parameters of the plucking (emulated by a low-pass filtered, feedback delay line) can be set with the sliders on the left. The '*bang*' in the middle plucks the string.

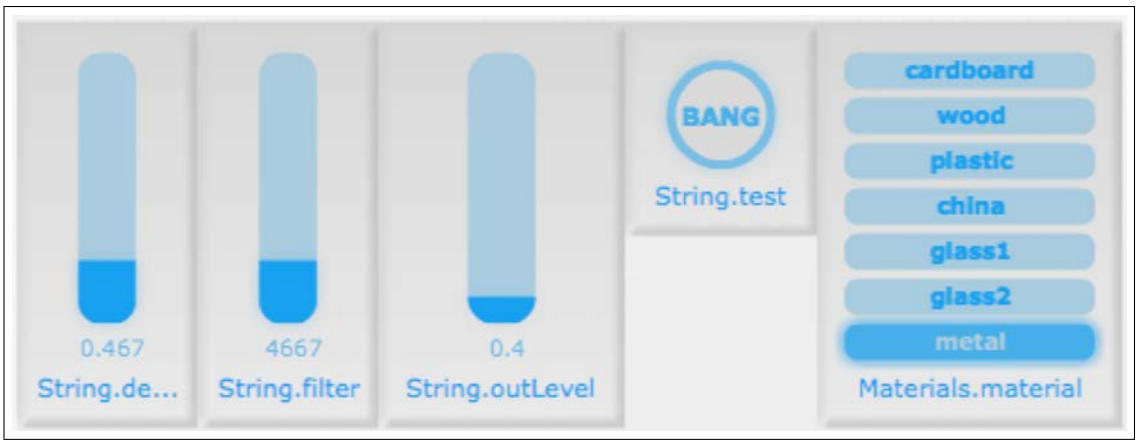


Figure 13.1: Material simulation in Integra Live.

13.2.2 Delay and Reverb in Max

LEApp_13_01_ is downloadable for Windows and Mac OS X platforms using the following links: http://www.typotex.hu/upload/Szigetvari_interaktivok/_SzigetvariAndrea_LiveElectronics_ENG/13_WIN.zip, http://www.typotex.hu/_upload/Szigetvari_interaktivok/SzigetvariAndrea_LiveElectronics_ENG/13_OSX.zip_

A feedback delay line with three taps is presented by LEApp_13_01 (see Figure 13.2). On the left and right sides, a generic audio input and an output are located, respectively. The '*Dry*' slider will forward the source directly to the output. The three horizontal sliders ('*Tape 1*', '*Tape 2*' and '*Tape 3*') set the delay times associated with the three tap heads (note that they have different ranges). These delay lines are preceded by a low-pass filter, controlled by the '*LPF*' dial. The three '*Gain*' sliders control the levels of the delayed signals. Finally, the three '*Feedback*' toggles enable the feedback on the respective delay lines. Note that the fedback signal will first go through the low-pass filter. The purpose of this filter is to eliminate high frequency resonances from the feedback loop.

LEApp_13_02 contains a simple reverberator (see Figure 13.3. A standard generic signal source and output is located on the left and right of the patch, respectively. The two dials on

Open Access 2026-10-22

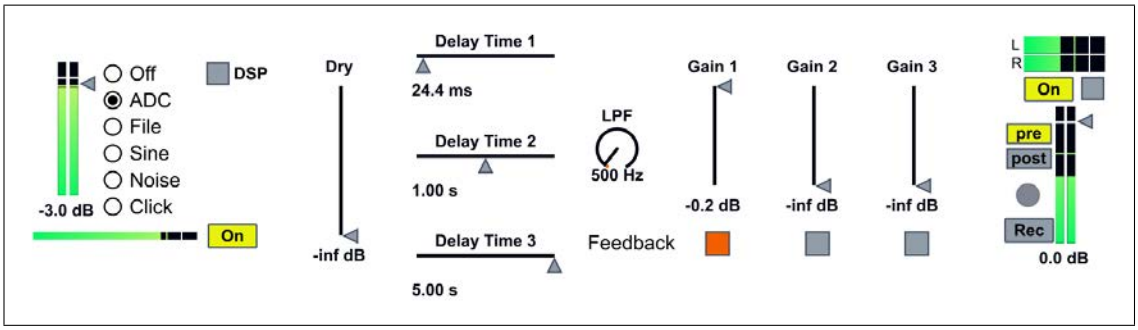


Figure 13.2: A simple delay line in Max.

the left, '*Reverb*' and '*Size*' control the reverb time and the size of a hypothetical room whose reverberation is being simulated (this latter setting is directly related to the pre-delay). The '*Pre-filter*' and '*Post-filter*' settings define the amount of damping that the signal is subject to during reflection. The '*Dry*', '*Early*' and '*Tail*' sliders set the loudnesses of the direct signal, the early reflections and the late reverberation, respectively. Finally, the '*Number of Reverbs*' tells the number of cascaded reverberation engines. Normally, this should be set to 1. Note that, after changing the number of reverbs, you have to re-send every reverberation parameter.

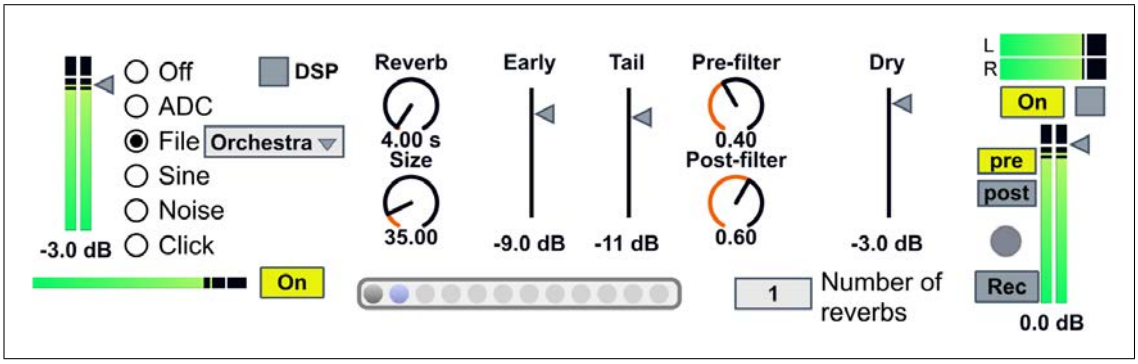


Figure 13.3: A simple cascaded reverberator in Max. Every reverberator (whose number is set by the 'Number of reverbs'*) obtains the same values.*

13.3 Exercises

1. Create a simple echo! Open `LEApp_13_01`, turn the gain of the first tape and the LPF to their maxima. Set a delay time for the first tape of approximately 200 ms. Now, select the ADC source and turn audio processing on. Speak into the microphone and listen the result. Avoid unwanted resonances by first adjusting the low-pass filter and, if that fails, by lowering the gain of the delay line. Experiment with different settings. What happens if you use more than one delay line simultaneously?

2. Delay lines offer the simplest way of creating real-time loops. In `LEApp_13_01`, turn off the dry signal and the first two delay lines and turn the LPF to its maximum. Turn Gain 3 to its maximum, set a delay time of a few seconds and enable the feedback

Open Access 2026-10-22

option. Set the audio input to ADC. If possible, attach a digital instrument to the line input, otherwise, use a microphone (in the latter case, you will need to turn off the input volume after recording the material to loop). Now, turn on audio processing and play (or sing) a short passage (shorter than the delay time). After listening to the result, try the effect with different delay times. Finally, try to create complex rhythmic patterns by turning on the other delay lines as well. Note that, in this case, the gains should be set slightly below 0 dB to avoid resonances. Take some time to find proper gain ratios. See if the gain settings that create a stable loop for a given set of delay times, work for other sets of delay times as well? Also, try how the LPF may affect the loop.

3. Delay lines play an important role in electronic sound desing. To prove this, create a digital jaw harp! Open `LEApp_13_01`. Enable feedback for the first delay line, set Gain 1 to -0.2 dB, Tape 1 to 21 ms and the LPF to 1.2 kHz. Turn off the dry signal and the other two delay lines. Select ADC as the audio input and use a microphone. Turn on audio processing. Set the sound output gain to a safe level where you don't experience feedback if you talk into the microphone. Now, tap on the microphone. If well done, the sound will be similar to a jaw harp. Experiment how changing the feedback gain, the delay time and the filter alters the timbre and/or the base pitch of your digital instrument. Finally, enrich the timbre of the sound by turning on the other two delay lines as well. Note that, to avoid resonances, you'll need to find a delicate balance between the gains and feedback times of the three delay lines.

4. Experiment with a basic reverberator. Open `LEApp_13_02` and observe the different presets (which you can select using the buttons at the bottom centre of the patch) with several different sound sources. Change each setting one-by-one (except the number of reverbs) and describe the changes.

5. *I am sitting in a room* (1969) is an iconic piece by Alvin Lucier. It consists of recording a text narrated by a performer. The recording is then played back and re-recorded in the same room, and this process is repeated many times. During this process, the room emphasizes some frequencies and attenuates others. Thus, the narrated text slowly transforms into a series of melodic/rhythmic gestures. By cascading reverberators with identical settings, one may achieve a similar result. Open `LEApp_13_02` and set the number of reverbs to 1. Select a preset. Select the ADC source, turn audio processing on and speak into the microphone. Listen the result. Repeat this process, increasing each time the number of reverbs by one. Do not forget to choose the same preset each time you change the number of reverbs. Take extreme care when setting the incoming and outgoing signal levels, as you might easily 'blow-up' the setup. Adjust the gains of the direct path, the early reflections or the late reverberation, if necessary.

6. Explore different settings of the `le_13_01_integra.integra` project!

Open Access 2026-10-22

Chapter 14

Panning, Surround and Multichannel Sound Projection

When translating a signal into the acoustic domain, one faces a problem unknown to acoustic instruments, as their amplification system literally comes 'out of the box'. When amplifying electronic signals, however, one should explicitly determine the way of projecting them into the acoustic space. There are basically two possibilities: a performer may decide to place the loudspeaker(s) at a single location in the performance venue, imitating 'traditional' instruments, where all sounds emerge from the instrument itself. Electronic musicians have another option as well, though: by using several loudspeakers, located at different positions within the venue, they may create sounds whose 'virtual source' lies at distinct locations from the performer. Moreover, with a multi-speaker setup it is not hard to change the positions of these 'virtual sources' in real-time, during the performance, allowing the creation of 'moving sounds'.

14.1 Theoretical Background

In the next few sections we present the most important concepts of sound projection. Although good diffusion systems (even a good stereo setup) would incorporate all of the principles referred to, we present the different theories in a section-by-section basis, illustrating them with the typical arrangements on which they are based. We do not discuss every phenomena and effect (e.g. the Haas- and Doppler-effects) related to the topic, though.

14.1.1 Simple Stereo and Quadro Panning

The easiest way of distributing a signal between the available loudspeakers is achieved by amplifying the sound in distinct amounts in each speaker. This is the basic principle for most simple setups, considering only a few additional effects. The way we distribute the sound affects both the spread of the signal and its virtual location: To obtain a narrow sound, almost all of the signal should be directed to a single speaker, while distributing the signal to all speakers in equal levels will be perceived as a wide source with an uncertain, inexact location.

A multichannel source (e.g. a stereo recording) gives us even more possibilities: by distributing every channel in the same way, we get a narrower result than if we distributed each

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

channel separately. This phenomenon, in the special case when diffusing stereo signals on two speakers, is described by the *stereo depth* (also called *stereo base*): by mixing the stereo signal into a mono one and distributing this mono signal across the speakers, we obtain a narrow result; conversely, by sending the two channels to two corresponding loudspeakers, the result is much wider.
In addition to the modification of amplitudes, filtering may affect the spatialisation of the sound as well. By applying a low-pass filter to a signal, we may simulate distance (since low frequencies are less attenuated, they can travel further than higher frequencies). Of course, finer control is possible if we send our signal through a reverberator before distributing it to the available loudspeakers.
The different loudspeaker channels are identified with numbers. The most common ordering for stereo (1–2) is Left–Right, while for quadro (1–4) it is Top Left–Top Right–Bottom Left–Bottom Right.

14.1.2 Simple Surround Systems

The ability to localise a sound depends on its frequency. While it is relatively easy to recognise (with eyes closed) the direction of high-frequency sources (e.g. a singing bird), it is more difficult for low-frequencies (e.g. a distant thunder). The reason is that the lower the frequency, the longer the wavelength of a signal. Thus, while adding multiple speakers to a venue helps the localisation of mid- and high-frequency signals, adding speakers will not enhance the localisation of low-frequency sounds. Since a speaker does not react the same to all frequencies, this consideration allows us to physically separate the speakers used for mid- and high-frequency sounds from the so-called subwoofers, amplifying only the low-frequency region.

14.1.3 Multichannel Projection

With an increased number of speakers, it is possible to create very realistic soundscapes. These setups usually require dedicated hardware (or separate, dedicated computers), dedicated halls, and dozens or hundreds of loudspeakers to operate properly. Besides the previously mentioned principles, these systems usually consider the phase relations of the loudspeakers in order to create the sound. The two most well-known theories of multichannel sound projection are *ambisonics* and *wave field synthesis*. They are quite similar in their nature, as both synthesize a realistic sound field with their virtual sources located *outside* the array of speakers. They differ mainly in two key properties. On the one hand, ambisonic spatialisation works perfectly only for listeners situated at the exact geometrical centre of the loudspeaker array, while the quality of WFS emulation does not depend on the position of the listener within the venue. On the other hand, some ambisonic systems can create perfect 3D localisation (i.e. the virtual source may be located anywhere), while WFS can only emulate virtual sources located on the same (horizontal) plane, thus allowing only 2D-localisation.

14.1.4 Musical Importance

Spatialisation, combined with proper reverberation (see Chapter 13), is our main means of influencing the *ambience* associated with the sound. Whether the listeners perceive the hall

Open Access 2026-10-22

as a cathedral, a cave or a cell; whether the sound arrives from the far distance or emerges just in front of their ears; whether its source is a point-like device, an instrument-sized object or a huge wall, it all depends on spatialisation and reverberation. For this reason, the more complex reverberators also include parameters regarding the spatialisation of their results and vice versa.

14.2 Examples

14.2.1 Spatialisation in Integra Live

The `le_14_01_integra.integra_` and `le_14_02_integra.integra_` files are downloadable using the following links: `http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_14_01_integra.integra`, `http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_14_02_integra.integra_`
The principle of amplitude modification is illustrated in the projects `le_14_01_integra.integra_` (see Figure 14.1) and `le_14_02_integra.integra_` (see Figure 14.2), respectively.

Figure 14.1: Stereo panning in Integra Live.

Both projects contain a soundfile player on the left side, able to loop the loaded sound files. By muting the soundfile player using the big toggle on the bottom left side, the line/microphone input will automatically turn on. The panning can be set using the big graphic controller on the right side, with two (stereo) or four (quadro) level meters, showing the actual signal strength on each loudspeaker. The overall loudness can be set with the big horizontal slider below the panning controller.

Open Access 2026-10-22

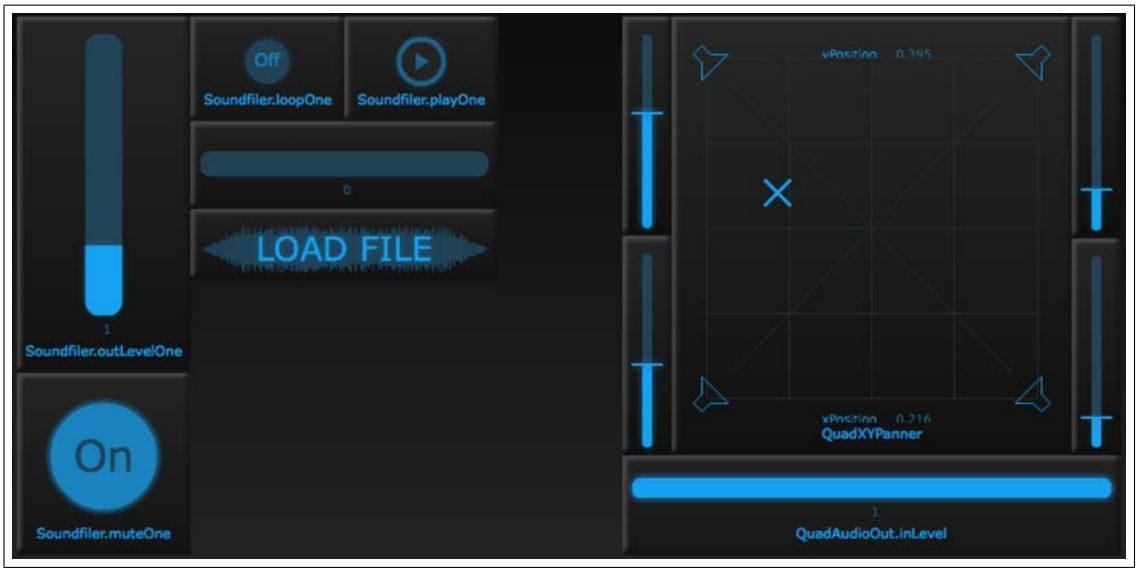


Figure 14.2: Quadraphonic panning in Integra Live.

14.2.2 Spatialisation in Max

`LEApp_14_01_` is downloadable for Windows and Mac OS X platforms using the following links: `http://www.typotex.hu/upload/Szigetvari_interaktivok/_SzigetvariAndrea_LiveElectronics_ENG/14_WIN.zip`, `http://www.typotex.hu/_upload/Szigetvari_interaktivok/SzigetvariAndrea_LiveElectronics_ENG/14_OSX.zip_`
Stereo sound projection is illustrated by `LEApp_14_01`, depicted in Figure 14.3. The 2D slider (which can be connected to MIDI controllers) controls stereo positioning (horizontal) and depth (vertical). When stereo depth is set to 0, the program mixes the input into a monophonic signal, which will be distributed between the two speakers according to the horizontal position of the 2D controller. By increasing the depth, the two channels separate from each other. Of course, if the source is monophonic, stereo depth will have no effect.

`LEApp_14_02` (see Figure 14.4) illustrates a basic ambisonic setup of 8 speakers. As a first step, one needs to set up the speaker positions in the top of the program window according to the real positions of the loudspeakers. The example contains four sources, which can be activated simultaneously. The positioning and the motion of these sources is controlled by the boxes below the respective input selectors.

The virtual position of the sources can either be set manually with the mouse or algorithmically. For the latter, one can choose from a variety of shapes (e.g. circle, line, eight etc.), which will be described by the algorithm when the '*start/stop*' toggle is pressed. The speed of the motion is controlled by the '*tempo*' number box (expressing the time elapsing between subsequent steps in milliseconds), and the type of trajectory (forward, reverse or both) can be set as well. Every control box contains a number of presets with pre-defined trajectories.

Open Access 2026-10-22

Figure 14.3: A simple stereo panner in Max. The 2D interface controls both the stereo positioning and depth.

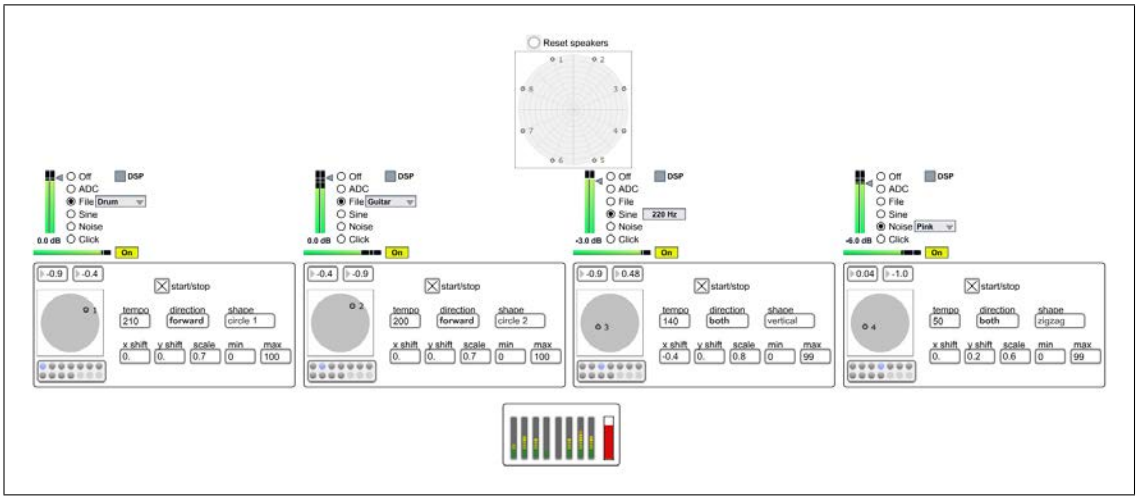


Figure 14.4: Ambisonics-based 8-channel projection in Max, using up to four individual monophonic sources.

Open Access 2026-10-22

14.3 Exercises

1. Compare the stereo panner of `le_14_01_integra.integra`_and `LEApp_14_01`_by trying them out with exactly the same sound samples! What differences can you hear? Can you reproduce the behaviour of the Integra Live project with the stand-alone application?

2. Compare the quadrophonic panner and the ambisonic example, similarly to the previous exercise!

3. Spatialise your own voice with the `LEApp_14_02`_application! After setting up the speakers, place the leftmost virtual source close to the position of one of the speakers. Turn on your microphone, and increase the output level slowly while speaking or singing into the microphone. Take care to avoid feedback! When you hear your voice from the chosen speaker, move the virtual source to at least two other speakers and test the level of the sound. If you experience feedback during this, lower either the input gain or the master volume. Once you have made sure that you do not get any feedback, choose a trajectory and start the automated motion of the virtual source. Sing or talk into the microphone and listen to the result. Try at least four different kinds of motion (e.g. fast, slow, linear, curved etc.).

4. A good spatialisation system contains properly configured filtering and reverberation devices, too. Create a better spatialiser based on the stereo panner of Integra Live! Open the original stereo panner (you should create a backup copy first) and go to Arrange View. Double-click on the Block to enter into Module View. Add a Low Pass Filter and a Tap Delay Module to the Block. Disconnect every incoming signal from the input of the Stereo Panner and connect these sources to the input of the Low Pass Filter instead. Connect the output of the Low Pass Filter to the input of the Tap Delay and the output of the Tap Delay to the input of the Stereo Panner. Add the '*delaytime*' and '*feedback*' controls of the Tap Delay and the '*frequency*' control of the Low Pass Filter to the Live View. Go back to Live View and try out the new tool. Find settings that imitate a speaker from the far distance on the middle-left of a big room and another one close to us in the right in a small room. Experiment with the settings in order to create unnatural pannings!

5. Upgrade the stereo panner in a different way! Open a fresh copy of the original stereo panner, go to Module View and add a Stereo Reverb Module. Disconnect the Stereo Panner from the Stereo Audio Out and insert the Stereo Reverb between them. After adding each relevant reverberation parameter (see Chapter 13 for more details), go back to Live View and test the new tool. Compare this device with the one created in the previous example by imitating the same two situations described above. How can these two setups be used?

Open Access 2026-10-22

Chapter 15

Mapping Performance Gestures

Mapping is a reinterpretation of information. This is needed when, for example, the raw controller data arriving from a sensor reaches the device controlled by it. The raw data needs to be adapted and scaled into a form and range that can be used by the 'instrument'. Mapping plays a role comparable to — if not even greater than — the choice of controllers in electronic instrument design, as they define the means by which a machine will interpret human gestures. Most gestural controllers are physically easy to operate — after all, anybody can change a slider or trigger a motion sensor. What makes them expressive and, at the same time, hard to master, is the way these gestures are turned into sound. Moreover, in some algorithmic pieces, the mappings form an integral part of the compositions themselves.

15.1 Theoretical Background

15.1.1 Mapping

Defining proper mappings is one of the most musically challenging tasks in the creation of a live electronic setup. Mapping strategies may be classified according to the number of relationships involved between gesture and device parameters:

One-to-one, when one gestural parameter controls one single parameter of a device (*example: a slider connected to an amplifier*).

One-to-many, when one gestural parameter controls multiple parameters of one or more devices (*example: a single motion, transformed into dozens of synthesis parameters*).

Many-to-one, when multiple gestural parameters control a single parameter of a device (*example: a synthesizer controlled by a wind controller, where the pitch may be influenced by both the fingering and the air pressure*).

Many-to-many, when multiple gestural parameters control multiple parameters of one or more devices (*example: musical systems where the incoming control parameters and the device parameters are linked by neural networks or similarly complex systems*).

 © Andrea Szigetvári, Ádám Siska,BKF

Open Access 2026-10-22

15.1.2 Scaling

In basic situations, data emerging from sensors can be directly interpreted by the modules receiving them. For example, a MIDI-based synthesizer may directly understand a MIDI pitch value arriving from a MIDI keyboard. However, in most practical scenarios, our data sources might not have a direct logical connection with the objects interpreting their values. It may happen, for instance, that we wish to use the pitch information arriving from the same MIDI keyboard (consisting of integer values between 0 and 127) to control an oscillator. However, as oscillators usually expect frequency values, we have to find a way to convert the MIDI value to a frequency first.
The simplest scaling is perhaps the linear one. In this case, we multiply our incoming values with a constant factor and add some offset value. Mathematically speaking, defining the factor and the offset value is equivalent to define an initial range and its scaled version. An example is when we need to convert a MIDI velocity value (range: 0–127) to an amplitude[1] value in the range 0–1: we only need to divide our incoming values by 127 (and there is no need to add any offset in this particular case). More general linear scaling include, for example, scaling the values of a MIDI controller to a stream of data defining the panoramics of the sound.
There are many non-linear scaling methods, including as simple ones as clipping an incoming data stream to as complex ones as artificial intelligence. Two commonly used non-linear scales include the MIDI-to-frequency and the decibel-to-amplitude (and their inverted forms: frequency-to-MIDI and amplitude-to-decibel), which are so-called 'exponential' scalings both. MIDI-to-frequency scaling will map the original MIDI-range 0–127 to frequencies between approx. 8 Hz and 12.5 kHz in a way that each integer MIDI value will correspond to the frequency of a certain pitch of the tempered twelve-tone scale.

15.2 Examples

15.2.1 Mappings in Integra Live

The main mapping feature of Integra Live is accessible through the '*Routing*' item of the Properties panel. This makes it possible to define *scalings* (which are one-to-one mappings) between the different module parameters. Although a routing always defines one-to-one mappings, by attaching multiple routings to the same attribute (or by writing scripts) it is also possible to create more complex mappings.
The `le_15_01_integra.integra_` file is downloadable using the following link: `http://www.typotex.hu/upload/Szigetvari_interaktivok/Szigetvari_Live_electronics/le_15_01_integra.integra_`

The project `le_15_01_integra.integra_`contains two identical blocks (their content is depicted in Figure 15.1). In both cases, the test source is a sine wave, and the frequency of this wave is routed into the frequency of the band-pass filter. Also, the frequency of the sine wave, the vibrato and the ring modulator, the amount of vibrato, ring modulation and distortion as well as the parameters of reverberation (source distance and early/tail balance)

[1] In fact, most synthesizers would use more complex mappings to convert MIDI velocities to amplitude values.